

ISCLS 2019

**Proceedings of the 6th International Sanskrit Computational
Linguistics Symposium**

23–25 October, 2019

Indian Institute of Technology Kharagpur
West Bengal, India

Sponsors



©2019 The Association for Computational Linguistics

Order copies of this and other ACL proceedings from:

Association for Computational Linguistics (ACL)
209 N. Eighth Street
Stroudsburg, PA 18360
USA
Tel: +1-570-476-8006
Fax: +1-570-476-0860
acl@aclweb.org

ISBN 978-1-950737-71-0

Introduction

Welcome to the 6th edition of the International Sanskrit Computational Linguistics Symposium (6th ISCLS) at IIT Kharagpur, West Bengal, India. The aim of ISCLS is to bring together researchers interested in any aspects of Sanskrit Computational Linguistics. Full papers were invited on original and unpublished research on various aspects of Computational Linguistics and Digital Humanities related to Sanskrit (Classical and Vedic), Prakrit, Pali, Buddhist Hybrid Sanskrit, etc. 13 contributions were accepted, and the final versions, after incorporating the reviewers' comments constitute the proceedings. We would like to thank the Program Committee for the 6th ISCLS for their reviewing efforts:

- Stefan Baums (University of Munich)
- Laxmidhar Behera (IIT Kanpur)
- Brendan Gillon (McGill University)
- Pawan Goyal (IIT Kharagpur)
- Olivier Hellwig (University of Zurich)
- Gérard Huet (Inria Paris)
- Amba Kulkarni (University of Hyderabad)
- Malhar Kulkarni (IIT Bombay)
- Pavan Kumar Satuluri (Chinmaya Vishwavidyapeeth, Veliyanad)
- Andrew Ollett (Harvard University, Cambridge, MA)
- Dhaval Patel (Ahmedabad)
- Ganesh Ramakrishnan (IIT Bombay)
- Peter Scharf (IIIT Hyderabad)
- Srinivasa Varakhedi (KKSU, Ramtek)
- K Varalakshmi (Osmania University, Hyderabad)

The first two papers talk about Sanskrit sentence generation and parsing. In "Sanskrit Sentence Generator", Amba Kulkarni and Madhusoodana Pai J present a sentence generator for Sanskrit, which takes an intermediate representation from which, using Panini's grammar, the desired sentence can be generated, without appealing to the world knowledge. In "Dependency Parser for Sanskrit Verses", Amba Kulkarni, Sanal Vikram and Sriram K describe their efforts to build a dependency parser which parses both prose as well as verse texts. The parser utilizes various constraints following traditional rules of verbal cognition, which are employed using and edge-centric binary join method.

The next two papers discuss the compound identification and type classification using word embeddings and machine learning methods. The paper, "Revisiting the Role of Feature Engineering for Compound Type Identification in Sanskrit" by Jivnesh Sandhan, Amrith Krishna, Pawan Goyal and Laxmidhar Behera, attempts to ask the question if the recent advances in neural networks can outperform traditional

hand engineered feature based methods on the semantic level multi-class classification task for Sanskrit. In "A Machine Learning Approach for Identifying Compound Words from a Sanskrit Text", Premjith B, Chandni Chandran V, Shriganesh Bhat, Soman Kp and Prabakaran P propose a classification framework for finding the compound words from a Sanskrit text, in particular, those found in Ayurveda text books, using word embeddings.

The next two papers talk about NLP corpus building. In "LDA Topic Modeling for pramāṇa Texts: A Case Study in Sanskrit NLP Corpus Building", Tyler Neill describes the methodology followed towards the preparation of digital corpus for word-level analysis. It also explains pitfalls in current digitalization practices of Sanskrit corpus. In "Vedavaapi: A Platform for Community-sourced Indic Knowledge Processing at Scale", Sai Susarla and Damodar Reddy Challa describe the architecture of an online platform for end-to-end indic knowledge processing addressing the challenges of composing independently developed tools for higher-level tasks, as well as employing human experts in the loop to work around the limitations of automated tools.

The next two contributions discuss the problems concerning information retrieval and questions answering from Sanskrit texts. The paper, "On Sanskrit and Information Retrieval" by Michaël Meyer discusses the challenges for traditional information retrieval systems to handle the peculiarities of Sanskrit, and discusses a few possible solutions. In "Framework for Question-Answering in Sanskrit through Automated Construction of Knowledge Graphs", Hrishikesh Terdalkar and Arnab Bhattacharya target the problem of building knowledge graphs for particular types of relations from Sanskrit texts and attempts to answer factoid questions using the extracted relations.

The next two papers discuss digital tools for Sanskrit Wordnet and Vaijayantīkośa. In "Introduction to Sanskrit Shabdmitra: An Educational Application of Sanskrit Wordnet", Malhar Kulkarni, Nilesh Joshi, Sayali Khare, Hanumant Redkar and Pushpak Bhattacharyya introduce Sanskrit Shabdmitra, a digital tool based on Sanskrit Wordnet, for learning and teaching Sanskrit. The paper, "Vaijayantīkośa Knowledge-Net" by Aruna Vayuvegula, Satish Kanugovi, Sivaja S Nair, Shivani V and Mahalakshmi discusses Vaijayantīkośa Knowledge-Net, a web-based tool for easy access and analysis of words in Vaijayantīkośa, a Sanskrit lexicon containing words from spoken language as well as those in Vedic literature.

The next two contributions attempt to capture the evolution of manuscript texts. The paper, "Utilizing Word Embeddings based Features for Phylogenetic Tree Generation of Sanskrit Texts" by Diptesh Kanojia, Abhijeet Dubey, Malhar Kulkarni, Pushpak Bhattacharyya and Reza Haffari infers phylogenetic trees of Sanskrit texts using inter-manuscript distances obtained via word embeddings. In "An Introduction to the Textual History Tool", Diptesh Kanojia, Malhar Kulkarni, Pushpak Bhattacharyya, Sayali Ghodekar, Irawati Kulkarni, Nilesh Joshi and Eivind Kahrs describe textual history tool to capture the historical view of the transmission of a text through the manuscript tradition, captured via inter-related data from various types of related texts.

The proceedings conclude with the paper, "Pāli Sandhi – A Computational Approach" by Swati Basapur, Shivani V and Sivaja Nair, which discusses complexities involved in creating a computational grammar for Sandhi tools in Pāli language.

ISCLS 2019 has received financial support from Dharohar, Indic-Academy and DST-SERB.

The conference also hosts two keynote talks by Prof. Rajeev Sangal and Prof. Korada Subrahmanyam. Further, various demo submissions are also presented at the conference.

We very much hope that you will have an enjoyable and inspiring time at the conference!

Pawan Goyal
Indian Institute of Technology, Kharagpur, WB, India
October 2019

Table of Contents

Sanskrit Sentence Generator	2
<i>Amba Kulkarni and Madhusoodana Pai</i>	
Dependency Parser for Sanskrit Verses	15
<i>Amba Kulkarni, Sanal Vikram and Sriram K</i>	
Revisiting the Role of Feature Engineering for Compound Type Identification in Sanskrit	29
<i>Jivnesh Sandhan, Amrith Krishna, Pawan Goyal and Laxmidhar Behera</i>	
A Machine Learning Approach for Identifying Compound Words from a Sanskrit Text	46
<i>Premjith B, Chandni Chandran V, Shriganesh Bhat, Soman Kp and Prabakaran P</i>	
LDA Topic Modeling for pramāa Texts: A Case Study in Sanskrit NLP Corpus Building	53
<i>Tyler Neill</i>	
Vedavaapi: A Platform for Community-sourced Indic Knowledge Processing at Scale	69
<i>Sai Susarla and Damodar Reddy Challa</i>	
On Sanskrit and Information Retrieval	84
<i>Michaël Meyer</i>	
Framework for Question-Answering in Sanskrit through Automated Construction of Knowledge Graphs	98
<i>Hrishikesh Terdalkar and Arnab Bhattacharya</i>	
Introduction to Sanskrit Shabdmitra: An Educational Application of Sanskrit Wordnet	118
<i>Malhar Kulkarni, Nilesh Joshi, Sayali Khare, Hanumant Redkar and Pushpak Bhattacharyya</i>	
Vaijayantīkośa Knowledge-Net	135
<i>Aruna Vayuvegula, Satish Kanugovi, Sivaja S Nair, Shivani V and Mahalakshmi</i>	
Utilizing Word Embeddings based Features for Phylogenetic Tree Generation of Sanskrit Texts	153
<i>Diptesh Kanojia, Abhijeet Dubey, Malhar Kulkarni, Pushpak Bhattacharyya and Reza Haffari</i>	
An Introduction to the Textual History Tool	167
<i>Diptesh Kanojia, Malhar Kulkarni, Pushpak Bhattacharyya, Sayali Ghodekar, Irawati Kulkarni, Nilesh Joshi and Eivind Kahrs</i>	
Pāli Sandhi – A computational approach	182
<i>Swati Basapur, Shivani V and Sivaja S Nair</i>	

Full Papers

Sanskrit Sentence Generator

Amba Kulkarni & Madhusoodana Pai J

Department of Sanskrit Studies

University of Hyderabad

apksh.uoh@nic.in, jmadhusoodan@gmail.com

Abstract

In this paper we describe a sentence generator for Sanskrit. Pāṇini's grammar provides the essential grammatical rules to generate a sentence from its meaning structure. The meaning structure is an abstract representation of the verbal import. It is the intermediate representation from which, using Pāṇini's rules, without appealing to the world knowledge, the desired sentence can be generated. At the same time, this meaning structure also represents the dependency parse of the generated sentence.

Keywords: Sanskrit, Sentence Generator, Pāṇini, Paninian Grammar, Computational Linguistics.

1 Introduction

Natural language generation (NLG) is the process of generating text from a meaning representation. It may be thought of as the reverse of natural language understanding (NLU). There has been considerably less focus in NLG than in NLU. Nevertheless, a generator is an essential component of any machine translation (MT) system. It is also needed in systems such as information summarization, question answering, etc. NLG systems are also being used by human writers to make the writing process efficient and effective (Galitsky, 2013). In the field of computational creativity, the interest does not lie any more on how a computer can generate creative pieces on its own but rather how such systems can be used to assist a person in a creative task. Poem machine by Hämäläinen () is an example of an online tool to generate Finnish poetry with a computationally creative agent. Automatic advertisement slogan generators (Iwama and Kano, 2018) are being used by Japanese.

NLG is also useful for second language learners. Second language learners can use such modules to generate sentences in a controlled way and learn the language at their own pace. For a classical language like Sanskrit which is for most of the people a second language and not the mother tongue, a computational aid can help a user in several ways. Some of the aspects where such an aid would be useful are listed below.

- Sanskrit is an inflectional language. That means the case suffixes (vibhakti-pratyayas) get attached to the stem (prātipadika/dhātu) and during the attachment some morpho-phonetic changes also take place. In some cases, one can't tell apart the stem and its suffix. This increases the load on memorization.
- Each Sanskrit noun has a gender which is independent of the sex or animacy of the referent. In Sanskrit, gender is an integral part of the nominal stem (prātipadika). That means one has to remember the gender of each nominal stem since the word forms differ with gender as well. The gender has no relation to the meaning/denotation of the word. For example wife in Sanskrit can be either a *patnī* in feminine gender or *dārā* in masculine gender or *kalatra* in neuter gender.
- The participants of an action are termed *kāraṅkas*. The definitions of these *kāraṅkas* are provided by Pāṇini which are semantic in nature. However, the exceptional cases make them syntactico-semantic. For example, in the presence of the prefix *adhi* with the verbs

śñ̄n, *sthā* and *as*, the locus instead of getting the default adhikaraṇaṃ role gets a karma (goal) role and subsequently accusative case marker, as in *sah grāmam adhiṣṭhati* (He inhabits/governs the village) where *grāma* gets a karma role, and is not an adhikaraṇaṃ.

- There are a set of words in whose presence a nominal stem gets a specific case marker. For example, in the presence of *saha*, the accompanying noun gets instrumental case suffix. The noun denoting the body part causing the deformity also gets an instrumental case suffix as in *akṣṇā kāṇaḥ* (one-eyed). Most of these rules being language specific, the learner has to remember all the relevant grammar rules.
- Sanskrit has a natural tendency to use passive (karmaṇi) with transitive verbs and impersonal passive (bhāve) with intransitive verbs. If the native language of a learner does not permit such usages, s/he finds it difficult to understand/construct sentences with such usages.
- There are also cases where the verbs in different pada (ātmanepada/ parasmaipada) have different meanings. A speaker, by mistake, if uses a wrong pada, the sentence may not convey the desired meaning. For example, the verb *bhuj* from *rudhādi-gaṇa* when used in the meaning of eating is always in ātmanepada while in the sense of *to rule* or *to govern* it is used in parasmaipada.¹
- In the causative constructions, the semantics associated with certain participants is different for different sets of verbs. For example, for the verbs denoting motion, the causer is also a karman with respect to the causative action. And then in such cases, even a person who has studied grammar well gets confused in assigning proper case marker to the verbs. The confusion grows more if the sentence is to be expressed in passive voice.

All these problems make the life of a Sanskrit speaker difficult. Even if a person has passive control, due to the above-mentioned problems, he either shies away from speaking / writing Sanskrit or ends up in speaking /writing wrong Sanskrit. Finally, the influence of mother tongue on Sanskrit speaking also results in wrong/nativized Sanskrit. A speaker who does not want to adulterate Sanskrit with the influence of his/her native language would like to have some assistance, and if it were by a mechanical device such as a computer, it would be advantageous.

With these problems in mind, and also the possible applications in computational linguistics as mentioned above, we decided to build a Sanskrit sentence generator.

2 Approaches

Natural language generation is comparatively easier to handle than natural language understanding. NLU involves handling of ambiguities, whereas the main problem in NLG is selection of appropriate lexicon and syntax for expressions. In the late nineties of the last millennium, several NLGs were developed which were general purpose (Dale, 2000). But they were difficult to adopt to small task oriented applications. Two different methods were used to develop NLGs - rule based and template based. A rule based system can generate sentences without any restriction, provided the rules are complete. A template based generation on the other hand is delimited in its scope by the set of templates. A programme that sends individualized bulk mails is an example of template based generation. There have been efforts to mix the use of rule based and template based generation. The recent trend in NLG, as with all other NLP systems is to use machine learning algorithms using large databases.

With the availability of a full-fledged generative grammar for Sanskrit in the form of Aṣṭādhyāyī, it is appropriate to use a rule based approach for building the generation module. A lot of work in the area of Sanskrit Computational linguistics has taken place in the last decade, some of which is related to the word generators. So we decided to use the existing word generators and build a sentence generator, modelling only the sūtras that correspond to the assignment of case markers.

In the next section, we discuss our approach to building a sentence generator using rules

¹bhujo'navane(1.3.66)

from *kāraka* and *vibhakti* sections of Pāṇini’s *Aṣṭādhyāyī*. In the fourth section, we provide the implementation details. In the fifth section we discuss the interface while the usability of the sentence generator is reported in the last section.

3 Sentence Generator: Architecture

Pāṇini has given a grammar which is generative in nature. He presents a system of grammar that provides a step by step procedure to transform thoughts in the minds of a speaker into a language string. Broadly speaking one may imagine three mappings in the direction from semantics to phonology ((Bharati et al., 1994), (Kiparsky, 2009)). These levels are represented pictorially as in Figure 1.

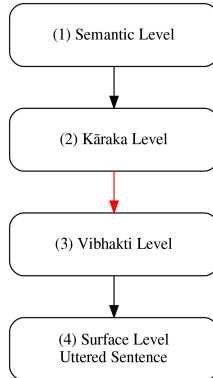


Figure 1: Levels in the Pāṇinian model

3.1 Semantic Level

This level corresponds to the thoughts in the mind of a speaker. The information is still at the conceptual level, where the speaker has identified the concept and has concretised them in his mind. The speaker, let us assume, for example, has witnessed an event where a person is leaving a place and is going towards some destination. For our communication, let us assume that the speaker has identified the travelling person as *person#108*, the destination as *place#2019*, and the action as *move-travel#09*. Also the speaker has decided to focus on that part of the activity of going where the *person#108* is independent in performing this activity, and that the goal of this activity is *place#2019*. This establishes the semantic relations between *person#108* and *move-travel#09* as well as between *place#2019* and *move-travel#09*. Let us call these relations *sem-rel#1* and *sem-rel#2* respectively. This information at the conceptual level may be represented as in Figure 2.

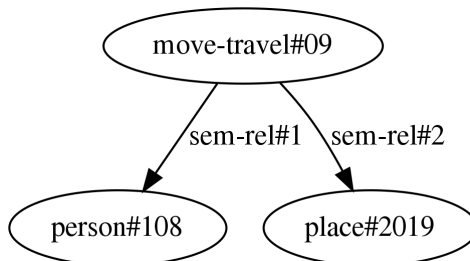


Figure 2: Conceptual representation of a thought

3.2 Kāraka Level

In order to convey this, now the speaker chooses the lexical items that are appropriate in the context from among all the synonyms that represent each of these concepts. For example, for the person#108, the speaker chooses a lexical term, say *Rāma*, among the synonymous words {*ayodhyā-pati*, *daśarathanandana*, *sītā-pati*, *kausalyā-nandana*, *jānakī-pati*, *daśa-ratha-putra*, *Rāma*, ...}. Similarly corresponding to the other two concepts, the speaker chooses the lexical terms say *vana* and *gam* respectively. With the verb *gam* is associated the pada and gaṇa information along with its meaning.

Having selected the lexical items to designate the concepts, now the speaker chooses appropriate kāraka labels corresponding to the semantics associated with the chosen relations. He also makes a choice of the voice in which to present the sentence. Let us assume that the speaker in our case decides to narrate the incidence in the active voice. The sūtras from Aṣṭādhyāyī now come into play. The semantic roles sem-rel#1 and sem-rel#2 are mapped to kartā and karma, following the Pāṇinian sūtras

- svatantraḥ kartā(1.4.54); which assigns a kartā role to Rāma.
- karturīpsitatamaṁ karma(1.4.49); which assigns a karma role to vana.

Let us further assume that the speaker wants to convey the information as it is happening i.e., in the present tense (vartamāna-kāla). Thus at the end of this level, the available information is as shown in Figure 3.

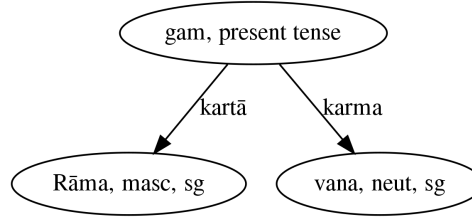


Figure 3: Representation in abstract grammatical terms

This information is alternately represented in simple text format as shown below.

word index	stem	features	role
1	Rāma puṃ	eka	kartā 3
2	vana napuṃ	eka	karma 3
3	gam parasmaipada bhvādi	vartamāna	kartari

The first field represents the word index which is used to refer to a word while marking the roles. The second field is the stem (with gender in case of nouns), the third field provides morphological features such as number, tense, etc. and the fourth field provides the role label and the index of the word with respect to which the role is marked.

3.3 Vibhakti Level

Now the sūtras from vibhakti section of Pāṇini's Aṣṭādhyāyī come into play. Vana which is a karma, gets accusative (dvitīyā) case marker due to the sūtra *karmaṇi dvitīyā (anabhihite)* (2.3.2). Since the sentence is desired to be in active voice, kartā is abhihita (expressed), and hence it will get nominative (prathamā) case due to the sūtra - *prātipadikārtha-liṅga-parimāṇa-vacana-mātre prathamā*(2.3.46). The verb gets a laṭ lakāra due to vartamāna-kāla (present tense) by the sūtra - *vartamāne laṭ*(3.2.123). It also inherits the puruṣa (person) and vacana (number) from the kartā *Rāma*, since the speaker has chosen an active voice. Thus at this level, now, the information available for each word is as follows.

word index	stem	features	role
1	Rāma puṃ	eka	kartā 3
2	vana napuṃ	eka	karma 3
3	gam ₁	vartamāna	kartari

Table 1: Input to Sentence Generator

word index	stem	morphological features
1	Rāma puṃ	eka prathamā
2	vana napuṃ	eka dvitīyā
3	gam parasmaipada bhvādi	laṭ prathama eka

3.4 Surface Level

With this information, now each pada is formed using the available word generator. Sandhi at the sentence level is optional. If the speaker intends, then the sandhi rules come into play and a sentence with sandhi is formed. Thus we get either *Rāmaḥ vanaṃ gacchati* or optionally *Rāmo vanaṃgacchati* as an output.

3.5 Sentence Generation: Input and Output

In the above architecture, there are three modules:

1. A module that maps the semantic information in the form of abstract concepts and abstract semantic relations into the linguistic elements viz. the nominal / verbal stem and syntactico-semantic relations

We have not implemented this module yet. However we have conceptualised it as follows. A user interface is planned, to model this part, through which the speaker selects the proper lexical terms as well as declares his intention selecting the syntactico-semantic relations and the voice. The gender associated with the nominal stem is provided by the interface, and the user does not have to bother about it. The user only provides the nominal stem, chooses the number and its role with respect to the verb. In the case of verbs, the user selects the verb based on its meaning, and the information of pada and gaṇa is automatically picked by the interface, coding this information in the form of a subscript. User also chooses appropriate relations between the words. The user interface takes care of exceptional cases hiding the language specific information from the user. The output of this module is, for the example sentence under discussion, is as shown in the Table 1.

2. A module that maps the syntactico-semantic relations to the morpho-syntactic categories such as case marker and position (in the case of upapadas, for example)

In this paper we describe this second module in detail that maps the syntactico-semantic relations into morpho-syntactic categories. The input to the generator is a set of quadruplets as shown in the Table 1. The first element provides the index, the second the stem, the third the morphological features and the last one the relation and the index of the second relata (viz. anuyogin). The current version recognises only the following expressions for stem-feature combinations, where '?' represents optionality, '*' is the Kleene operator for zero or more occurrences.

- (a) {Noun}{Taddhita}?{Gender}{Vacana}?
- (b) {Upasarga}*{Verb}{Sanādi_suffix}{Kṛt_suffix}{Vacana}?
- (c) {Upasarga}*{Verb}{Sanādi_suffix}{prayoga}{lakāra}

Number and Gender are not specified if it has an adjectival relation with other word.

This representation is the same as the internal representation of the output of the Samsādhanī² parser. We call this representation, an intermediate form, or the meaning structure. It represents the verbal import of the sentence in abstract form, hiding the details of which

²<http://scl.samsaadhanii.in/scl>

linguistic unit codes what information.

3. A module that composes a surface form/word form from the morphological information. This third module corresponds to the word generation. Given the morphological information, this module produces the correct form of the word. For this module, the word-generator developed in-house³, which is also a part of Samsādhanī tools is being used. We decided to produce the output in unsandhied form. Hence, for this example, the output would be

Rāmaḥ vanaṃ gacchati.

The focus of this paper is on the second module viz. morphological spellout rules.

4 Morphological spellout module

There are 3 major tasks that are carried out in this module.

1. Assigning case marker to the substantive based on its syntactico-semantic role,
In Pāṇini's grammar we come across 3 different types of case marker assignment. They are
 - (a) case marking for a kāraka relation,
 - (b) case marking in the presence of certain words called upapadas,
 - (c) case marking expressing the noun-noun relationsAll these sūtras are found in the third section of the second chapter of Aṣṭādhyāyī from 2.3.2 till 2.3.50.
2. Inheriting morphological features of the adjectives from their heads, and
3. Assigning morphological features for finite verbs such as person and number, and
4. Assigning lakāra corresponding to the tense, aspect and modality of the verb.

Now we explain each of these steps below.

4.1 Assigning case marker

For generating the substantial forms, we need the case marker corresponding to the kāraka role. The default cases for kartā, karma, karaṇaṃ, sampradānaṃ, apādānaṃ and adhikaraṇaṃ are 3,2,3,4,5,and 7 respectively, provided the kāraka is an-abhihita (not expressed). When the kartā (karma) is expressed by the verbal suffix, then kartā (karma) gets the nominative case suffix by *prātipadikārthaliṅgaparimāṇavacanamātre prathamā*(2.3.46). Similarly, in the case of causatives, the case markers get decided based on the semantics of the verbal roots. For example, the sūtra *gatibuddhipratyavasānārthaśabdakarmākarmakāṇāmaṇi kartā sa ṇau* (1.4.52) assigns a karma role and hence accusative case suffix to the prayojya-kartā, if the verb has one of the following meaning - motion, eating, knowledge or information related, or it is a verb with literary work as a karma or it is an intransitive verb. We have summarized all these rules in Appendix A.

For other kārakas viz. karaṇaṃ, sampradānaṃ, apādānaṃ and adhikaraṇaṃ, the case assignment is pretty straightforward. However, there is some problem, from the user's perspective, in the selection of a kāraka. We illustrate this problem with examples.

1. In the presence of the prefix *adhi* with the verbs *śñi*, *sthā* and *as*, the locus instead of getting the default adhikaraṇaṃ role, gets a karma (goal) role, as in *saḥ grāmam adhiṣṭhati* (He inhabits/governs the village) where *grāma* gets a karma role, and is not an adhikaraṇaṃ. Now this is an exception to the rule, and only the native speaker of Sanskrit might be aware of this phenomenon. The user, based on his semantic knowledge, would consider *grāma* a locus, and the generator then will fail to generate the correct form.
2. Another problem is with cases of exceptions under apādānaṃ and sampradānaṃ. For a verbal root *bhī* to mean *to be afraid of*, according to Pāṇini's grammar, the source of fear is termed apādānaṃ. But this is not obvious to a user who has not studied Pāṇini's grammar. He may treat it as a cause. Similarly, in the case of motion verb *gam*, the destination, according to the Pāṇini's grammar is a karma, but due to the influence of native language such as Marathi or Malayalam, the speaker may think it as an adhikaraṇaṃ.

³<http://scl.samsaadhanii.in/scl>

Another case is of the relation between two nouns such as part and whole, kinship relations, or relation showing the possession, as in *vrkṣasya śākhā* (the branches of a tree), *Daśarathasya putraḥ* (son of Dasharatha) and *Rāmasya pustakam* (Rama’s book). In all these cases Sanskrit uses a genitive case. Pāṇini does not discuss the semantics associated with all such cases, neither he proposes any semantic role in such cases. He deals with all such cases by a single rule *ṣaṣṭhī śeṣe* (2.3.50) assigning a genitive case in all the residual cases. While for analysis purpose, it is sufficient to mark it as a generic relation, for the generation purpose, the user would like to specify the semantics associated with it as part-and-whole-relation, or kinship, etc.

Hence in all such cases, we plan⁴ to provide templates of expectancies for such verbs and internally they are mapped to the Pāṇinian labels. The set of tags providing the role labels and other relations are provided in Appendix A. These tags were found to be appropriate for both analysis as well as generation (Kulkarni, 2019). This tagset essentially consists of the *kāraka* roles which account for the direct participants in the activity, other tags such as *hetu* (cause), *prayojanamī* (purpose), *kriyāviśeṣanamī* (adverb), etc. which indicate the modifiers of the action, tags such as *pūrvakāla* (precedence) showing the relation between sub-ordinate clause with the main clause, and tags marking the relations between nouns such as *adjectival* relation, etc. All these relations are semantic in nature.

One more set of relations between nouns is due to the *upapadas* (accompanying words). In the presence of an *upapada*, the accompanying word gets a specific case marker. For example, in the presence of *saha*, the accompanying word gets an instrumental case. This is again language specific, and hence non-native speakers of Sanskrit may go wrong in speaking sentences that involve *upapadas*. Pāṇini has not provided any semantic interpretation associated with such *upapadas*. (Kulkarni, 2019) has provided a semantic classification of these *upapadas* (See Appendix A).

Handling Causatives: In Sanskrit a causative suffix (*ṇic*) is added to the verbal root to change the sentence from non-causative to causative. In *kartari ṇic prayoga*, the *prayojakakartā* being expressed by the verbal suffix gets nominative case. If the verb is transitive, the *karma* gets *dvitīyā vibhakti* by *anabhihite karmaṇi dvitīyā*. The *prayojyakarma* however behaves in a different way with different verbs. Next, in the case of *karmaṇi ṇic prayoga*, *karma* being *abhihita* gets nominative case and *prayojakakartā* gets instrumental case. Now when the verb is *dvikarmaka*, which of the two *karmas* is expressed and which is unexpressed is decided on the basis of the verbal root. In the case of verbal roots *duh*, *yāc*, *pac*, *daṇḍ*, *rudhi*, *pracchi*, *chī*, *brū*, *śāsu*, *jī*, *math*, *muṣ* *mukhyakarma* gets accusative case and *gaṇakarma* gets nominal case. In the case of verbal roots *nī*, *hr*, *kṛṣ*, *vah* *gaṇakarma* gets accusative case and *mukhyakarma* gets nominal case⁵. Following Pāṇini’s grammar, we have classified the verbs into semantic classes as below.

- *akarmaka* (intransitive)
- *sakarmaka* (transitive)
 - verbs in the sense of to motion, knowledge or information, eating and the verbs which have literary work as their object
 - * verbs in the sense of motion
- *dvikarmaka* (ditransitive)-type 1
- *dvikarmaka* (ditransitive)-type 2

This list then takes care of the proper *vibhakti* assignment in all the type of *causatives*. See AppendixA for the summary of all rules.

4.2 Handling adjectives

Consider the following input to the system, which has *viśeṣaṇa* in it.

⁴The work is in progress, and hence is not being reported.

⁵*pradhānakarmaṇyākhyeye lādīnāhurdvīkarmaṇām . apradhāne duhādīnām ... (akathitaṃ ca (Mahābhāṣyam))*

word index	stem	features	role
1	vīra		viśeṣaṇam 2
2	Rāma puṃ	eka	kartā 3
3	vana napuṃ	eka	karma 3
4	gam ₁	vartamāna	kartari

Table 2: example with adjective

Note here that no morphological features have been provided for the viśeṣaṇam. In order to generate the correct word form of the word vīra, we need its gender, number, and case (liṅga, vacana, vibhakti). Only information available to the generator from the user that *vīra* is a viśeṣaṇam of the second word. The required information is inherited from the parent node i.e. the viśeṣya. If the adjective is a derived participle form of a verb, which itself may have kāraka expectancies, we provide the necessary verbal root and the participle suffix also as input parameters for generation. For example, in Table 3, vyūḍham is an adjective of pāṇḍavānikam, and the stem and the features for it are provided as *vi+vah1* and *bhūtakarṇa* respectively.

4.3 Handling finite verbs

In the case of verb form generation, the verb form generator needs the information of

- pada,
- gaṇa,
- puruṣa,
- vacana, and
- lakāra.

to generate the verb form.

Pāṇini has given sūtras to assign lakāras for different tense and mood. For example *-vartamāne lat(3.2.123)*. These sūtras are implemented as a hash data structure that maps the tense and mood to the lakāra. The voice determines the person and number of the verbal form. If the voice is kartari (karmaṇi), then the person and number information is inherited from the kartā(karma). In the case of impersonal passive (bhāve), the person and number are assigned the values third (prathama-puruṣa) and singular(eka-vacana) respectively. A note on the information of puruṣa is in order. As we notice, the information of person is not provided with a noun stem in the input. Then from where does the machine get this information? Here we use Pāṇini’s sūtras:

- yuṣmadyupapade samānādhikaraṇe sthāninyapi madhyamaḥ(1.4.105).
- asmadyuttamaḥ(1.4.107).
- śeṣe prathamamaḥ(1.4.108).

Next comes the information about pada and gaṇa. We notice that, though the majority of the verbs belong to a single gaṇa, there are several dhātus which belong to more than one gaṇa. For example the very first dhātu in the dhātupāṭha viz *bhū* belongs to two different gaṇas viz bhvādi and curādi. It is the meaning which distinguishes one from the other. *Bhū* in bhvādigāṇa is in the sense of sattāyām (to exist) and the one in the curādigāṇa is in the sense of prāptau (to acquire). A detailed study of the verbs belonging to different gaṇas is carried out by (Shailaja, 2014). She has indexed these dhātus for distinction. The verb generator of Saṃsādhani uses these indices to distinguish between these verbs. The speaker, on the other hand, would not be knowing these indices. So we provide a user interface to the user wherein the user can select the dhātu, gaṇa and its meaning, and the interface assigns a unique desired index automatically.

If a verb has ubhayapada both the parasmaipada and ātmanepada forms would be generated. Otherwise only the form with associated pada would be generated. Certain verbs use different padas to designate different meanings. For example, the verb *bhuj* has two meanings viz. *to eat* and *to rule* or *to govern*. In the sense of *to eat*, the verb has only ātmanepada forms and in the sense of *to govern*, it has only parasmaipada forms. In such cases, the user interface hides all

these complexities from the user.

4.4 Evaluation

In order to evaluate the coverage, a list of around 1000 sentences is manually collected covering a wide range of syntactic phenomenon and also verbs with different expectancies. Each sentence is parsed with the available parser and the parsed output, which is the same as the meaning representation or the semantic input for the generation, is manually verified. This semantic representation is given to the generator as an input.

There were a few challenges in the evaluation. In the absence of a taddhita (secondary derivatives) word generator, we provide the nominal stem formed by affixing the taddhita suffix. For example, we directly provide the stem śaktimat instead of śakti + matup. Similarly in the absence of a handler for feminine suffix, we provide the stem formed after the addition of feminine suffix as in anarthā (which is formed by adding a feminine suffix to anartha). In order to handle the out of vocabulary words, we developed a morphological analyser that assigns the default paradigm for the generation of such words.

5 Sanskrit Sentence Generator: Interface

The Graphical User Interface (GUI) of the Sanskrit Sentence Generator facilitates a user to provide the required input in a prescribed form. As mentioned earlier, all the language specific details such as the gaṇa, pada information of a verb, or the gender of a nominal stem are hidden from the user. The user just selects the appropriate nominal / verbal stem and the grammatical relations among the words. Figure 4 shows the generator interface for the following input.

word index	stem and features	relation
1	ḍṛś1	pūrvakālah 11
2	tu	sambandhaḥ 1
3	pāṇḍava-ānīka {puṃ eka}	karma 1
4	vi+vah1 {bhūtakarma}	viśeṣaṇam 3
5	duryodhana {puṃ eka}	kartā 11
6	tadā	kālādhikaraṇam 11
7	ācārya {puṃ eka}	karma 8
8	upa_sam+gam1	pūrvakālah 11
9	rājan	abhedah 5
10	vacana {napuṃ eka}	karma 11
11	brū1 {anadyatanabhūtaḥ}	kartari

Table 3: Input for the generator

We have also provided another interface. This interface takes the input from the Sanskrit parser. It allows us to test the completeness of both parser as well as the generator at the sentence level. This interface takes the machine internal representatin of the parser’s output (which is the same as shown in the Table 1) and feeds it to the generator. The overall architecture of our generator (and parser) is as shown in Figures 5 and 6.

6 Conclusion

Pāṇini’s grammar provides a grammatical framework for generation. While the complexity of Sanskrit generation lies at the word level, the sentence generation is pretty straightforward. The only challenge in designing the generator was in deciding the granularity of the semantic relations appropriate for both analysis and generation. We wanted to make sure that the grammatical relations used are universal in nature, without carrying any baggage of the language idiosyncrasy. Having confirmed that this tagset is appropriate for both generation and analysis (Kulkarni, 2019), we can now open it for other languages as well; to start with the Indian languages. Now

Sanskrit Sentence Generator (v04)

- 1 दृशुँ पूर्वकालः 11
- 2 तु सम्बन्धः 1
- 3 पाण्डवानीकं पुं एक कर्म 1
- 4 विन्ध्यह् भूतकर्म विशेषणम् 3
- 5 दुर्योधनं पुं एक कर्ता 11
- 6 तदा कालाधिकरणम् 4
- 7 आचार्यं पुं एक कर्म 8
- 8 उप_सम्भ्रम् पूर्वकालः 11
- 9 राजन् अभेदः 5
- 10 वचनं नपुं एक गौणकर्म 11
- 11 श्रुः कर्त्तरि अनद्यतनभूतः

दृष्ट्वा तु व्यूढम् पाण्डवानीकम् राजा दुर्योधनः तदा आचार्यम् उपसङ्गम्य वचनम् अब्रवीत् (पा।प।)

दृष्ट्वा तु व्यूढम् पाण्डवानीकम् राजा दुर्योधनः तदा आचार्यम् उपसङ्गम्य वचनम् अब्रूत् (आ।प।)

संश्लेषणं दृश्यताम्

Figure 4: Generation of a Shloka from its analysis

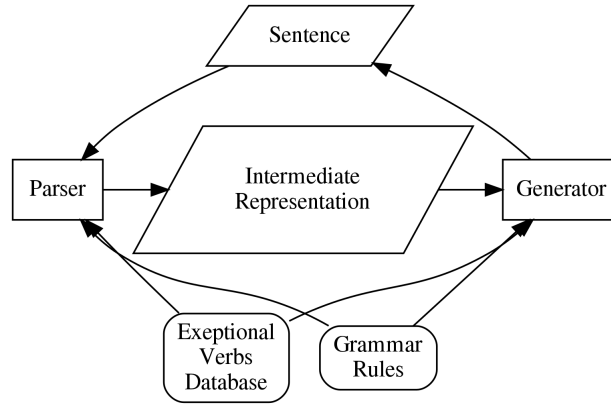


Figure 5: parser-generator: inverse operations

we are in the process of designing a user interface that hides the language and grammar specific details from the user and allows him to provide the input purely in semantic form.

Having said this, now we list some advantages and limitations of our generator.

1. This generator can be plugged in to a machine translation system.
2. It acts as a useful aid to the non-native speakers of Sanskrit to write in Sanskrit effectively guaranteeing grammatically correct sentences.
 - One need not memorize the word forms and the gender of the nominal stems
 - No need to remember all the special rules assigning case suffix to a noun representing the specific kāraka role.
 - With a single keystroke, one can generate passive constructs which are predominantly found in Sanskrit literature, with which a non-native speaker may not be at ease with.
 - The generator does not dictate any word order. So one may generate a sentence in any word order as one desires. In the future, it should also be possible to provide a generator that will help the user to render the text in a chosen prosodic meter.
3. The generator is useful for testing the parser performance as well. Since both the modules are developed independently, testing helps in mutual improvement of the systems.
4. The major contribution of the development of this module was in identifying some morpho-syntactic relation labels such as those due to upapadas (Kulkarni, 2019).

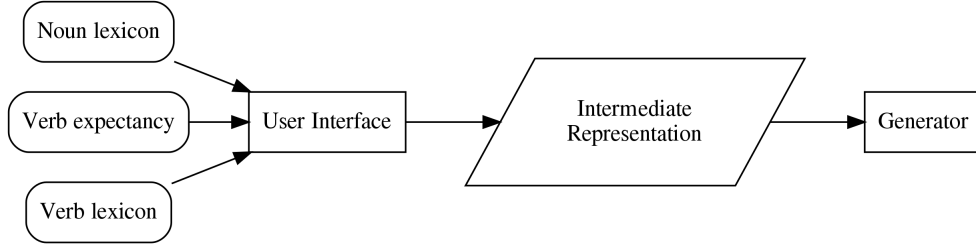


Figure 6: User interface

5. One disadvantage of this generator is the amount of information one has to provide for generation in a particular format.
6. While most of the relation labels are semantic in nature, one may need some initial training for the proper use of some relational tags.
7. One also needs some training in specifying the use of conjuncts and disjuncts since the current implementation is dominated by the syntax of Sanskrit(Panchal and Kulkarni, forthcoming). More research is needed to arrive at a uniform treatment of the conjuncts across languages.

References

- [Bharati et al.1994] Akshar Bharati, Vineet Chaitanya, and Rajeev Sangal. 1994. *Natural Language Perspective - A Paninian Perspective*. Prentice Hall of India.
- [Cardona2007] George Cardona. 2007. On the Structure of Pāṇini's System. *Sanskrit Computational Linguistic*, 1&2:1–31.
- [Dale2000] Ehud Dale, Robert; Reiter. 2000. *Building natural language generation systems*. Cambridge University Press, Cambridge, U.K.
- [Galitsky2013] Boris Galitsky. 2013. A web mining tool for assistance with creative writing. In *Advances in Information Retrieval. Lecture Notes in Computer Science. Lecture Notes in Computer Science. 7814*.
- [Hämäläinen] Mika Hämäläinen. Poem Machine - a Co-creative NLG Web Application for Poem Writing. *Department of Digital Humanities, University of Helsinki*.
- [Iwama and Kano2018] Kango Iwama and Yoshinobu Kano. 2018. Japanese advertising slogan generator using case frame and word vector. In *Proceedings of The 11th International Natural Language Generation Conference, Japan*, pages 197–198, Japan, November. Association for Computational Linguistics.
- [Joshi2009] S. D. Joshi. 2009. Background of the Aṣṭādhyāyī. *Sanskrit Computational Linguistic*, 3:1–5.
- [Kiparsky2009] Paul Kiparsky. 2009. On the Architecture of Pāṇini's Grammar. *Sanskrit Computational Linguistic*, 1&2:32–94.
- [Kulkarni2019] Amba Kulkarni. 2019. Appropriate Dependency Tagset for Sanskrit Analysis and Generation. In *Proceedings of Sanskrit in China International Conference 2019: Sanskrit on Paths*. forthcoming.
- [Panchal and Kulkarniforthcoming] Sanjeev Panchal and Amba Kulkarni. forthcoming. Ca-śabdayukta-vākyaviśeṣaṇam. In Gauri Mahulikar, editor, *Proceedings of NFSI*. Chinmaya Vishvavidyalaya, Veliyanad.
- [Pande1992] Gopal Dutt Pande. 1992. *Aṣṭādhyāyī of Pāṇini*. Chaukhamba Surbharti Prakashan, Varanasi.
- [R and P2017] Perera R and Nand P. 2017. Recent Advances in Natural Language Generation: A Survey and Classification of the Empirical Literature. *Computing and Informatics*, 36 (1):1–32.
- [Ramakrishnamacharyulu2009] K.V. Ramakrishnamacharyulu. 2009. Annotating Sanskrit Texts Based on Śābdabodha Systems. *Sanskrit Computational Linguistics*.
- [Rao1969] Veluri Subba Rao. 1969. *The Philosophy of a Sentence and its Parts*. Munshiram Manoharlal Publishers, New Delhi.
- [Shailaja2014] N. Shailaja. 2014. *Comparison of Paninian Dhātuvṛttis*. Ph.D. thesis, Department of Sanskrit Studies, University of Hyderabad.

A Tagset of Dependency Relations

- **Kāraka-sambandhāḥ**
- kartā
 - prayojaka-kartā
 - prayojya-kartā
- karma
 - mukhya-karma
 - gaṇa-karma
 - vākya-karma
- karaṇam
- sampradānam
- apādānam
- adhikaraṇam
 - kāla-adhikaraṇam
 - deśa-adhikaraṇam
 - viśaya-adhikaraṇam
- **Kāraketara-sambandhāḥ**
 - **Kriyā-kriyā-sambandhāḥ**
 - * pūrva-kālaḥ
 - * vartamāna-samāna-kālaḥ
 - * bhaviṣyat-samāna-kālaḥ
 - * bhāvalakṣaṇa-pūrva-kālaḥ
 - * bhāvalakṣaṇa-vartamāna-samāna-kālaḥ
 - * bhāvalakṣaṇa-anantara-kālaḥ
 - * sahāyaka-kriyā
 - **Kriyā-sambandhāḥ**
 - * sambodhyaḥ
 - * hetuḥ
 - * prayojanam
 - * kartṛ-samānādhikaraṇam
 - * karma-samānādhikaraṇam
 - * kriyāviśeṣaṇam
 - * pratiśedhaḥ
- **Nāma-nāma-sambandhāḥ**
 - * śaṣṭhī-sambandhaḥ
 - * aṅgavikāraḥ
 - * vīpsā
 - * viśeṣaṇam
 - * sambodhana-sūcakam
 - * vibhaktam
 - * avadhiḥ
 - * abhedaḥ
 - * lyapkarmādhikaranam
 - * nirdhāraṇam
 - * atyanta-saṃyogaḥ
 - * apavarga-sambandhaḥ
 - * vakyakarmadyotakaḥ
- **Upapada-sambandhāḥ**
 - sandarbhabinduḥ
 - tulanābinduḥ
 - viśayādhikaraṇam
 - nirdhāraṇam
 - prayojanam
 - udgāravācakaḥ
 - saha-arthaḥ
 - vinā-arthaḥ
 - svāmī
 - srotaḥ
- **Vākyetasambandhāḥ**
 - anuyogī
 - pratiyogī
 - nitya-sambandhaḥ
- **Samuccayādisambandhāḥ**
 - samuccitaḥ
 - samuccaya-dyotakaḥ
 - anyataraḥ
 - anyatara-dyotakaḥ

Note: The bold entries are the headings and do not indicate relation labels

Dependency Parser for Sanskrit Verses

Amba Kulkarni, Sanal Vikram and Sriram K

Department of Sanskrit Studies

University of Hyderabad

apksh.uoh@nic.in, sanal.vikram@gmail.com, sriramk8@gmail.com

Abstract

Sentence parser is an essential component in the mechanical analysis of natural language texts. Building a parser for Sanskrit text is a challenging task because of its free word order and the dominance of verse style in Sanskrit literature in comparison to prose style. In this paper, we describe our efforts to build a parser which parses both prose as well as verse texts. It employs an Edge-Centric Binary Join method using various constraints following traditional rules of verbal cognition. We also propose a Daṇḍa-anvaya-janaka which converts the parsed verse form to its canonical prose order.

1 Introduction

Parsing natural language sentences automatically to reveal the underlying semantics has attracted many researchers to this field in the past two decades. The parse of a sentence is useful for several applications ranging from machine translation, information retrieval to question answering. Parsing sentences with fixed word order is comparatively easier than parsing texts that show some flexibility in the word order. We come across such flexibility in poetry. The syntax and semantics of poems have been an area of serious studies. Delmonte (2018) studies the syntax and semantics of Italian poetry. He observes that the best parsers for Italian based on statistical probabilistic information fail to parse poetic structures while the rule based system performs well. Lee and Kong (2012) have noticed the importance of treebank for poems in order to use the statistical or machine learning models, and have developed a dependency treebank for Classical Chinese poems. The Stanford Dependency relations were extended in order to account for certain poetic constructs in Chinese.

(Krishna et al., 2019) proposed a model, called *kāvya guru*, for the conversion of Sanskrit sentences in verse to prose form, which considers the task of conversion as a linearisation problem. It first uses—Dynamic Meta Embeddings (DME)—for training, where it forms a single meta embedding from multiple pretrained word embeddings of a given token. Then it uses a linearisation model—Self-Attention Based Word Ordering (SAWO)—which generates multiple permutations of words, which are then sent to a seq2seq model that produces the required prose order form. They compared the performance of their system with an LSTM based Linearisation Model, and seq2seq model with Beam Search Optimisation, and their system performs the best with a BLEU score of 55.26.

Majority of Sanskrit literature is in verse form. These verses follow metrical patterns which make them easy to memorise. The metrical pattern also brings in deviation from the default word order found in the prose. This makes it difficult to understand the verse without any special training. Sanskrit being a flexional language, and also rich in derivational morphology, enjoys the flexibility in the word order. There is, as well, a natural tendency to have a kind of rhythm even in the normal speech in Sanskrit, which results in the deviation from normal word order. Gillon (1996) reports several cases of dislocations of arguments from their default order even in prose. This flexibility, however, makes parsing such texts a bit challenging.

In this paper we describe a parser for Sanskrit that can parse both verse and prose. In the next section we describe the basic architecture of our parser that extracts a tree from a graph satisfying some local and global constraints. In the third section we provide the algorithm for constraint solver and illustrate it with an example. Next two sections describe an application of this parser to get the prose order (also termed *daṇḍa-anvaya*) of any verse. We conclude with the discussion on the performance of the parser stating its limitations and the areas where it needs further improvement.

2 Design of a Parser

We find two main approaches towards the design of a dependency-based parser. They are Grammar based and Data driven. The Link parser (Sleator and Temperley, 1993) based on Link grammar formalism and the Minipar (Lin, 1998) based on Chomsky’s Minimalism are among the grammar based dependency parsers. Data-driven dependency parsers are the state-of-art parsers. They use supervised machine learning algorithms to train the machine on annotated corpus. These parsers need manually annotated corpus, called tree banks, for training. Among these parsers, we come across two dominating approaches. They are graph-based dependency parsing and transition-based dependency parsing. The graph-based approach creates a parser model that assigns scores to all possible dependency graphs and then uses maximum spanning tree methods from Graph theory for getting the highest-scoring dependency graph. The transition-based approach scores transitions between parser states based on the parse history and then follows a greedy approach and produces a single parse corresponding to the highest-scoring transition sequence that derives a complete dependency graph.

Most of the natural language parsers call a part of speech (POS) tagger and a chunker before invoking a parser. These two modules reduce the ambiguity due to multiple morphological analyses. A POS tagger selects the best part of speech in the context, and a chunker groups all the auxiliary verbs with the main verbs, the post-positions with the noun, and multi-word expressions as one chunk. The head of such chunks is marked which relates to other words or heads of other chunks in a sentence. The POS taggers and chunkers ease the task of a parser, by reducing the ambiguities at the morphological level. However the disadvantage of calling these modules before a parser is that the errors may get cascaded.

Our parser differs from the state-of-the-art parsers in three ways. First, in the absence of any annotated corpus, we follow the grammar based approach. Secondly, our parser is invoked right after the morphological analyser. The main reason behind this decision was the following. Indian literature on verbal import was found to be useful from parsing point of view since it has discussions on various factors that are instrumental in the process of verbal cognition. Our main goal is to build a parser modeling the theories of *śābdabodha*. When we looked at various Indian literature related to the theories of verbal cognition, there was no discussion on any kind of POS tagger or chunker. Moreover, use of chunker also presupposes that dependencies relate the whole chunk and do not involve a sub-part of it. But in Sanskrit we come across instances of compounds termed as *asamartha-samāsa* (Joshi, 1968; Gillon, 1993) where the dependencies relate to the sub-part of a compound which need not necessarily be a head. Use of a chunker module before calling a parser would fail to parse such constructs. Finally, the state-of-art parsers typically produce a single parse. We decided to produce all possible parses. This is to ensure that we do not miss out the correct parse. The onus of choosing the correct parse, from among the parses produced, is on the reader.

The challenge before us was to handle the free word order in Sanskrit both in prose as well as in verse. The basic algorithm we followed for parsing is given below.

1. Define one node each corresponding to each morphological analysis of every word in a sentence.
2. Establish directed edges between the nodes, if there is either a mutual or unilateral expectancy (*ākāṅkṣā*) between the corresponding words and the word meanings are not mu-

tually incongruous (yogyatā).

3. Define constraints, both local on each node as well as global on the graph as a whole. One of these constraints corresponds to sannidhi (proximity).
4. Extract all possible trees from this graph that satisfy both local and global constraints. Produce all possible solutions to ensure that in case of sentences with multiple interpretations,¹ machine does not miss any interpretation.
5. Produce the most probable solution as the first solution by defining an appropriate cost function. The cost C associated with a solution tree is defined as $C = \sum_e d_e \times r_k$, where e is an edge from a word w_j to a word w_i with label k , $d_e = |j - i|$, r_k is the rank² of the role with label k .

Then the problem of parsing a sentence may be modeled as the task of finding a sub-graph T of G such that T is a Directed Tree (or a Directed Acyclic Graph).

To start with, in order to get familiarity with the kind of problems due to ambiguity, we designed a parser (Kulkarni et al., 2010) that handles a text in formally defined canonical prose order. This parser was implemented as a constraint solver. This parser was found to be very inefficient due to the use of matrix data structure which resulted in sparse matrices for long sentences or sentences with heavily ambiguous words, affecting the efficiency. This algorithm was later improved by using vertex-centric traversal using dynamic programming (Kulkarni, 2013). The major disadvantage of this method is, being node-centric traversal, if the initial words have several incoming arrows, then the number of partial solutions in the beginning are many and as one traverses various paths, the possibilities grow exponentially. It also checks the compatibility of each new edge with all the edges on the path explored so far. This leads to some redundancy, since if a node falls on more than one path, it would be visited more than once, and during each such visit all the incoming edges are checked for compatibility with all other edges on the path traversed so far. In the worst case scenario the incompatibility between the nodes would be noticed only at the final node.

Both these algorithms were designed for sentences that have a default SOV order. Now we present below an algorithm that is designed to handle both prose as well as verse order. This algorithm also overcomes the disadvantages of the earlier algorithm viz. the redundancy in compatibility checking. It has been observed that the arguments having mutual expectancy (utthita ākāṅkṣā), such as the core arguments of a predicate, follow weak non-projectivity while the arguments having unilateral expectancy (utthāpya ākāṅkṣā) are exceptions to this rule (Kulkarni et al., 2015). We use these constraints to extract a tree from the graph.

3 Edge-centric Binary Join

We modify the previous algorithm at three levels.

1. Any edge that is a part of the solution should be compatible with remaining $n - 2$ edges in the solution tree, where n is the number of words in the sentence. This is to ensure that the solution has $n - 1$ edges. Hence, all those edges that are not compatible with at least $n - 2$ other edges are thrown away.
2. We define the compatibility of two sets of edges as a simple operation of set intersection.
3. We build the solutions recursively starting with the individual words bottom-up, each time joining two sets of compatible edges. In $n - 1$ joins we get all possible directed acyclic graphs (DAG), where n is the number of words in a sentence. Join operation is defined as a set union.

This algorithm is edge-centric.

Before giving the detailed algorithm, we define a few terms.

1. Local constraints:

¹as in the case of texts involving pun or multiple meanings (śleṣa).

²All the roles are ranked, on the basis of heuristics, from 1 to 99.

- (a) A morpheme corresponding to a suffix marks only one relation.
That is, a node can have one and only one incoming edge.
- (b) Each kāraka relation is marked by a single morpheme.
There cannot be more than one outgoing edge with the same label from the same node, if the relation corresponds to a kāraka relation,³ i.e. there cannot be two words satisfying the same kāraka role of the same verb.
- (c) A morpheme does not mark a relation to itself.
A word cannot satisfy its own expectancy, i.e. a word cannot be linked to itself.
- (d) There can be only one valid analysis of every word per solution. Since a word has one node corresponding to each morphological analysis it has, there are further restrictions as below.
 - i. If a word has both an incoming edge as well as an outgoing edge, they should be through the same node.
 - ii. If there is more than one outgoing edge for a word, then all of them should be through the same node.
 - iii. A viśeṣaṇa cannot have a viśeṣaṇa.⁴

These conditions ensure that only one morphological analysis is chosen per word.

2. Global Constraints:

- (a) Sannidhi: There are no crossing of edges.
If all the nodes are plotted in a straight line, then the edges connecting them (drawn on the upper side of the line) should not intersect each other. Adjectival relation and the relation due to genitive suffix are exceptions to this rule.
- (b) Certain relations always occur in pairs. For example, a kartṛsamānādhikaraṇa (a predicative adjective, literally having same locus as that of kartṛ) assumes that there is a relation of kartṛ already established.

3. Compatible edge:

An edge e_1 is said to be compatible with another edge e_2 if they satisfy local constraints, and we set $Compatible(e_1, e_2) = 1$.

4. Compatible set of edges:

Let R be a set with edges $\{r_1, r_2, \dots, r_n\}$, and S be a set with edges $\{s_1, s_2, \dots, s_m\}$. S is compatible with R iff $\forall_i \forall_j Compatible(s_i, r_j) = 1$.

5. Joinable sets:

Let R_1 and R_2 be two sets of edges. Let S_1 and S_2 be the sets of edges that are compatible with R_1 and R_2 respectively. R_1 and R_2 are joinable provided $R_1 \subseteq S_2$ and $R_2 \subseteq S_1$. For such joinable sets, the edges compatible with $R_1 \cup R_2$ are defined as $(S_1 \cap S_2) - (R_1 \cup R_2)$.

Now we give the detailed algorithm.

1. Let there be N edges.
2. For each edge, list down all other edges it is locally compatible with.
3. Construct all possible DAGs, by calling ConstructDags 0 N ,
where ConstructDags is defined as
ConstructDags initial final =
 if (final - initial > 0)
 then
 dags = RemoveSmallDags size (JoinDags dag1 dag2)
 where
 size = final - initial - 1
 dag1 = ConstructDags init mid, and

³adhikaraṇa is treated as an exception since one can have more than one adhikaraṇa as in—
Skt: *rāmaḥ adya pañca vādane gr̥ham agacchat*
Eng: Today Rama came home at five o'clock.

⁴guṇānām ca parārthatvāt asambandhaḥ samatvāt syāt MS 3.1.22


```

dag2 = ConstructDags (mid+1) final,
      where mid = (initial + final) / 2
else
  dags = GetInitialDags init,
  which returns as many initial DAGs as there are
  incoming arrows at the node with index init. Each
  such initial DAG contains a single incoming arrow.
RemoveSmallDags N dags
  removes all the DAGs, from dags that have less than N edges.
JoinDags D1 D2
  joins two dags D1 and D2, if they are joinable sets, and for
  the combined dag D, computes the edges compatible with D.
4. Remove all those solutions that do not satisfy the global compatibility condition.
5. For each globally compatible solution, compute the Cost =  $\sum w * |j - i|$ , where  $w$  is the
   weight of the relation from  $j^{th}$  word to  $i^{th}$  word and then prioritise the solutions based on
   this Cost.

```

3.1 An Example

We illustrate the algorithm with the following simple sentence.

San: *gacchati rāmaḥ vanam.* (1)
gloss: Goes Ram forest{acc.}.
Eng: Ram goes to the forest.

In this sentence, each of the two words *rāmaḥ* (Ram) and *vanam* (forest) has two possible analyses, and the word *gacchati* (goes) has three possible analyses as shown below.

0. *rāmaḥ* = *rāma* {masc.} {sg.} {nom.}
1. *rāmaḥ* = *rā* {pr.} {1p} {pl.}
2. *vanam* = *vana* {neu.} {sg.} {nom.}
3. *vanam* = *vana* {neu.} {sg.} {acc.}
4. *gacchati* = *gam* {pr.} {3p.} {sg.}
5. *gacchati* = *gam* {pr. part.} {masc.} {sg.} {loc.}
6. *gacchati* = *gam* {pr. part.} {neu.} {sg.} {loc.}

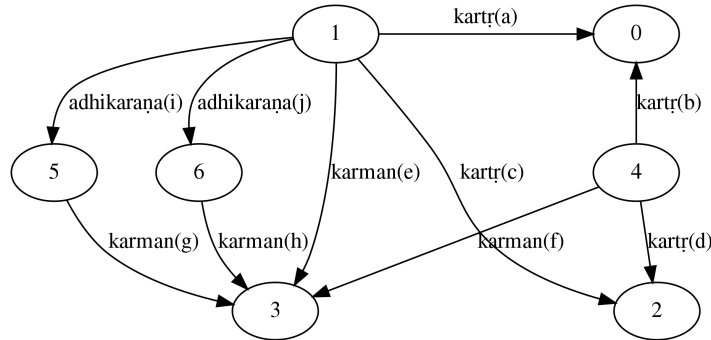


Figure 1: All possible relations for sentence 1

All possible relations are shown in Figure 1 and their compatible relations in Table 1.

Edge ID	From (j)	To (i)	Relation Name (r)	Compatible Edges
a	1	0	kartr	-
b	4	0	kartr	f
c	1	2	kartr	i,j
d	4	2	kartr	-
e	1	3	karman	i,j
f	4	3	karman	b
g	5	3	karman	i
h	6	3	karman	j
i	1	5	adhikaraṇa	c,g
j	1	6	adhikaraṇa	c,h

Table 1: All possible edges and their compatible edges

Instructions	Step	Output at each step
ConstructDags 0 2	12.	{b,f c,i c,j e,i e,j g,i h,j}
ConstructDags 0 1	7.	{b,f b c e f g h }
ConstructDags 0 0	2.	{b}
GetInitDags 0	1.	{b}
ConstructDags 1 1	4.	{c e f g h}
GetInitDags 1	3.	{c e f g h}
JoinDags {b}, {c e f g h}	5.	{b,f b c e f g h}
RemoveSmallDags	6.	{b,f b c e f g h}
ConstructDags 2 2	9.	{i j}
GetInitDags 2	8.	{i j}
JoinDags {b,f b c e f g h}, {i j}	10.	{b,f c,i e,i g,i c,j e,j h,j b c e f g h i j }
RemoveSmallDags	11.	{b,f c,i c,j e,i e,j g,i h,j}
GlobalCompatibilityChk	13.	{b,f}

Table 2: Trace of algorithm on sentence 1

First we filter out the edge a , since it maps the relation between two analyses of the same word, thereby violating local compatibility. Similarly, we filter out edge d , since it is not compatible with any of other edges. We retain all other edges as they are compatible with at least 1 ($= n - 2$) other edge. Next we start building the solutions recursively. We start with the incoming edges of the first word. There is only one incoming edge, marked as b . This forms our first set of edges R_1 . The set of compatible edges with R_1 , denoted by S_1 has only one edge f . For the second word there are five incoming edges, marked as c , e , f , g , and h . Each of these starts a new partial solution. We call them R_2 , R_3 , R_4 , R_5 and R_6 . For each of these edges, the compatible edges are shown in Table 1. We call them S_2 , S_3 , S_4 , S_5 and S_6 respectively. Now we check which of these partial solutions are joinable with R_1 . We notice that only R_4 is joinable with R_1 . Joining these two partial solution sets, results in $\{b,f\}$. The set of edges compatible with this partial solution is given by $(S_1 \cap S_4) - (R_1 \cup R_4) = \phi$. We carry earlier partial solutions viz. R_2 , R_3 , R_4 , R_5 , and R_6 , as well, being potential partial solutions, since each of them has one edge, and we still have one more word to visit. Now we get the edges of the third word, and join them with the current partial solutions. Corresponding to the third word, we have i and j as two incoming edges. Checking compatibility with all the partial solutions in the previous stage, we get seven possible solutions as shown in Figure 2. In Table 2, we show the invocation of the algorithm for this sentence. The result shows the step number followed by the list of

possible relations at that step. In this trace, we have not shown the compatible edges at each stage for each partial dag.

Finally, we check all these solutions for global compatibility. In this example only $\{b, f\}$ satisfies the global compatibility. And thus we get a unique solution. This corresponds to the top left tree in Figure 2. If there are more than one globally compatible solutions, we rank them with the same cost function defined earlier.

In this algorithm, JoinDags is called $n - 1$ times. If there are r_i incoming edges for i^{th} word, then in the worst case, there are $\prod_i r_i$ set union and set intersection operations.

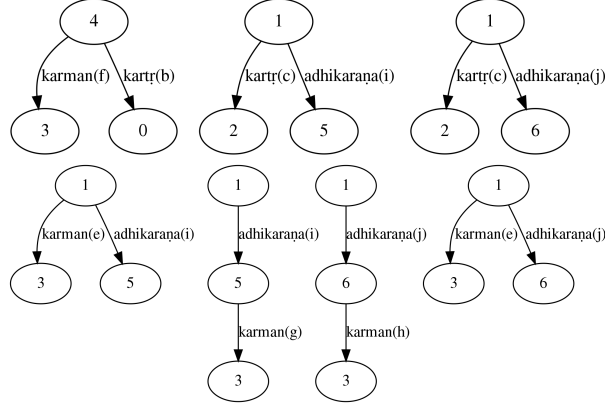


Figure 2: All Possible solutions

3.2 Another Example

Figure 3 shows the parse of the first śloka from Śisūpālavadhānam by the poet Māgha, which occupies a prominent place among the Mahākāvya. It has the three virtues of the best Kāvya, viz. upamā of Kālidāsa, arthagauravam of Bhāravi and padalālityam of Daṇḍi. We also tried to parse the daṇḍa-anvaya of the same śloka, and Figure 4 shows the parse of the anvaya. The śloka and its prose form are given below.

Śloka: *śriyaḥ patih śrīmati śāsitum jagat jagat-nivāsaḥ vasudeva-sadmani /
vasan dadarśa avatarantam ambarāt hiraṇya-garbha-aṅga-bhuvam munim hariḥ* // (2)

Daṇḍa-anvaya: *śriyaḥ patih jagat-nivāsaḥ hariḥ jagat śāsitum śrīmati vasudeva-sadmani vasan
ambarāt avatarantam hiraṇya-garbha-aṅga-bhuvam munim dadarśa* | (3)

Eng: Lakṣmi's consort, Viṣṇu, who is the source of the world, who was residing in the house of Vasudeva to control the world, saw Brahma's son Nārada, descending from the sky.

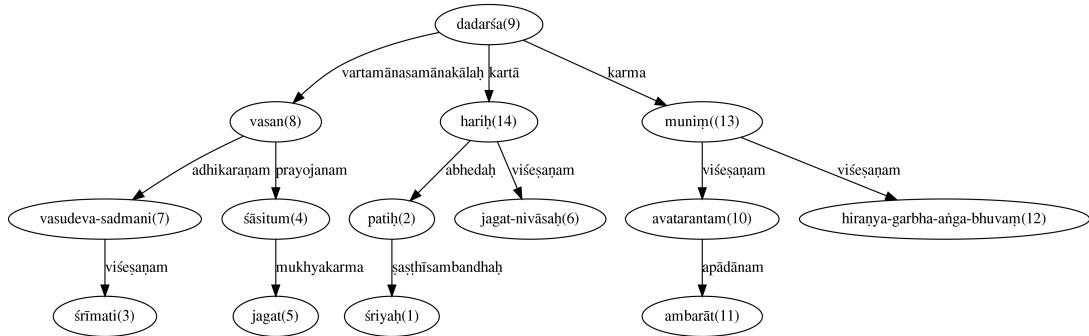


Figure 3: Parse of the śloka (2)

on filling these slots with the nominal forms in the śloka. Once the basic skeleton with all the expectancies is ready, then the commentator connects the *viśeṣaṇas* (adjectives) to their *viśeṣyas* (headwords), providing flesh to the skeleton.

The parse produced by the machine provides us the *khaṇḍānvaya*. All the words that are directly related to the verb work as a backbone, or as a part of the sentence carrying core information. The adjectives attached to the nouns, to arguments of non-finite verbs, etc. typically occupy the second or higher level in the tree structure, and add the flesh to the structure.

- The second approach is the *Daṇḍa-anvaya* (also known as *anvaya-mukhī*). In this method, first the commentator arranges the words in the śloka in a prose form, following a default word order typically encountered in prose.

In the next section, we present an algorithm that produces the *Daṇḍa-anvaya* for a śloka, from the parsed output of a śloka.

5 Daṇḍa-anvaya-janaka

The dependency structure, produced by the parser described above, of the following śloka from *Bhagavadgītā* is shown in Figure 5.

*Dṛṣṭvā tu pāṇḍavānīkam Vyūḍham duryodhanaḥ tadā /
Ācāryam upasaṅgamyā Rājā vacanam abravīt || (BhG 1.2)*

At that time, after seeing the army of the Pāṇḍavas arranged in military phalanx, Duryodhana approached (his) teacher and spoke (these) words.

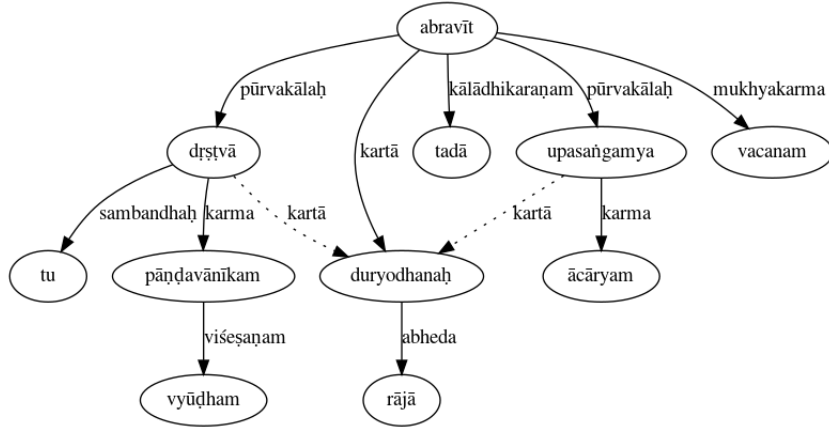


Figure 5: Dependency graph of Bhagavadgita 1.2 śloka

The machine internal representation of this parsed output is in the form of a set of quintuplets containing the relations among words. Each quintuplet (a, b, r, x, y) consists of information about one dependency relation where,

a represents the word ID

b represents the morphological variant of the word

r represents the relation of the word with its parent word

x represents the word ID of the parent word

y represents the morphological variant of the parent word

Word (a, b)	Relation (r)	Parent Word (x, y)
dr̥ṣṭvā (0, 0)	pūrvakālaḥ	abravīt (10, 0)
tu (1, 0)	sambandhaḥ	dr̥ṣṭvā (0, 0)
pāṇḍavānīkam (2, 0)	karma	dr̥ṣṭvā (0, 0)
vyūḍham (3, 0)	viśeṣaṇam	pāṇḍavānīkam (2, 0)
duryodhanaḥ (4, 0)	kartā	abravīt (10, 0)
tadā (5, 0)	kālādhikaraṇam	abravīt (10, 0)
ācāryam (6, 0)	karma	upasaṅgamyā (7, 0)
upasaṅgamyā (7, 0)	pūrvakālaḥ	abravīt (10, 0)
rājā (8, 0)	abhedāḥ	duryodhanaḥ (4, 0)
vacanam (9, 1)	mukhyakarma	abravīt (10, 0)

Table 3: Output of Samsaadhanii parser

Table 3 shows the machine internal representation of the dependency graph shown in Figure 5. The shared roles are marked by dotted lines. For the purpose of re-ordering the words in Daṇḍa-anvaya order, these shared roles are not useful and hence ignored.

Initializing Reordering Task

Anvaya reordering tool is a simple script written in Python. It takes the set of quintuplets as input and creates a corresponding Python tree object. Since multiple morphological variants of a word cannot occur in a single set of dependency solution, variant information is not used presently but is preserved for proposed uses in the future.

Graphical representation of the tree object created with the parsed information of the Bhagavadgītā verse is same as in Figure 5, without the dotted lines.

Deciding the Order

We found the clues for anvaya-order in the *Samāsacakra*. The two relevant kārīkās go like this.

*Ādau kartṛpadam vācyam dvitīyādīpadam tataḥ
Ktvātumunlyap ca madhye tu kuryād ante kriyāpadam*
(Samāsacakram kārīkā 4, (Bhagirath, 1901, p. 12))

Starting with kartṛ, followed by other words, placing the non-finite verbal forms such as *ktvā*, *tumun*, *lyap* in between, place the main verb at the end.

*Viśeṣaṇam puraskṛtya viśeṣyam tadanantaram
Kartṛ-karma-kriyā-yuktam etad anvaya-lakṣaṇam*
(Samāsacakram kārīkā 10, (Bhagirath, 1901, p. 13))

Starting with adjectives, targeting the headword, in the order of kartṛ-karma-kriyā (subject-object-verb), gives an anvaya (the natural order of words in a sentence).

In recent studies, Aralikatti (1991) has shown that the unmarked word order in Sanskrit is SOV. That is, all the arguments of a verb are placed to the left of the verb starting with the kartṛ, then karman followed by other arguments, the attributive adjectives are placed to the left of the noun they qualify, and the predicate is at the end of the sentence. The sub-ordinate clauses, if any, are before the predicate.

Taking clue from these resources, we define a sentence to be in **canonical word order** if it satisfies the following criteria:

All the modifiers are placed to the left of the word they modify.

This is equivalent to the following.

1. The adjectives are to the left of the substantives they qualify.
2. All the arguments of a verb (either in finite form or in non-finite form) are to its left.

3. All the non-finite forms that modify the finite verb form are to its left.

This implies that the main verb⁶ is always the last word of a sentence. This canonical word order provides us the Daṇḍa-anvaya for ślokas. We assigned the priorities to the dependency relation labels following these clues. These priorities were further fine-tuned by studying the commentaries and prose orders of around 400 ślokas from literature including Bhagavadgītā, Nītiśataka, various subhāṣitas and about 50 poetic prose sentences from Kādambarī.

Adjusted by various measures, currently, the relative positions of various arguments are fixed following the rules given below.

1. Sambodhya (vocative) comes at the initial position in the canonical order.
2. Kartṛ comes after vocative.
3. Kāraḱa relations follow in reverse order i.e. adhikaraṇa, apādāna, sampradāna, karaṇa and karman.
4. Viśeṣanas, modifiers with genitive case markers, etc. are placed before their viśeṣya.
5. Kriyāviśeṣana, pratiṣedha etc. are placed right before their corresponding verb.
6. Mukhyakriyā is positioned at the end of the sentence.
7. Avyaya particles such as *tu* and *api* are placed right after their parent word.
8. The non-finite verbal forms are placed before the karman. All the arguments of non-finite verb appear to their left.
9. The kartṛ-samānādhikaraṇa and karma-samānādhikaraṇa are placed after the kartṛ and karman respectively.

Sorting the Tree

The reordering tool traverses through the tree object using level-order-iteration and sort recursively at each node. Primary sorting is carried out based on the relation priorities. The indeclinables such as emphatic particles, and conjuncts are left out as their positions are fixed with respect to their parent node. If there are relations with equal priorities at any level, secondary sorting is done based on the word order (ID) in the original sentence.

The reordered dependency tree of the example śloka is represented in Figure 6.

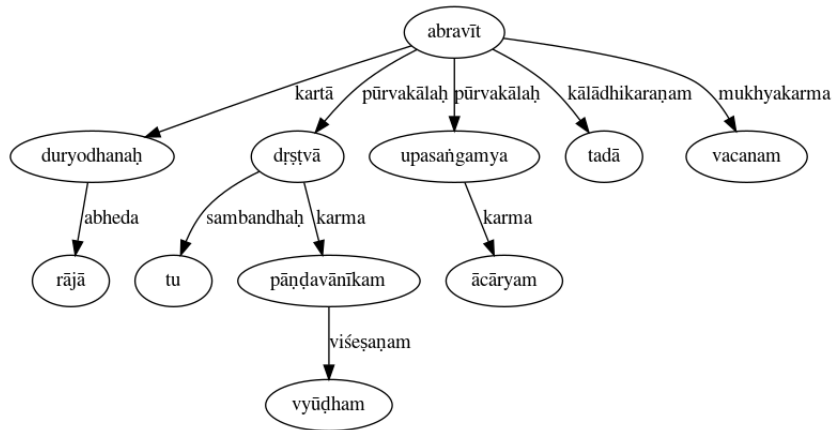


Figure 6: Dependency tree object with sorted relations

Linearizing the Tree

The sorted dependency tree is linearized to get the anvaya order. The tree is traversed using post-order-iteration and each node is added to the linear order pattern.

⁶The main verb can be either in finite form, or in a participial form with either of the suffixes: *kta*, *ktavatu* (Speijer, 1886 Reprint 2009), or any of the kṛtya suffixes viz. *anīyar*, *tavyat*, *tavya*, *yat*, *kyap*, *ṇyat* or *kelīmer*.

The tree mentioned in Figure 6 is linearized in the order: *Rājā Duryodhanah vyūḍham pāṇḍavānikam dr̥ṣṭvā tu ācāryam upasaṅgamyā tadā⁷ vacanam abravīt.*

5.1 Performance

This parser was tested on 195 instances and their canonical prose versions. The sample was taken from the corpus available at Heritage Platform⁸, which essentially corresponds to the citations in the dictionary entry and thus is a random selection from Sanskrit texts belonging to different branches of knowledge and different time period. We provided manually their canonical form. And both the canonical form as well as verse form was run through the parser. Out of 195, the parser could not parse 45 instances both in prose as well as in verse form. One major reason for the failure is out of vocabulary words. The average number of parses for verse order text and prose order text were 151 and 60 respectively. There were around 10 instances, where the number of parses was greater than 1000. This was mainly due to over-analysis with the genitive case markers, in the absence of proper handling of mutual congruity. The median for number of parses is 4, for both verse as well as prose.

Some of the limitations of the current parser are—

1. The parser is based on the Vaiyākaraṇa's theory of śābdabodha. As such, it expects a verb in a sentence. Sanskrit has a tendency of eliding stative verbs meaning 'to be' like *asti*, *bhavati* etc. Parser shows poor performance dealing with such sentences.
2. The relation of kartṛsamānādhikaraṇa is established with a noun, only if it agrees with kartṛ in gender, number, person and case suffix. There are exceptions in literature where samānādhikaraṇas have semantic compatibility though they don't agree in gender, number etc. For example,
 - Chandaḥ pādau tu vedasya (*chandaḥ* and *pādau* do not agree in number).
 - Māyā idam sarvam (Gender of *māyā* does not agree with that of *idam* and *sarvam*).Parser fails to establish relations among such words.
3. Parser performs poorly on some domain specific sentences. Here is an example from mathematical domain: *caturādhikam śatamaṣṭaguṇam dvāṣaṣṭistathā sahasraṇām ayutadvaya-viṣkambhasyāsannaḥ vṛttapariṇāhaḥ.*

6 Conclusion

The main purpose behind the development of an indigenous parser was to evaluate the usefulness of the theories of śābdabodha for the mechanical parsing of Sanskrit sentences. The theories of śābdabodha discuss in minute detail the flow of information, various means of encoding the information, the amount of information encoded, and so on. These theories were further supported by providing various conditions such as ākāṅkṣā, yogyatā and sannidhi, that help in the process of verbal cognition. So we decided to model these conditions computationally.

In this paper we have presented an edge-centric algorithm that handles both prose as well as poetry. In this algorithm, the incompatibility between the edges is noticed at an early stage. And hence the non-solutions are thrown out at an early stage. The user interfaces allow the user to select the best suited segmentation and provide the canonical word order of such segmented text.

We noticed that the performance of the algorithm when the input is in prose form is better than when it is in verse form. The relations contributing to the over-generation are the relation due to genitive case suffix and the adjectival relation. More research towards the nature of dislocation and syntactic constraints on dislocation, and also the semantic compatibility of the words related thus would help in rejecting the non-solutions mechanically.

⁷Here tadā, though a kālādhikaraṇam, acts as a connector between the previous and the current sentence, and thus should be at the beginning of a sentence. However, since the current implementation does not handle inter-sentential relations, the word 'tadā' is not placed at the beginning.

⁸<http://sanskrit.inria.fr>

References

- [Aralikatti1991] R. N. Aralikatti. 1991. A note on word order in modern spoken Sanskrit and some positive constraints. In Hans Henrich Hock, editor, *Studies in Sanskrit Syntax*, pages 13–18. Motilal Banarsidass Publishers.
- [Attardi2006] Giuseppe Attardi. 2006. Experiments with a multilanguage non-projective dependency parser. In *Proceedings of CoNLL*, pages 166–170.
- [Bhagirath1901] Hariprassad Bhagirath. 1901. *Samāsacakra*. Jagadishwar Press, Mumbai.
- [Delmonte2018] Rodolfo Delmonte. 2018. Syntax and semantics of italian poetry in the first half of the 20th century. arXiv:1802.03712v2 [cs.CL].
- [Gillon1993] Brendan S. Gillon. 1993. Bhartṛhari’s solution to the problem of asamartha compounds. *Études Asiatiques/Asiatiche Studien*, 47(1):117–133.
- [Gillon1996] Brendan S. Gillon. 1996. Word order in Classical Sanskrit. *Indian Linguistics*, 57(1):1–35.
- [Joshi1968] S. D. Joshi. 1968. *Patañjali’s Vyākaraṇa-mahābhāṣya. Samarthāhnikā. Edited with translation and explanatory Notes*. Center of Advanced Study in Sanskrit. Class C, No. 3. University of Poona, Poona.
- [Krishna et al.2019] Amrith Krishna, Vishnu Sharma, Bishal Santra, Aishik Chakraborty, Pavankumar Satuluri, and Pawan Goyal. 2019. Poetry to prose conversion in sanskrit as a linearisation task: A case for low-resource languages. forthcoming.
- [Kulkarni et al.2010] Amba Kulkarni, Sheetal Pokar, and Devanand Shukl. 2010. Designing a constraint based parser for Sanskrit. In G. N. Jha, editor, *Proceedings of the Fourth International Sanskrit Computational Linguistics Symposium*, pages 70–90. Springer-Verlag LNAI 6465.
- [Kulkarni et al.2015] Amba Kulkarni, Preeti Shukla, Pavankumar Satuluri, and Devanand Shukl. 2015. How free is the ‘free’ word order in Sanskrit. In Peter Scharf, editor, *Sanskrit Syntax*, pages 269–304. Sanskrit Library.
- [Kulkarni2013] Amba Kulkarni. 2013. A deterministic dependency parser with dynamic programming for Sanskrit. In *Proceedings of the Second International Conference on Dependency Linguistics (DepLing 2013)*, pages 157–166, Prague, Czech Republic, August. Charles University in Prague Matfyzpress.
- [KulkarniAugust 2019] Amba Kulkarni. August 2019. *Sanskrit Parsing based on the theories of śābd-abodha*. D. K. Printworld, Delhi.
- [Lee and Kong2012] John Lee and Yin Hei Kong. 2012. A dependency treebank of classical Chinese poems. In *Conference on North American Chapter of the Association of Computational Linguistics: Human Language Technologies*, pages 191–199.
- [Lin1998] Dekang Lin. 1998. Dependency-based evaluation of minipar. In *Workshop on the Evaluation of Parsing Systems*, Granada, Spain.
- [Mcdonald and Nivre2007] Ryan Mcdonald and Joakim Nivre. 2007. Characterizing the errors of data-driven dependency parsing models. In *Proceedings of EMNLP-CoNLL*, pages 122–131.
- [McDonald et al.2005] Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of HLT/EMNLP*, pages 523–530.
- [Nakagawa2007] Tetsuji Nakagawa. 2007. Multilingual dependency parsing using global features. In *Proceedings of the Joint Conference on EMNLP-CoNLL*.
- [Nivre and Scholz2004] J. Nivre and M. Scholz. 2004. Deterministic dependency parsing of english text. In *Proceedings of COLING 2004*, pages 64–70, Geneva, Switzerland.
- [Nivre2006] J. Nivre. 2006. Constraints on non-projective dependency parsing. In *Eleventh Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 73–80, Trento, Italy. ACL.

- [Panchal and KulkarniJuly 2018] Sanjeev Panchal and Amba Kulkarni. July 2018. Yogyataa as an absence of non-congruity. In Gérard Huet and Amba Kulkarni, editors, *Computational Sanskrit and Digital Humanities, selected papers presented at the 17th world Sanskrit Conference, Vancouver*. D. K. Publishers and Distributors Pvt. Ltd.
- [Scharf et al.2015] Peter Scharf, Anuja Ajotikar, Sampada Savardekar, and Pawan Goyal. 2015. Distinctive features of poetic syntax preliminary results. In *Sanskrit Syntax*, pages 305—324. Sanskrit Library, USA.
- [Sleator and Temperley1993] Daniel D. Sleator and Davy Temperley. 1993. Parsing english with a link grammar. In *Third International Workshop on Parsing Technologies*.
- [Speijer1886 Reprint 2009] J. S. Speijer. 1886; Reprint 2009. *Sanskrit Syntax*. Motilal Banarsidass, New Delhi.
- [Tubb and Boose2007] Gary A. Tubb and Emery R. Boose. 2007. *Scholastic Sanskrit: A Handbook for students*. The American Institute of Buddhist Studies at Columbia University in the City of New York, New York.
- [Yamada and Matsumoto2003] H. Yamada and Y. Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of IWPT*, pages 195–206, Nancy, France.

Revisiting the Role of Feature Engineering for Compound Type Identification in Sanskrit

Jivnesh Sandhan¹, Amrith Krishna², Pawan Goyal² and Laxmidhar Behera¹

¹ Department of Electrical Engineering

Indian Institute of Technology, Kanpur, UP, India

²Department of Computer Science

Indian Institute of Technology, Kharagpur, WB, India

jivnesh@iitk.ac.in

Abstract

We propose an automated approach for semantic class identification of compounds in Sanskrit. It is essential to extract semantic information hidden in compounds for improving overall downstream Natural Language Processing (NLP) applications such as information extraction, question answering, machine translation, and many more. In this work, we systematically investigate the following research question: Can recent advances in neural network outperform traditional hand engineered feature based methods on the semantic level multi-class compound classification task for Sanskrit? Contrary to the previous methods, our method does not require feature engineering. For well-organized analysis, we categorize neural systems based on Multi-Layer Perceptron (MLP), Convolution Neural Network (CNN) and Long Short Term Memory (LSTM) architecture and feed input to the system from one of the possible levels, namely, word level, sub-word level, and character level. Our best system with LSTM architecture and FastText embedding with end-to-end training has shown promising results in terms of F-score (0.73) compared to the state of the art method based on feature engineering (0.74) and outperformed in terms of accuracy (77.68%).

1 Introduction

The landscape of Natural Language Processing has significantly shifted towards the realm of Deep Learning and Artificial Neural Networks. With the benefit of hindsight, the title for the seminal work on a neural pipeline for NLP from Collobert et al. (2011), “Natural Language Processing (Almost) from Scratch”, seems prophetic. Neural networks have demonstrated promising results in a wide variety of problems like sentiment analysis (Tai et al., 2015), information extraction (Nguyen et al., 2009), text classification (Kim, 2014), machine translation (Bastings et al., 2017) among others. Many of such models in fact have become part and parcel of a standard NLP pipeline for data processing, especially for the resource-rich languages such as English (Tenney et al., 2019).

There have been academic debates over the philosophical implications of the use of such statistical black box approaches in Computational Linguistics, especially towards the trade-off between performance and interpretability as also summarised in Krishna et al. (2018b). However, in this work, we focus more on the pragmatic side of using such approaches for low resource languages like Sanskrit. Deep Learning models demand a humongous amount of data to train a model effectively. Additionally, it is challenging and often tricky to incorporate available linguistic knowledge into these neural architectures (Strubell et al., 2018). Summarily, we can say that a standard off the shelf neural model relies mostly on its capacity to learn distributional information from the large datasets provided as input during training. In this pretext, we revisit the problem of compound type identification in Sanskrit (Krishna et al., 2016) and experiment with various neural architectures for solving the task.

The process of compounding and the nature of compositionality of the compounds are well studied in the field of NLP. Given that compounding is a productive process of word-formation in languages, this is of much interest in the area of word-level semantics in NLP. There are various aspects involved in the compound analysis. These include productivity and recursiveness of the words involved in the process, presence of implicit relations between the components, and finally, the analysis of a compound relies on its pragmatic or contextual features (Kim and Baldwin, 2005). Recently, there has been a concerted effort in studying the nature of compositionality in compounds by leveraging on distributional word-representations or word embeddings and then learning function approximators to predict the nature of compositionality of such words (Mitchell and Lapata, 2010; Cordeiro et al., 2016; Salehi et al., 2015; Jana et al., 2019). In Sanskrit, Krishna et al. (2016) have proposed a framework for semantic type classification of compounds in Sanskrit. They proposed a multi-class classifier using Random Forests (Geurts et al., 2006; Pedregosa et al., 2011), where they classified a given compound into one of the four coarse level compound classes, namely, *Avyayībhāva*, *Tatpuruṣa*, *Bahuvrīhi* and *Dvandva*. They have used an elaborate feature set, which summarily consists of rules from the grammar treatise *Aṣṭādhyāyī* pertaining to compounding, semantic relations between the compound components from a lexical database *Amarakoṣa* and distributional subword patterns from the data using Adaptor Grammar (Johnson et al., 2007). Inspired from the recent advances in using neural models for compound analysis in NLP, we revisit the task of compound class identification and validate the efficacy of such models under the low-resource setting like that of Sanskrit.

In this work, we experiment with multiple deep learning models for compound type classification. Our extensive experiments include standard neural models comprising of Multi-Layer Perceptrons (MLP), Convolution Neural Networks (CNN) (Zhang et al., 2015) and Recurrent models such as Long Short Term Memory (LSTM) configurations. Unlike the feature-rich representation of Krishna et al. (2016), we rely on various word embedding approaches, which include character level, sub-word level, and word-level embedding approaches. Using end-to-end training, the pretrained embeddings are fine tuned for making them task specific embeddings. So all the architectures are integrated with end-to-end training (Kim, 2014). The best system of ours, an end-to-end LSTM architecture initialised with fasttext embeddings has shown promising results in terms of F-score (0.73) compared to the state of the art classifier from Krishna et al. (2016) (0.74) and outperformed it in terms of accuracy (77.68%). Summarily, we find that the models we experimented with, report competitive results with the current state of the art model for compound type identification. We achieve the same without making use of any feature engineering or domain expertise. We release the codebase for all our models experimented with at <https://github.com/Jivnesh/ISCLS-19>.

2 Compound Classification Task in Sanskrit

In this work, we address the challenge of semantic type identification of compounds in Sanskrit. This is generally treated as a word-level semantic task in NLP (Rink and Harabagiu, 2010; Hashimoto et al., 2014; Santos et al., 2015). We treat the task as a supervised multiclass classification problem. Here, similar to Krishna et al. (2016), we expect the users to provide a compound in its component-wise segmented form as input to the model. But our model relies on distributed representations or embeddings of the input as features, instead of the linguistically involved feature set proposed in Krishna et al. (2016).

Approaches for compound analysis have been of great interest in NLP for multiple languages including English, Italian, Dutch and German (Séaghdha and Copestake, 2013; Tratz and Hovy, 2010; Kim and Baldwin, 2005; Girju et al., 2005; Verhoeven et al., 2014a). These methods primarily rely on lexical networks, distributional information (Séaghdha and Copestake, 2013) or a combination of both lexical and distributional information (Nastase et al., 2006). In Sanskrit, Krishna et al. (2016) proposed a similar statistical approach which combined lexical and distributional information by using information from the lexical network *Amarakoṣa* (Nair and

Kulkarni, 2010) and variable length n-grams learned from data using Adaptor grammar (Johnson et al., 2007). Here, the authors also adopted rules from *Aṣṭādhyāyī* as potentially discriminative features for compound type identification (Kulkarni and Kumar, 2013). While this model has shown to be effective for the task, it nevertheless is a linguistically involved model. Recently, Dima and Hinrichs (2015), Cordeiro et al. (2016) and Ponkiya et al. (2016) have shown that use of word embedding as the sole features can produce models with competitive results as compared to other feature-rich models. Inspired from these observations, we attempt to build similar models which use only embeddings as features for the compound type identification task.

Compounds in Sanskrit can be categorized into 30 possible classes based on how granular categorizations one would like to have (Lowe, 2015). There are slightly altered set of categorizations considered by Gillon (2009), Olsen (2000), Bisetto and Scalise (2005) and Tubb and Boose (2007). Semantically *Aṣṭādhyāyī* categorizes the Sanskrit compounds into four major semantic classes, namely, *Avyayībhāva*, *Tatpuruṣa*, *Bahuvrīhi* and *Dvandva* (Kumar et al., 2010). Similar to prior computational approaches in Sanskrit compounding (Krishna et al., 2016; Kumar et al., 2010), we follow this four class coarse level categorization of the semantic classes in compounds. Compounding in Sanskrit is extremely productive, or rather recursive, resulting in compound words with multiple components (Lowe, 2015). Further, it is observed that compounding of a pair of components may result in compounds of different semantic classes. *Avyayībhāva* and *Tatpuruṣa* may likely be confusing due to particular sub-category of *Tatpuruṣa* if the first component is an *avyaya*. For example, *upa jīvataḥ* has the first component as *avyaya* which is strong characteristic of *Avyayībhāva*. However, this compound belongs to *Tatpuruṣa* class. Likewise, a negation *avyaya* in the first component can create confusion between *Tatpuruṣa* and *Bahuvrīhi* classes. The instances mentioned above reveal the difficulties associated with distinguishing the semantic classes of a compound.

3 System Description

While the compounds in Sanskrit can consist of multiple components, we restrict our problem to that of compounds with two components only. Thus, given the two components of the compound, we treat this as a classification problem. For the task, we use neural models, which can be categorized based on the architectural point of view, namely, Multi-Layer Perceptron (MLP), Convolutional Neural Network (CNN) and Long Short Term Memory (LSTM) based classifier, among others.

These networks typically require a feature representation of the input (in our case, the two components of the compound word), and learn to classify into one of the possible compound categories. We again utilize multiple possibilities of input feature representation. For instance, consider *svamānasam*, which is a *Tatpuruṣa* compound. We can break this compound in three possible ways: 1) word level: *sva mānasam* 2) subword level: *sva mā nas am* (subword level segmentation is based on segmentation learned by Byte Pair Encoding (BPE) (Sennrich et al., 2016) from corpus data). 3) Character level: *s v a m ā n a s a m*.

We learned word embeddings of these components of the compound from our Sanskrit corpus (Section 4.1). Word embeddings map a word x from a vocabulary V to a real-valued vector \vec{x} of dimensionality D in a feature space (Schnabel et al., 2015). The idea based on distributional hypothesis (Harris, 1954), and the learning objective attempts to put similar words closer in the vector space. We used FastText for learning word-level embedding, BPE along with Word2Vec (w2v) (Mikolov et al., 2013) and Glove (Pennington et al., 2014) for learning subword level embedding, and character level embedding learned using CharCNN (Zhang et al., 2015). Note that we learned embeddings for the individual components, and finally concatenated vectors corresponding to each component and fed as input to the classifier.

We also integrated our system with task-specific end-to-end training for text classification (Kim, 2014). This approach facilitates pre-trained initialized vector to be updated during the task-specific training process. Performance of the classifier, with and without end-to-end

training, is reported in Appendix I. In all the architectures, *relu* activation function for dense layer, softmax cross entropy loss function and *adam* optimizer are used.

3.1 MLP based classifier

Multi-layer Perceptron in supervised learning problem consists of an input layer to receive input, output layer to make a decision and multiple hidden layers in between them. Training involves learning the parameters of the model using backpropagation. As discussed earlier, We experiment with feeding input in two levels, namely, word level (FastText and FastText*) and subword level (W2V and Glove along with BPE). Architectures used for them are reported in Table 1. Next to the embedding layer, a drop-out layer with drop-out rate 0.2 is used to avoid over-fitting (Srivastava et al., 2014).

Embedding	Layer	units
w2v [20 x 100]	1	1000
	2	500
	3	100
	4	4
glove [20 x 225]	1	1000
	2	4
FastText [2 x 350]	1	500
	2	4
FastText* [1 x 1400]	1	500
	2	4

Table 1: MLP architecture used for different embeddings. [a x b] indicates that there are total ‘a’ segments of compound and dimension of each segment is ‘b’. For instance, for w2v, there are 20 segments (max) to account for the BPE vocabulary of the compound, and each word in the BPE vocabulary is represented using 100 dimensions.

FastText*: In this case, as shown in Figure 1, FastText vectors of two components of the compound are concatenated along with element-wise absolute difference and element-wise product between the embedding vector of these two vectors (it is denoted by FastText*). Moreover, the resultant vector is passed to MLP based classifier (Table 1) with no end-to-end training.

The architecture we have used to combine information from the two components is similar to the one used for the Natural Language Inference (NLI) problem in Conneau et al. (2017). The key idea behind their approach was to obtain a unified representation of two sentences, each represented as a vector, similar to Figure 1.

3.2 CNN based classifier

CNN has shown outstanding performance in the field of computer vision. The purpose behind adopting CNNs in NLP is to derive position-invariant features (such as phrases, n-grams) using the convolution operation. Max pooling over these features helps to find the essential n-grams and then fully connected hidden layers are employed, similar to MLP, for final predictions. Recently, Kim (2014) has shown the application of CNN for textual data. In our CNN architecture, end-to-end training is integrated into the embedding layer. Next to the embedding layer, drop-out layer with a drop-out rate of 0.2 is used. For different input levels, architecture details are shown in Table 2. Now We will explain CNN used for the character level input.

CharCNN: Zhang et al. (2015) used character level information of text as input for a convolutional neural network. The advantage of the model is that by using character level embedding with convolution layers, word-level embedding can be obtained. This model requires fixed-size input of encoded characters where embeddings of each character are initialized with Gaussian

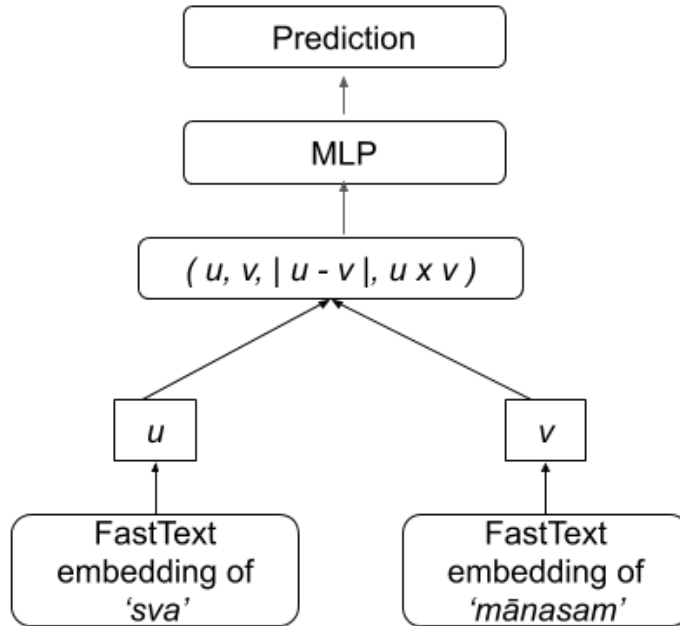


Figure 1: FastText feature augmented with element-wise difference and multiplication of compound’s component (Conneau et al., 2017)

distribution with mean 0 and variance 0.05. CharCNN architecture employed for our experiment is mentioned in Table 2.

Embedding	Layer	Filter	Kernel	Pull
CharCNN [25 x 1014]	1	256	7	3
	2	256	7	3
	3	256	3	N/A
	4	256	3	N/A
	5	256	3	N/A
	6	256	3	3
	7	500	N/A	N/A
	8	4	N/A	N/A
w2v [20 x 700]	1	300	25	4
	2	100	N/A	N/A
	3	4	N/A	N/A
glove [20 x 900]	1	350	25	4
	2	400	N/A	N/A
	3	4	N/A	N/A
FastText [2 x 350]	1	150	25	2
	2	4	N/A	N/A

Table 2: CNN architecture used for different embeddings. For embedding layer, same convention is used. For charCNN, 25 segments correspond to the max number of characters in the compound, and 1014 dimensional embedding is used for each of these.

3.3 LSTM based classifier

The conventional feed-forward neural network treats all input-output pairs independently, which limits the ability to learn patterns in sequential data. RNNs are designed to capture this time dependency where network memorizes the previous input-output interactions in order to predict

the current output. Due to the problem of Vanishing Gradient (Pascanu et al., 2013; Bengio et al., 1994), RNNs can capture only short-term dependencies. To overcome this limitation, LSTM (Hochreiter and Schmidhuber, 1997) is used which employs a gating mechanism to carry forward the long-term dependencies. LSTM has achieved great success in working with sequences of words. In our LSTM architecture, next to embedding layer, drop-out layer with rate 0.2 is used. Embedding layer is integrated with end-to-end training. Architectural details for different input levels are given in Table 3.

Embedding	Layer	Type	units
w2v [20 x 450]	1	LSTM	450
	2	Fully Connected	400
	3	Fully Connected	4
glove [20 x 900]	1	LSTM	450
	2	Fully Connected	400
	3	Fully Connected	4
FastText [2 x 350]	1	LSTM	100
	2	Fully Connected	4

Table 3: LSTM architecture used for different embeddings.

4 Experiments

4.1 Dataset

Our text corpus contains data from the Digital Corpus of Sanskrit (DCS)¹, as well as scraped data from Wikipedia and Vedabase corpus. The number of words in each corpus are 3.8 M, 0.7 M, and 0.2 M, respectively. DCS and Vedabase are segmented, but the Wikipedia data is unsegmented. We have used this corpus to learn word embedding features. Most of the data in our corpus is in the form of poetry.

Figure 2 presents a few statistics regarding the corpus utilized.

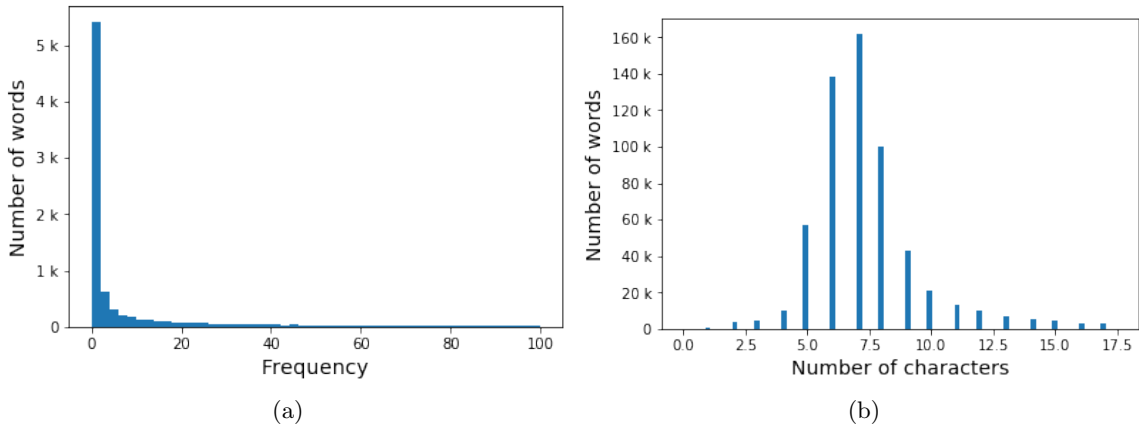


Figure 2: (a) Histogram plot of frequency of the compounds from the classification dataset in the corpus. 50% of compounds have zero occurrence in the corpus. (b) Distribution of number of characters per word in the corpus.

The labelled dataset for the compound classification task with a segmented pair of components is obtained from the department of Sanskrit studies, UoHyd². These compounds are part of ancient texts, namely, *Bhagavadgītā*, *Carakasamhita*, etc. We have used the same experimental

¹<http://www.sanskrit-linguistics.org/dcs/>

²<http://sanskrit.uohyd.ac.in/scl/>

setting as Krishna et al. (2016) for the classification task. The dataset for the compound classification task has more than 32,000 sandhi splitted compounds with labels. There are four broad classes, namely, *Avyayībhāva*, *Tatpuruṣa*, *Bahuvrīhi* and *Dvandva*. More than 75% data points were from *Tatpuruṣa* class, Krishna et al. (2016) down-sampled it to 4,000, which takes it close to the count of the second most highly populated class *Bahuvrīhi*. *Avyayībhāva* class is highly skewed, 5% of the *Bahuvrīhi* class. After down-sampling, number of compounds are 239 in *Avyayībhāva*, 4,271 in *Bahuvrīhi*, 1,176 in *Dvandva*, and 4,266 in *Tatpuruṣa*. Out of 9,952 data-points, 7,957 were kept for training and remaining for testing. We have created development (dev) dataset for hyperparameter tuning, from 20 % stratified sampling of the training data. We have not used test dataset in any part of training or hyperparameter tuning.

4.2 Hyperparameter tuning for input representation

Figure 3(e) and 3(f) show the effect of embedding size on the dev set performance. In FastText, accuracy on dev-set saturated at 350, which we used as the default embedding size. Since most of the data is in the form of poetry, the window size is kept larger. As we increase the epoch size, there was a gradual increase in performance (Figure 3(e)). Parameters min-n and max-n were chosen by plotting the distribution of the number of characters in word (Figure 2(b)).

Figure 2(a) shows that more than 50% data sample from the classification task has zero occurrences in the corpus. So this Out of Vocabulary (OOV) issue is handled by applying BPE with vocabulary size 100. Results did not improve by increased vocabulary size of BPE. BPE vocabulary size is chosen as 100, for both glove and w2v features. Embedding for w2v and Glove is calculated for segmented sub-words. Figure 3(b) and 3(c) indicates that by increasing embedding size, there is a gradual increase in F-score on dev dataset for both BPE+W2v and BPE+Glove. So we chose 450 as the embedding size for w2v. For Glove, feature size, epoch size and window size are 450, 70 and 20, respectively.

In CharCNN, the vocabulary size of characters is 60. Apart from the Sanskrit alphabets, there are other eight symbols present in the dataset, which include numbers. The maximum length of characters in the input is 25. Features corresponding to each character is of size 1014, which is initialized from Gaussian distribution with mean 0 and variance 0.05. Filter size, kernel size, and pull size for each layer are shown in Table 2. Last two layers are fully connected layers with a *relu* activation function. All the hyper-parameters are reported in Appendix II.

4.3 Results

Classifier’s performance is evaluated based on micro accuracy and macro precision, recall and F-score. F-score is the combined metric of precision and recall, so accuracy and the F-score will be our main evaluation metric.

Embedding	Classifier	A	P	R	F
baseline	ERF	77.39	0.78	0.72	0.74
	RF(N-gram)	75.88	0.83	0.64	0.70
Random	CNN+	66.15	0.63	0.57	0.59
charcnn	CNN+	74.65	0.73	0.65	0.68
bpe+w2v	CNN+	71.90	0.74	0.64	0.67
bpe+glove	CNN+	74.13	0.76	0.64	0.68
FastText*	MLP	74.51	0.72	0.66	0.68
FastText	LSTM+	77.68	0.76	0.71	0.73

Table 4: Evaluation measures are accuracy (A), macro precision (P), macro recall (R) and macro F-score (F). Results reported on the test data are averaged over 5 runs. ‘+’ sign indicates end-to-end training integrated with classifier.

We have used two baseline models to compare against, first one is Krishna et al.’s (2016)

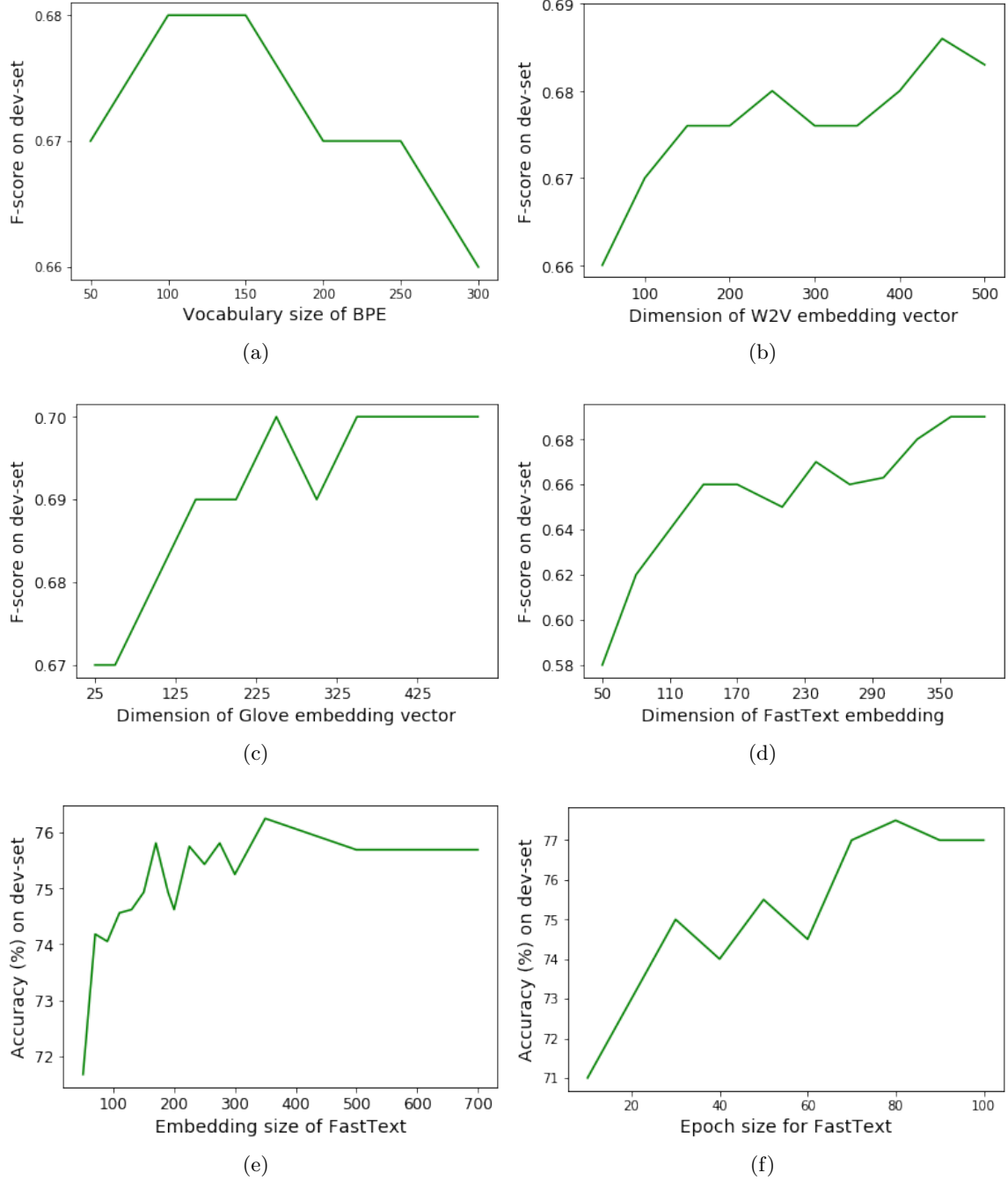


Figure 3: Investigating the sensitivity of the results (F1-score and Accuracy) with respect to the dimensionality of various embeddings on the development set: (a) As vocabulary size of BPE increases, macro F1-score decreases. So we have used the BPE vocabulary size as 100. (b) As embedding size of w2v increases, there is a gradual increase in F-score. So we have chosen 450 as the embedding size. (c) As embedding size of Glove increases, there is a gradual increase in F-score. So we have chosen 450 as the embedding size. (d) As the FastText dimension increases (with component-wise subtraction and product augmentation), there is a gradual increase in F-score. (e) Effect on accuracy as embedding size of FastText increases. For our experimentation, we have chosen embedding size of 350. (f) Effect on accuracy as epoch size varies for FastText.

feature engineered model with ERF classifier (F-score 0.74). Another baseline is N-gram based features with Random Forest (RF) classifier (F-score 0.70). In this model, only N-gram based

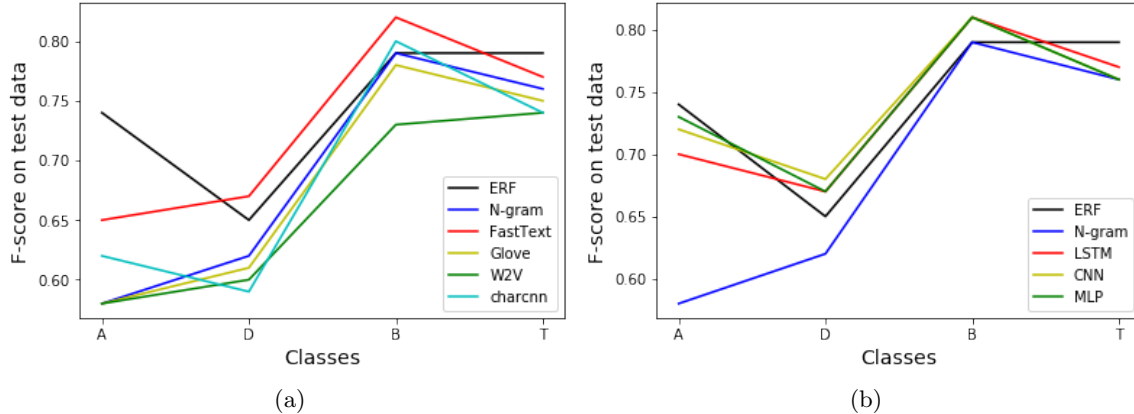


Figure 4: (a) Class-wise F-score for different embeddings with the same architecture (CNN) (b) Class-wise F-score for different architectures with the same embedding (FastText). Note that class size increases as we move from left to right along x-axis. ERF and N-gram are baselines reported in Table 4.

feature engineering is involved, but it was able to give comparable performance.

There are three possible ways to feed input to the system, namely, word level, subword level, and character level. Based on these categorizations, step by step, we evaluated our MLP, CNN, and LSTM based classifiers. First, for word-level inputs, we randomly initialized all the embedding vectors and checked the performance of the classifier. We were able to reach up to 0.59 (macro) F-score with CNN+ classifier (Table 4). Next, for subword level input, we used W2V and Glove embedding on BPE segmented (the segmentation is not morphemic) sub-words of the compound. These embeddings helped to get significant improvement compared to word level randomly initialized embedding, achieving F-score of 0.67 and 0.68, respectively. As shown in Figure 2(a), W2V and Glove could not give very good embeddings due to the rare occurrence of compound words in the corpus. Then we experimented with another embedding, FastText, which has shown excellent performance compared to all other systems. We were able to reach 0.73 (macro) F-score. We almost achieved state of the art result without feature engineering. Then we used the FastText* embedding combination technique to check whether we can improve further, but it declined the actual result to 0.68. Finally, character level input with CharCNN architecture with randomly initialized embedding reached 0.68. Our system outperformed in terms of accuracy (77.68) to state of the art baseline (77.39). We also integrated end-to-end training to learn task-specific embedding in all systems mentioned above. Detailed results for all the systems are presented in Appendix I.

4.4 Error Analysis

We have done a detailed analysis of particular instances of compound types which get misclassified. From confusion matrix heat map in Figure 5, we can see that most of the mis-classification has gone to *Tatpuruṣa* class for our best performing system. There are no mis-classification between *Dvandva* and *Avyayībhāva*. Specific sub-type of *Tatpuruṣa* has similar properties as that of *Avyayībhāva*, where first component of compound is *avyaya*, which creates conflict between these two classes. In our observation, 11 data-points from *Tatpuruṣa* got mis-classified into *Avyayībhāva* where all of them have the first component as *avyaya*. Also from Figure 5(a), we can see that most of the compounds from *Avyayībhāva* were misclassified into *Tatpuruṣa*. Our best model is able to perform better compared to the baseline model for *Bahuvrīhi* and *Dvandva* which are the second and the third most highly populated classes (Figure 4). Figure 5(b) indicates that our best system mostly got confused between *Tatpuruṣa* and *Bahuvrīhi*, because there is a special sub-type in both of these semantic classes which exhibits similar properties.

There are more than 600 unique components of compound common in training set of *Bahuvrīhi* and *Tatpuruṣa*. Out of these, 205 components have more number of occurrences in *Bahuvrīhi* than that of *Tatpuruṣa* and 201 components have more occurrence in *Tatpuruṣa* than that of *Bahuvrīhi*. So common component compounds present in a conflicting class which has less occurrences will be misclassified. Since we have not provided any other information, classifier is getting confused due to common component occurrences in both the classes. Similar cases have been found for *Dvandva* and *Tatpuruṣa*. For example, *bāla* occurred 7 times in *Dvandva* and 12 times in *Tatpuruṣa*, so majority of compounds of *Dvandva* having *bāla* as component will be misclassified into *Tatpuruṣa*. There are 11 such unique components in training set which have number of occurrences more than 4 in either class. We need to provide contextual information in order to overcome this problem. In summary, error cases observed in our best system are similar to that of baseline system. In this classification setup, apart from individual components of compounds, we have not provided contextual information or canonical paraphrasing. With this restriction, the classification problem is not entirely solvable; however, we explored up to what degree the ambiguities can be resolved.

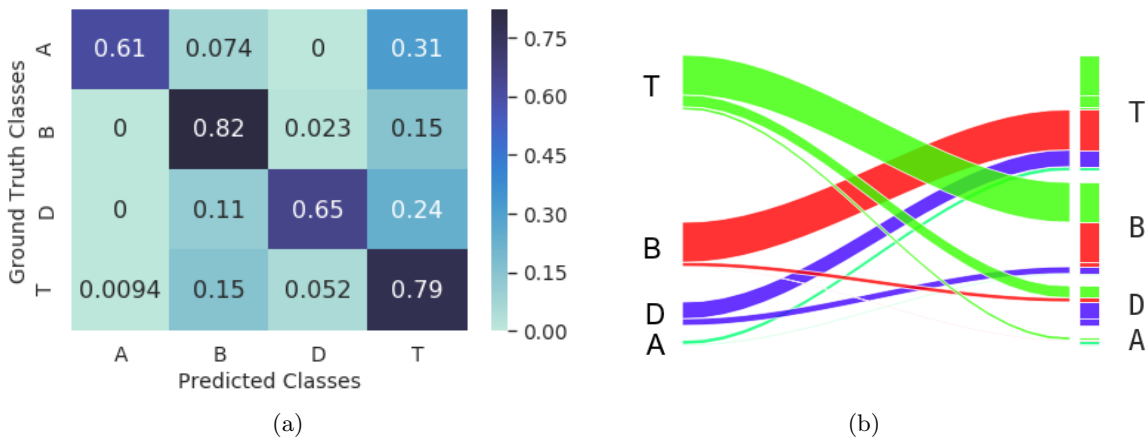


Figure 5: (a) Confusion matrix heat-map for our best performing system (A, B, D and T refer to *Avyayībhāva*, *Bahuvrīhi*, *Dvandva*, and *Tatpuruṣa*, respectively) (b) Alluvial graph for showing mis-classification to demonstrate conflicts between classes.

5 Related Work

Semantic analysis of compounds is an essential preprocessing step for improving on overall downstream NLP applications such as information extraction, question answering, machine translation, and many more (Fares et al., 2016). It has captivated much attention from the computational linguistics community, particularly on languages like English, Dutch, Italian, Afrikaans, and German (Verhoeven et al., 2014b). By rigorously studying Sanskrit compounding system and Sanskrit grammar, analysis of compounds in Hindi and Marathi has been done (Kulkarni et al., 2012). Another interesting approach uses simple statistics on how to automate segmentation and type identification of compounds (Kumar et al., 2010). Nastase et al. (2006) show that from two types of word meaning, namely, based on lexical resources and corpus-based, noun-modifier semantic relations can be learned. Another exciting work by Séaghdha and Copestake (2013) has done noun-noun compound classification using statistical learning framework of kernel methods, where the measure of similarity between compound components is determined using kernel function. Based on *Aṣṭādhyāyī* rules, Kulkarni and Kumar (2013) has developed rule-based compound type identifier. This study helped to get more insights on what kind of information should be incorporated into lexical databases to automate this analysis. Kulkarni and Kumar (2011) proposed a constituency parser for Sanskrit compounds to generate paraphrase of the compound which helps to understand the meaning of compounds better.

Recently, neural models are widely used for different downstream NLP applications for Sanskrit. The error corrections in Sanskrit OCR documents is done based on a neural network based approach (Adiga et al., 2018). Another work used neural models for post-OCR text correction for digitising texts in Romanised Sanskrit (Krishna et al., 2018a). Hellwig and Nehrdich (2018) proposed an approach for automating feature engineering required for the word segmentation task. Another neural-based approach for word segmentation based on seq2seq model architecture was proposed by Reddy et al. (2018), where they have shown significant improvement compared to the previous linguistically involved models. Feedforward networks are used for building Sanskrit character recognition system (Dineshkumar and Suganthi, 2015). Krishna et al. (2018c) proposed energy-based framework for jointly solving the word segmentation and morphological tagging tasks in Sanskrit. The pretrained word embeddings proposed by Mikolov (2013) and Pennington (2014) had a great impact in the field of Natural Language Processing (NLP). However, these token based embeddings were unable to generate embeddings for out-of-vocabulary (OOV) words. To overcome this shortcoming, subword level information was integrated into recent approaches, where character-n-gram features (Bojanowski et al., 2017) have shown good performance over the compositional function of individual characters (Wieting et al., 2015). Another interesting approach (Zhang et al., 2015) is the use of character level input for word-level predictions.

6 Conclusion

For resource-rich languages, deep learning based models have helped in improving the state of the art for most of the NLP tasks, and have now replaced the need for feature engineering with the choice of a good model architecture. In this work, we systematically investigated the following research question: Can the recent advances in neural network outperform traditional hand engineered feature based methods on the semantic level multi-class compound classification task for Sanskrit? We experimented with some of the basic architectures, namely, MLP, CNN, and LSTM, with input representation at the word, sub-word, and character level. The experiments suggest that the end-to-end trained LSTM architecture with FastText embedding gives an F-score of 0.73 compared to the state of the art baseline (0.74) which utilized a lot of domain specific features including lexical lists, grammar rules, etc. This is clearly an important result.

There are many limitations of this study. For instance, what is the effect of the corpus size on the performance? We work with a corpus with less than 5 million tokens, which is negligible compared to 840 billion tokens, on which Glove embeddings for English have been trained. Would a larger dataset have helped? Could methods based on cross-lingual embeddings help in this scenario for transfer learning from languages similar to Sanskrit?

Acknowledgements

The first author would like to thank Pranav Kulkarni, IIT Kanpur, for his helpful feedback and suggestions.

References

- Devaraja Adiga, Rohit Saluja, Vaibhav Agrawal, Ganesh Ramakrishnan, Parag Chaudhuri, K Ramasubramanian, and Malhar Kulkarni. 2018. Improving the learnability of classifiers for sanskrit ocr corrections. In *The 17th World Sanskrit Conference, Vancouver, Canada. IASS*.
- Joost Bastings, Ivan Titov, Wilker Aziz, Diego Marcheggiani, and Khalil Sima'an. 2017. Graph convolutional encoders for syntax-aware neural machine translation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1957–1967, Copenhagen, Denmark, September. Association for Computational Linguistics.
- Yoshua Bengio, Patrice Simard, Paolo Frasconi, et al. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166.

- Antonietta Bisetto and Sergio Scalise. 2005. The classification of compounds. *Lingue e linguaggio*, 4(2):319–0.
- Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.
- Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *J. Mach. Learn. Res.*, 12:2493–2537, November.
- Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. 2017. Supervised learning of universal sentence representations from natural language inference data. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 670–680, Copenhagen, Denmark, September. Association for Computational Linguistics.
- Silvio Cordeiro, Carlos Ramisch, Marco Idiart, and Aline Villavicencio. 2016. Predicting the compositionality of nominal compounds: Giving word embeddings a hard time. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1986–1997.
- Corina Dima and Erhard Hinrichs. 2015. Automatic noun compound interpretation using deep neural networks and word embeddings. In *Proceedings of the 11th International Conference on Computational Semantics*, pages 173–183.
- R Dineshkumar and J Suganthi. 2015. Sanskrit character recognition system using neural network. *Indian Journal of Science and Technology*, 8(1):65.
- Murhaf Fares, Stephan Oepen, and Erik Velldal. 2016. Identifying compounds : On the role of syntax.
- Pierre Geurts, Damien Ernst, and Louis Wehenkel. 2006. Extremely randomized trees. *Machine learning*, 63(1):3–42.
- Brendan S Gillon. 2009. Tagging classical sanskrit compounds. In *International Sanskrit Computational Linguistics Symposium*, pages 98–105. Springer.
- Roxana Girju, Dan Moldovan, Marta Tatu, and Daniel Antohe. 2005. On the semantics of noun compounds. *Computer speech & language*, 19(4):479–496.
- Zellig S Harris. 1954. Distributional structure. *Word*, 10(2-3):146–162.
- Kazuma Hashimoto, Pontus Stenetorp, Makoto Miwa, and Yoshimasa Tsuruoka. 2014. Jointly learning word representations and composition functions using predicate-argument structures. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1544–1555.
- Oliver Hellwig and Sebastian Nehrlich. 2018. Sanskrit word segmentation using character-level recurrent and convolutional neural networks. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2754–2763.
- Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.
- Abhik Jana, Dima Puzyrev, Alexander Panchenko, Pawan Goyal, Chris Biemann, and Animesh Mukherjee. 2019. On the compositionality prediction of noun phrases using poincar embeddings. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Florence (Italy), July. Association for Computational Linguistics.
- Mark Johnson, Thomas L Griffiths, and Sharon Goldwater. 2007. Adaptor grammars: A framework for specifying compositional nonparametric bayesian models. In *Advances in neural information processing systems*, pages 641–648.
- Su Nam Kim and Timothy Baldwin. 2005. Automatic interpretation of noun compounds using wordnet similarity. In *International Conference on Natural Language Processing*, pages 945–956. Springer.
- Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar, October. Association for Computational Linguistics.

- Amrith Krishna, Pavankumar Satuluri, Shubham Sharma, Apurv Kumar, and Pawan Goyal. 2016. Compound type identification in sanskrit: What roles do the corpus and grammar play? In *Proceedings of the 6th Workshop on South and Southeast Asian Natural Language Processing (WSSANLP2016)*, pages 1–10.
- Amrith Krishna, Bodhisattwa P. Majumder, Rajesh Bhat, and Pawan Goyal. 2018a. Upcycle your OCR: Reusing OCRs for post-OCR text correction in Romanised Sanskrit. In *Proceedings of the 22nd Conference on Computational Natural Language Learning*, pages 345–355, Brussels, Belgium, October. Association for Computational Linguistics.
- Amrith Krishna, Bodhisattwa Prasad Majumder, Anil Kumar Boga, and Pawan Goyal. 2018b. An ekalavyaapproach to learning context free grammar rules for sanskrit using adaptor grammar. *Computational Sanskrit & Digital Humanities*, page 83.
- Amrith Krishna, Bishal Santra, Sasi Prasanth Bandaru, Gaurav Sahu, Vishnu Dutt Sharma, Pavankumar Satuluri, and Pawan Goyal. 2018c. Free as in free word order: An energy based model for word segmentation and morphological tagging in Sanskrit. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2550–2561, Brussels, Belgium, October-November. Association for Computational Linguistics.
- Amba Kulkarni and Anil Kumar. 2011. Statistical constituency parser for sanskrit compounds. *Proceedings of ICON*.
- Amba Kulkarni and Anil Kumar. 2013. Clues from as. t. adhyayi for compound type identification. In *Proceedings of the International Sanskrit Computational Linguistics Symposium*. DK Printworld (P) Ltd.
- Amba Kulkarni, Soma Paul, Malhar Kulkarni, Anil Kumar, and Nitesh Surtani. 2012. Semantic processing of compounds in indian languages. *Proceedings of COLING 2012*, pages 1489–1502.
- Anil Kumar, Vipul Mittal, and Amba Kulkarni. 2010. Sanskrit compound processor. In *International Sanskrit Computational Linguistics Symposium*, pages 57–69. Springer.
- John J Lowe. 2015. The syntax of sanskrit compounds. *Language*, 91(3):e71–e115.
- Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.
- Jeff Mitchell and Mirella Lapata. 2010. Composition in distributional models of semantics. *Cognitive science*, 34(8):1388–1429.
- Sivaja S Nair and Amba Kulkarni. 2010. The knowledge structure in amarakośa. In *International Sanskrit Computational Linguistics Symposium*, pages 173–189. Springer.
- Vivi Nastase, Jelber Sayyad-Shirabad, Marina Sokolova, and Stan Szpakowicz. 2006. Learning noun-modifier semantic relations with corpus-based and wordnet-based features. In *AAAI*, pages 781–787.
- Truc-Vien T Nguyen, Alessandro Moschitti, and Giuseppe Ricciardi. 2009. Convolution kernels on constituent, dependency and sequential structures for relation extraction. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 3-Volume 3*, pages 1378–1387. Association for Computational Linguistics.
- Susan Olsen. 2000. Composition. *G. Booij and al*, pages 897–916.
- Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318.
- Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830.
- Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

- Girishkumar Ponkiya, Pushpak Bhattacharyya, and Girish K. Palshikar. 2016. On why coarse class classification is bottleneck in noun compound interpretation. In *Proceedings of the 13th International Conference on Natural Language Processing*, pages 293–298, Varanasi, India, December. NLP Association of India.
- Vikas Reddy, Amrith Krishna, Vishnu Sharma, Prateek Gupta, Vineeth M R, and Pawan Goyal. 2018. Building a word segmenter for Sanskrit overnight. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC-2018)*, Miyazaki, Japan, May. European Languages Resources Association (ELRA).
- Bryan Rink and Sanda Harabagiu. 2010. Utd: Classifying semantic relations by combining lexical and semantic resources. In *Proceedings of the 5th International Workshop on Semantic Evaluation*, pages 256–259. Association for Computational Linguistics.
- Bahar Salehi, Paul Cook, and Timothy Baldwin. 2015. A word embedding approach to predicting the compositionality of multiword expressions. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 977–983.
- Cicero Nogueira dos Santos, Bing Xiang, and Bowen Zhou. 2015. Classifying relations by ranking with convolutional neural networks. *arXiv preprint arXiv:1504.06580*.
- Tobias Schnabel, Igor Labutov, David Mimno, and Thorsten Joachims. 2015. Evaluation methods for unsupervised word embeddings. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 298–307.
- Diarmuid O Séaghdha and Ann Copestake. 2013. Interpreting compound nouns with kernel methods. *Natural Language Engineering*, 19(3):331–356.
- Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany, August. Association for Computational Linguistics.
- Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.
- Emma Strubell, Patrick Verga, Daniel Andor, David Weiss, and Andrew McCallum. 2018. Linguistically-informed self-attention for semantic role labeling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 5027–5038, Brussels, Belgium, October–November. Association for Computational Linguistics.
- Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1556–1566, Beijing, China, July. Association for Computational Linguistics.
- Ian Tenney, Dipanjan Das, and Ellie Pavlick. 2019. Bert rediscovers the classical nlp pipeline. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Florence, Italy, July. Association for Computational Linguistics.
- Stephen Tratz and Eduard Hovy. 2010. A taxonomy, dataset, and classifier for automatic noun compound interpretation. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 678–687. Association for Computational Linguistics.
- Gary Alan Tubb and Emery Robert Boose. 2007. *Scholastic Sanskrit: A handbook for students*. Columbia University Press.
- Ben Verhoeven, Menno van Zaanen, Walter Daelemans, and Gerhard van Huyssteen. 2014a. Automatic compound processing: Compound splitting and semantic analysis for Afrikaans and Dutch. In *Proceedings of the First Workshop on Computational Approaches to Compound Analysis (ComACoM 2014)*, pages 20–30, Dublin, Ireland, August. Association for Computational Linguistics and Dublin City University.

- Ben Verhoeven, Menno van Zaanen, Walter Daelemans, and Gerhard Van Huyssteen. 2014b. Automatic compound processing: Compound splitting and semantic analysis for afrikaans and dutch. In *Proceedings of the First Workshop on Computational Approaches to Compound Analysis, (ComAComA 2014), Dublin, Ireland, August 24, 2014/Verhoeven, B.[edit.]; ea*, pages 20–30.
- John Wieting, Mohit Bansal, Kevin Gimpel, and Karen Livescu. 2015. Towards universal paraphrastic sentence embeddings. *CoRR*, abs/1511.08198.
- Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, pages 649–657.

Appendix I

Embedding	Classifier	A	P	R	F
baseline	ERF	77.39	0.78	0.72	0.74
	RF(N-gram)	75.88	0.83	0.64	0.70
Random	MLP	64.9	0.62	0.55	0.58
	MLP+	65.78	0.61	0.56	0.58
	CNN	65.91	0.60	0.55	0.58
	CNN+	66.15	0.63	0.57	0.59
	LSTM	65.28	0.62	0.53	0.56
	LSTM+	65.88	0.63	0.56	0.59
charcnn	CNN	74.32	0.72	0.65	0.67
	CNN+	74.65	0.73	0.65	0.68
bpe+w2v	MLP	68.53	0.71	0.59	0.62
	MLP+	69.85	0.74	0.58	0.63
	CNN	72.27	0.77	0.61	0.65
	CNN+	71.90	0.74	0.64	0.67
	LSTM	67.48	0.71	0.60	0.63
	LSTM+	68.94	0.73	0.60	0.64
bpe+glove	MLP	71.37	0.75	0.60	0.65
	MLP+	72.12	0.73	0.63	0.66
	CNN	73.01	0.75	0.62	0.67
	CNN+	74.13	0.76	0.64	0.68
	LSTM	69.17	0.72	0.60	0.63
	LSTM+	69.42	0.71	0.62	0.64
FastText*	MLP	74.51	0.72	0.66	0.68
FastText	MLP	76.77	0.75	0.71	0.72
	MLP+	77.06	0.75	0.71	0.72
	CNN	77.04	0.76	0.71	0.73
	CNN+	77.40	0.76	0.70	0.73
	LSTM	77.49	0.76	0.70	0.73
	LSTM+	77.68	0.76	0.71	0.73

Table 5: Evaluation measures are accuracy (A), macro precision (P), macro recall (R) and macro F-score (F). Results reported on test data in table are averaged over 5 runs. ‘+’ sign indicates end-to-end training integrated with classifier.

Appendix II

Embedding	Parameter	Description	Value
CharCNN	maxlen	maximum no of characters in input	25
	Voc-size	Vocabulary size of characters	60
	size	randomly initialized embedding size	1014
w2v	size	Dimensionality of the word vectors	450
	window	Max distance between current & predicted word	15
	BPE-Voc	BPE vocabulary size used for segmentation	100
	sample	down-sampling of more-frequent words	1e-3
	min-count	Ignores all words with frequency lower than this	1
Glove	epochs	Number of iterations over the corpus.	10
	size	Dimensionality of the word vectors	450
	window	Max distance between current & predicted word	20
	BPE-Voc	BPE vocabulary size used for segmentation	100
	min-count	Ignores all words with frequency lower than this	1
FastText	epochs	Number of iterations over the corpus.	70
	size	Dimensionality of the word vectors	350
	window	Max distance between current & predicted word	11
	min-n	Minimum length of char n-grams	2
	max-n	Maximum length of char n-grams	11
FastText*	epochs	Number of iterations over the corpus.	70
	input size	size of FastText features used as input	350

Table 6: Hyper-parameters used in all the systems.

A Machine Learning Approach for Identifying Compound Words from a Sanskrit Text

Premjith B, Chandni Chandran V, Shriganesh Bhat, Soman K.P,
Center for Computational Engineering and Networking (CEN)
Amrita School of Engineering, Coimbatore, Amrita Vishwa Vidyapeetham, India
prem.jb@gmail.com and
Prabaharan P
Center for Cybersecurity Systems and Networks
Amrita School of Engineering, Amritapuri, Amrita Vishwa Vidyapeetham, India

Abstract

In this paper, we propose a classification framework for finding the compound words from a given Sanskrit text. The compound word identification plays a significant role in learning the elucidations of verses in Ayurveda text books which are written in Sanskrit. This process was modelled using several classification algorithms and we examined their efficacy with varying word embedding dimensions. Sanskrit words were vectorized using fastText word embedding method. The results show that the performance of K-Nearest Neighbor is better than other classifiers and the prediction accuracy is 90.38%.

1 Introduction

Compound words (समास) are abundant in Sanskrit. These words are formed by joining two or more nominal words together and it is even possible to have a sequence of more than 10 words in a compound word (En.wikipedia.org, 2015). Computational analysis of a compound word is hard because of its productive nature, unexpressed relationship between the component words and the semantics of a compound word often rely on the contexts (Krishna et al., 2016). Generally, compound words in any language is an open set of words and can be constructed by obeying the sandhi rules in that language. However, the sandhi splitting does not impart the underlying meaning of a compound. To know the meaning of a compound, it is essential to identify the constituent words which in turn helps to learn the relationship between the words (Kumar et al., 2010) (Kulkarni and Kumar, 2011). This can be achieved with the help of word segmentation algorithms (Huet, 2009), (Reddy et al., 2018), (Hellwig and Nehrdich, 2018). These algorithms can segment all the words including compound words and it affects the understanding of texts written in verse (श्लोक) form.

Ayurveda has a long history and almost all the texts are written in Sanskrit. Approximately 67% of the compendium were framed in verse form with the motivation to memorize it easily (Panja, 2013). Despite this advantage, it is difficult for a novice to understand the meaning of a verse accurately. Usually, most of the students who join for Ayurveda course have little knowledge in interpreting such verses. In addition to that, a substantial number of words in each verse belong to the category of compound words. The difficulty level of interpreting the meaning of a verse again increases due to the presence of these complex words. This hardness can be lessened by splitting the compound words into its constituents using aforementioned computational algorithms. However, one can elucidate the whole meaning of a verse only after achieving the Anvaya (अन्वय) form. When we split the compounds before reordering the words may lead to the scattering of the constituent words and hence the reader loses the connection between the words as well as the meaning of the verse. Therefore, a computational tool for identifying the compound words before performing the word segmentation is required for an Ayurveda student to learn the concepts and meaning of a verse precisely.

In this paper, we propose a machine learning tool for distinguishing compound words from non-compound words. This task is modelled as a binary classification problem. Various classification algorithms (Alpaydin, 2009), (Soman et al., 2006) such as Naïve Bayes, K-Nearest Neigh-

bor, Decision Tree, Random Forest, Support Vector Machine, Multi-Layer Perceptron, Logistic Regression and Adaboost were used for the classification. Input to the classifier is a word or a sequence of words and output is the class label which is either compound or non-compound. Input words are represented as vectors using fastText (Bojanowski et al., 2016) word embedding algorithm. We didn't use any linguistic features for this classification.

2 Sanskrit compounds and non-compound words

In English, words can be formed in multiple ways like compounding, prefixation, suffixation etc. (Bauer, 1983), (Rajendran, 2000). However, Sanskrit extensively uses compounding and affixation methods for the formation of words. Phrasal construction is also commonly used as a word formation scheme.

A compound is typically formed by combining two or more entities. These entities have their own existence when they occur independently. Affixation is a different way of word formation in which morphemes are added to a root word to obtain various word forms and is not a productive process. Unlike the components of a compound, constituent morphemes of an affixed word do not exhibit the properties of a normal word. In addition to that, compound words have the following characteristics (Kumar et al., 2010),

- Single word
- Mono case endings
- Mono accent
- Fixed component word order
- Presence of Sandhi

A subset of these properties such as single word, presence of Sandhi etc. is applicable to non-compound words also. This poses a difficulty in computationally discriminating compound words from other words in the language.

3 Method

The problem of identifying compound words from a Sanskrit document was modelled as a binary classification problem (Class labels are compound word class and other word class). Several machine learning algorithms such as Naïve Bayes, K-Nearest Neighbor, Decision Tree, Random Forest, Support Vector Machine, Multi-Layer Perceptron, Logistic Regression and Adaboost classifier were used to model the problem. The major ingredient of any machine learning algorithm is features. There are various approaches for converting words into vectors of which word embedding algorithms were used for feature representation. Word embedding algorithms are built over neural network architectures and are said to learn the semantic as well as syntactic similarities in a corpus. In this paper, fastText was used for embedding words as vectors. The fastText uses sub word information along with the typical word vectors which helps the algorithm to learn the character level as well as the sub word level information from a word. It helps to capture the minute morphological information which are hidden in the words. It is an important aspect for the computational processing of Indian languages because of their morphological richness. Apart from the fastText embedding, we didn't use any linguistic features for the representation of Sanskrit words.

```

Result: 1 - if the word is a compound word or 0 - if the word is not a compound word
Read the data ;
Fill the empty labels with zero (0). Thi label belongs to the class of non-compound words ;
Replace compound word labels with one (1) ;
Tokenize the sentence ;
Apply Fasttext with parameters specified in the Table 4 ;
while Till the last word in the corpus do
    | if If there are more than one word in the sequence then
    | | Obtain the vector representation for the word sequence by taking the mean of the
    | | individual word vectors;
    | else
    | | Take the word embedding for the respective word;
    | end
end
Split the data into train and test data. 80% of the input data was categorized as train set
and the remaining 20% was considered as test data ;
Use a classification algorithm to train the model with train data and train label;
Evaluate the performance of the model using the testing data ;
if A new text comes then
    | Tokenize the text;
    | while For each word do
    | | Get the vector representation;
    | | Predict the class label using the trained mode;
    | | if label == 0 then
    | | | Print "Non-compound word"
    | | else
    | | | Print "Compound word"
    | | end
    | end
else
end

```

Algorithm 1: Algorithm for the identification of the compound words in a Sanskrit text

4 Experiments and Discussions

The compound word classification problem is a binary class problem and the words were represented using Fasttext word embedding algorithm. In this paper, we didn't use any linguistic information for representing the words.

4.1 Dataset description

We collected the tagged dataset from University of Hyderabad website ¹ which contained decomposed compound words along with undecomposed non-compound words. The dataset contains 32,183 tokens and among which 17,479 are unique. The statistics of the dataset is given in Table 1 and 2.

4.2 Discussion

The classification problem was modeled using 8 classification algorithms, which were defined in scikit-learn (Pedregosa et al., 2011) python package, with fastText word embedding. We also tried with Word2vec and Doc2vec methods for word representation, but they failed to obtain vector representation for Out-of-Vocabulary (OoV) words which is very crucial in Natural Language Processing applications. The classification capability of the machine learning algorithms

¹<http://sanskrit.uohyd.ac.in/scl/>

Type of word	Number of words
Compound word	13,009
Non-compound words	19,174
Total	32,183

Table 1: Number of words in compound word class and non-compound words class.

Type of word	Number of unique words
Compound word	12,224
Non-compound words	5,255
Total	17,479

Table 2: Number of unique words in compound word class and other words class.

were evaluated using four metrics - accuracy, precision, recall and f1-score and the performance scores are given in Table 3. The analysis shows that K-Nearest Neighbor (KNN) algorithm performed better than other classification algorithms in terms of all the evaluation metrics. We finalized the evaluation scores after 3 runs of each model.

Another trend we observed from the results was the non-linearity in the data. The data was found to be highly non-linearly separable in the feature space and it causes the linear classification algorithms like Support Vector Machine to perform poorly. These classification performance of these algorithms didn't improve further even after the feature mapping of the data points to an extremely higher dimensional space. Therefore, we came to the conclusion that the only way to enhance the performance of the classifier is to increase the number of data points in the corpus otherwise we have to incorporate certain linguistic features. Figure 2 (a) shows the confusion matrix heat-map. We also executed a 10-fold cross validation over the entire dataset and the cross validation heat-map is given in Figure 2 (b).

The receiver operating characteristic curves of all the algorithms are shown in Figure 1. It also shows the superiority of KNN over other classification algorithms in the identification of compound words. We also tested the performance of the algorithms with various embedding sizes. The analyses showed that the classification accuracy was better when the embedding dimension was 500. The increase in embedding beyond 500 didn't increase the performance of the algorithms to a significant level.

Classifier	Accuracy (in %)	Precision	Recall	F1-score
Naïve Bayes	65.23	0.6837	0.6822	0.6523
K-Nearest Neighbor	90.38	0.8999	0.9162	0.9023
Decision tree	84.37	0.8390	0.8329	0.8356
Random forest	86.78	0.8644	0.8583	0.8610
SVM	60.15	0.3008	0.5000	0.3756
MLP	75.75	0.7511	0.7340	0.7392
Logistic Regression	60.20	0.8009	0.5006	0.3769
AdaBoost	78.14	0.7720	0.7755	0.7736

Table 3: Performance Evaluation of various classification algorithms.

The optimal parameters for the KNN algorithm and fastText are shown in Table 4. A grid search method was used to fix the optimal parameters of KNN whereas the fastText hyper parameters were determined after a series of runs with varying embedding dimensions.

Even though the training dataset contains segmented compounds, the classification model was able to pick out the compounds words from a set of words, which are not decomposed,

Parameters	Value
Number of neighbors	5
Weights	Uniform
Leaf size	30
Word embedding dimension	500
Context Window size	1
Minimum count	1

Table 4: Parameters and their values used with KNN classifier and Fasttext word embedding algorithms.

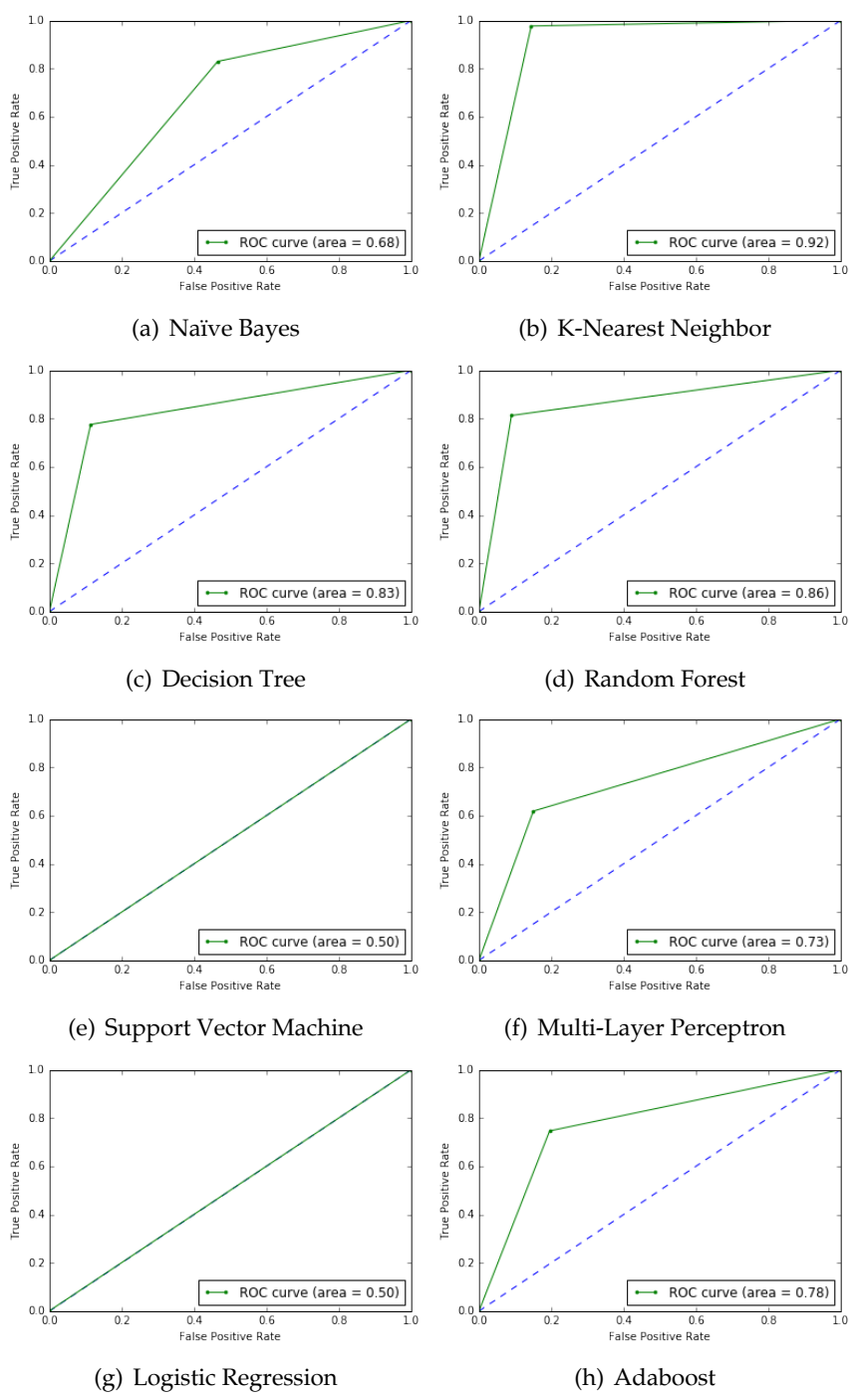


Figure 1: Receiver operating characteristic curves

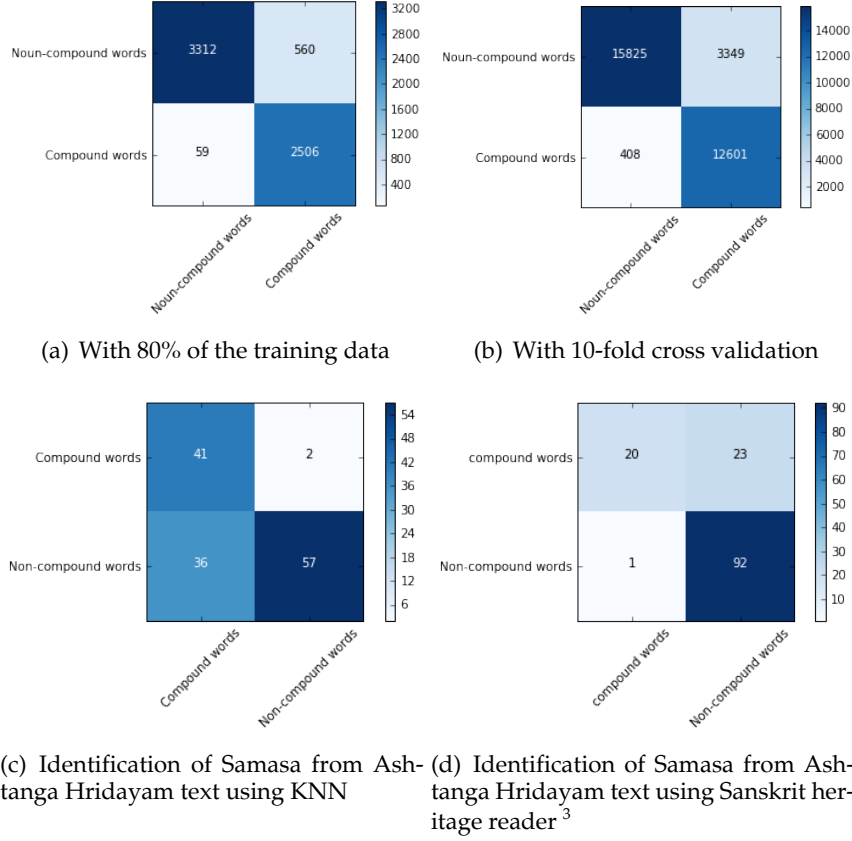


Figure 2: Confusion matrix heat map for the Compound word identification

taken from Ashtanga Hridayam (अष्टाङ्गहृदय). 136 words were selected for testing the potential of the trained model. This test dataset contained 43 compound words and 93 were non-compound words. The model was able to identify 41 compound words correctly, but it failed to classify the non-compounds properly with a prediction accuracy of 61.29%. The confusion matrix heat-map is shown in Figure 2 (c). We also used Sanskrit heritage engine to identify the compound words from the above mentioned test data. This engine was able to pick non-compound words with an accuracy of 98.92%, but at the same time failed to identify the compound words correctly (prediction accuracy = 46.51%). The confusion matrix is depicted in 2 (d).

5 Conclusion

In this paper, we proposed a machine learning approach for compound word identification from a Sanskrit text. Compound words can be constructed by joining two or more independent words and the resulting word conveys a common meaning which may or may not be related to the meanings of the component words. The identification of the compound words is important in learning verses in Ayurveda texts. In this paper, we investigated the implication of various machine learning algorithms with fastText word embedding algorithms in the classification of Sanskrit words into compound and non-compound words. We observed that, K-Nearest Neighbor classifier achieved the highest accuracy of 90.38% for an embedding dimension of 500. We also noticed that data is highly non-linearly separable which is the reason for SVM to give poor results. For this reason, the current model can be upgraded by adding more training examples. Moreover, the classification accuracy can further be increased by incorporating linguistic information which are specific to compounds and non-compounds.

References

- Ethem Alpaydin. 2009. Introduction to machine learning. MIT press.
- Laurie Bauer. 1983. English word-formation. Cambridge university press.
- Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. Enriching word vectors with subword information. arXiv preprint arXiv:1607.04606.
- En.wikipedia.org. 2015. Sanskrit compound. https://en.wikipedia.org/wiki/Sanskrit_compound. [Online; accessed 19-May-2019].
- Oliver Hellwig and Sebastian Nehrlich. 2018. Sanskrit word segmentation using character-level recurrent and convolutional neural networks. In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, pages 2754–2763.
- G rard Huet. 2009. Sanskrit segmentation. South Asian Languages Analysis Roundtable XXVIII, Denton, Ohio (October 2009).
- Amrith Krishna, Pavankumar Satuluri, Shubham Sharma, Apurv Kumar, and Pawan Goyal. 2016. Compound type identification in sanskrit: What roles do the corpus and grammar play? In Proceedings of the 6th Workshop on South and Southeast Asian Natural Language Processing (WSSANLP2016), pages 1–10.
- Amba Kulkarni and Anil Kumar. 2011. Statistical constituency parser for sanskrit compounds. Proceedings of ICON.
- Anil Kumar, Vipul Mittal, and Amba Kulkarni. 2010. Sanskrit compound processor. In International Sanskrit Computational Linguistics Symposium, pages 57–69. Springer.
- Asit Panja. 2013. A critical review of rhythmic recitation of charakasamhita as per chhanda shastra. Ayu, 34(2):134.
- F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12:2825–2830.
- S Rajendran. 2000. Types of word formation in tamil. Linguisticoliterary, pages 323–343.
- Vikas Reddy, Amrith Krishna, Vishnu Dutt Sharma, Prateek Gupta, Pawan Goyal, et al. 2018. Building a word segmenter for sanskrit overnight. arXiv preprint arXiv:1802.06185.
- KP Soman, Shyam Diwakar, and V Ajay. 2006. Data mining: theory and practice [with CD]. PHI Learning Pvt. Ltd.

LDA Topic Modeling for pramāṇa Texts: A Case Study in Sanskrit NLP Corpus Building

Tyler Neill

Leipzig University

Institute for Indology and Central Asian Studies

Schillerstraße 6, 04109

Leipzig, Germany

tyler.g.neill@gmail.com

Abstract

Sanskrit texts in epistemology, metaphysics, and logic (i.e., pramāṇa texts) remain under-represented in computational work. To begin to remedy this, a 3.5 million-token digital corpus has been prepared for document- and word-level analysis, and its potential demonstrated through Latent Dirichlet Allocation (LDA) topic modeling. Attention is also given to data consistency issues, with special reference to the SARIT corpus.

1 Credits

This research was supported by DFG Project 279803509 “Digitale kritische Edition des *Nyāyabhāṣya*”¹ and by the Humboldt Chair of Digital Humanities at the University of Leipzig, especially Dr. Thomas Köntges. Special thanks also to conversation partner Yuki Kyogoku.

2 Introduction

Sanskrit texts concerned with epistemology, metaphysics, and logic (hereafter: pramāṇa texts) have so far been underrepresented in computational work. Digitized texts are available, but supervised word-level analysis is lacking, and so corpus-level operations remain mostly limited to manual plain-text searching.

In response to this, by building on the knowledge-base of the Digital Corpus of Sanskrit (DCS) (Hellwig, 2010–2019) and looking toward a comparably robust future for pramāṇa studies, a 3.5 million-token corpus of pramāṇa texts has been prepared for word-level NLP, and its potential demonstrated through Latent Dirichlet Allocation (LDA) topic modeling. Attention is also given to data consistency issues, with special reference to the SARIT corpus, and with the goal of continuing to improve existing text corpora, including ultimately with rich annotation.

3 Overview

The process of building the present corpus for use with LDA topic modeling can be idealized as the following sequence of nine steps, in three phases:

Phase	Steps
Obtain Data	(1) Collect E-Texts, (2) Choose Versions, (3) Extract XML to Plain-Text
Prep for LDA	(4) Create Doc IDs, (5) Clean Content, (6) Resize Docs, (7) Segment Words
Implement LDA	(8) Model Topics, (9) Query Topics and Documents

Table 1: Workflow Overview

In reality, Steps 3 through 5 were found to frequently overlap, especially in those cases involving more of the data consistency issues discussed in Section 9.

¹See also the earlier FWF project out of which this grew: <https://www.istb.univie.ac.at/nyaya/>.

Nyāya-Vaiśeṣika	Tokens (10 ³)	Bauddha	Tokens (10 ³)	Other	Tokens (10 ³)
Vātsyāyana	45.8	Dharmakīrti	64.5	Jaimini	16.5
Praśastapāda	11.0	Candrakīrti	77.9	Kumārila Bhaṭṭa	50.1
Uddyotakara	117.0	Śāntarakṣita	38.8	Sucarita Miśra	172.8
Jayanta Bhaṭṭa	209.7	Arcaṭa	57.0	Madhva	29.4
Bhāsarvajña	165.5	Kamalaśīla	268.9	Jayatīrtha	364.6
Śrīdhara	95.7	Prajñākaragupta	235.4	(<i>Yuktidīpikā</i>)	56.1
Vācaspati Miśra	314.8	Karṇakagomin	161.5	Māṭhara	17.8
Udayana	149.9	Durveka Miśra	120.1	Patañjali	17.1
Gaṅgeśa	34.7	Jñānaśrīmitra	155.3	Siddhasena	27.1
Pravāduka	29.8	Ratnakīrti	48.8	Abhayadeva Sūri	37.4
Vāgīśvara Bhaṭṭa	41.1	Manorathanandin	108.7	Abhinavagupta	45.6
Total	1242.9	Total	1336.9	Total	834.5

Table 2: Corpus Makeup by Well-Represented Authors

4 Obtaining Data

The approximately 70 pramāṇa texts included in the corpus so far — totaling about 3.5 million tokens — were chosen out of a practical need of the aforementioned *Nyāyabhāṣya* project to be able to more effectively cross-reference relevant texts, above all from the voluminous Nyāya-Vaiśeṣika and Bauddha traditions. A representative sample of authors and their cumulative token counts in the corpus so far is presented in Table 2.² Many of the corresponding e-texts are incomplete, owing to imperfect editing or digitization. In addition, many more such pramāṇa texts are available not only online (easily over twice as much) but also in private offline collections. Even more textual material awaits basic digitization. Owing to a lack of resources, however, virtually no new material could be digitized here, e.g., through OCR and/or double-keyboarding.

4.1 Collecting Available E-Texts

Among existing digital collections, the open online repositories GRETIL and SARIT emerged as most relevant for Nyāya- and Bauddha-centric pramāṇa studies.³ All work based on data derived from these sources can therefore be shared without hesitation. In those few cases where exceptions were made for clearly superior text versions in still-private collections of personal colleagues, original and cleaned versions of such texts cannot yet be shared in full.⁴

²For more detail on this list, along with nearly all data and tools discussed in this paper, see the associated GitHub page: <https://github.com/tylergneill/pramana-nlp>.

³Despite the sophisticated analysis of its other texts, the DCS has few materials directly related to pramāṇa; all are either complete and of small size (e.g. *Vimśatikākārikā* and *-Vṛtti*) or of large size (e.g. *Prasannapadā*, *Abhidharmakośabhāṣya*, *Nyāyabhāṣya*, *Sarvadarśanasamgraha*) and very incomplete (2% or less). Nor do TITUS, The Sanskrit Library, or Muktabodha have significant materials for this genre.

The “Digital Resources” corpus of the University of Hyderabad (<http://sanskrit.uohyd.ac.in/Corpus/>) includes a few such texts (some even sandhi-split) but not enough from the Leipzig project “wishlist” to warrant inclusion in this first round of work; a second round would certainly utilize the digitizations of Vāsudeva’s *Pada-pañcīkā* on Bhāsarvajña’s *Nyāyasāra*, Cinnambhaṭṭa’s *Prakāśīkā* on Keśavamiśra’s *Tarkabhāṣā*, Rucidattamiśra’s *Prakāśa* on Gaṅgeśa’s *Tattvacintāmaṇi*, and Dharmarājādhvarin’s *Tarkacūḍāmaṇi* thereon, among others. Other digital projects of note for pramāṇa studies are: Ono Motoi’s sandhi analysis of Dharmakīrti’s works for KWIC-indexation (now housed on GRETIL and included here); R.E. Emmerick’s indexation database and programs including bhela.exe (now lost to obsolescence); and Yasuhiro Okazaki’s analyzed index of Uddyotakara’s *Nyāyavārttika* (not used here; see: <http://user.numazu-ct.ac.jp/~nozawa/b/okazaki/readme.htm#n.con>).

⁴For example, Uddyotakara’s *Nyāyavārttika*, Bhaṭṭavāgīśvara’s *Nyāyasūtratātparyadīpikā*, and Pravāduka’s (a.k.a. Gambhīravamaśaja’s) *Nyāyasūtravivaraṇa*, provided by Prof. Karin Preisendanz in Vienna, as well as Ernst Steinkellner’s edition of Dharmakīrti’s *Pramāṇaviniścaya* I & II, provided by Hiroko Matsuoka in Leipzig.

4.2 Choosing One E-Text Version Per Work

In comparing and selecting from among digital text versions, data quality, both of edition and digitization, was considered to be of secondary importance relative to two other NLP needs: quantity of text and clarity of structural markup. Only in a few cases was a uniquely available version of a text deemed to be of insufficient quality for inclusion in the analysis presented here.⁵ Occasional exceptions to the one-work-one-file rule were made for base texts quoted in commentaries (e.g., Kaṇāda’s *Vaiśeṣikasūtra* within Candrānanda’s *Tīkā* thereon).

4.3 Extracting XML to Plain-Text

As a third, overlapping criterion, special priority was given to the SARIT corpus, nearly half of which (by file size) consists of pramāṇa texts. Along with these texts’ relatively good data quality, their hierarchical TEI/XML encoding seemed worth trying to exploit for the current purpose. As a positive side-effect of this inclusion, an XSLT workflow was developed to extract the XML to plain-text. For reasons explored below (Section 9.1), multiple transforms were crafted for each text and then daisy-chained together with Python’s *lxml* library. During extraction, rendering of structural elements into machine-readable identifiers was sensitive both to philological understanding of the texts and to the particular NLP purpose at hand.

5 LDA Topic Modeling as Guiding Use Case

LDA topic modeling, as the special purview of the *Nyāyabhāṣya* project’s Digital Humanities specialist Dr. Köntges, was chosen on pragmatic grounds as the best means for stimulating potentially useful NLP experimentation on the envisioned corpus of pramāṇa texts.

In machine learning, topic models comprise a family of probabilistic generative models for detecting latent semantic structures (called topics) in a textual corpus. Among these, the relatively recently-developed LDA model,⁶ characterized by its use of sparse Dirichlet priors for the word-topic and topic-document distributions,⁷ has proven popular for its ability to produce more readily meaningful, human-interpretable results even with smaller datasets and limited computational power. Consequently, the literature on it is already quite vast,⁸ and its software implementations are increasingly numerous and user-friendly.⁹ In recent years, humanities scholars working in a variety of modern and historical languages have used LDA to support their research¹⁰ in an ever-expanding variety of ways, from studying societal trends reflected in newspapers (Nelson, 2011; Block, 2016), to exploring poetic themes and motifs (Rhody, 2012; Navarro-Colorado, 2018), to direct authorship verification (Savoy, 2013; Seroussi et al., 2014). For Classical Sanskrit, it has also been used to scrutinize authorship, albeit indirectly, by helping to control for significance of other parameters.¹¹

⁵For example: GRETIL’s versions of Vyāsatīrtha Rāghavendra’s *Nyāyadīpatarkatāṇḍava* (transcription error-rate too high), Madhva’s *Mahābhāratatattvanirṇaya* (encoding corrupt), and Śākyabuddhi’s *Pramāṇavārttikatīkā* (diplomatic transcription of a damaged manuscript).

⁶The original paper is Blei (2003).

⁷These sparse Dirichlet priors “encode the intuition that documents cover only a small set of topics and that topics use only a small set of words frequently” (Anouncia and Wiil, 2018, p. 271).

⁸See, e.g., David Mimno’s annotated bibliography: <https://mimno.infosci.cornell.edu/topics.html>.

⁹Used here are open-source tools by Dr. Köntges: (Meletē)ToPān (2018), built on the R libraries *lda* and *LDavis*, and Metallo (2018). Other options include Java-based MALLETT and various Python machine-learning packages like *gensim*.

¹⁰This subtle point, that digital humanities methods do not supplant, but support traditional humanities approaches, is made nicely by David Blei (2012):

Note that the statistical models are meant to help interpret and understand texts; it is still the scholar’s job to do the actual interpreting and understanding. A model of texts, built with a particular theory in mind, cannot provide evidence for the theory. (After all, the theory is built into the assumptions of the model.) Rather, the hope is that the model helps point us to such evidence. Using humanist texts to do humanist scholarship is the job of a humanist.

¹¹Low-dimensional topic models ($k \leq 10$) are used by Hellwig (2017) to determine which linguistic features to exclude from authorship layer analysis.

Most important for the present undertaking in corpus building, however, is the basic data requirement in LDA for units at two levels: 1) words and 2) documents.

5.1 Data Need #1: Segmented Words

The first of these, words, is here accepted as equivalent to segmented tokens, namely as provided by the Hellwig-Nehrdich Sanskrit Sandhi and Compound Splitter tool (Hellwig and Nehrdich, 2018), using the provided model pre-trained on the four-million-token DCS corpus.¹² Splitted output from this tool was then modified only slightly, replacing hyphens with space, and these spaces, along with pre-existing spaces, were in turn used to define tokens for this corpus.¹³ For example, *kiñcit*, written as such, would be one token, whereas *kiñ tu* would be two. Efforts should be made to standardize tokenization for this corpus in the future. Similarly, the Splitter’s natural error rate increases if orthography is not standardized, as is the case here.¹⁴ Nevertheless, given the tool’s ease of use, it was seen as preferable, from the humanities perspective, to work with relatively more familiar, human-interpretable units than to work with, for example, raw character n-grams for the LDA modeling.¹⁵ Moreover, LDA being a statistical method, the relatively large amount of data involved (namely, several million tokens) helps to improve the signal-to-noise ratio.

A further possible concern is that this Splitter, as used here, does not perform any sort of lemmatization or stemming, as have been aimed at by, for example, SanskritTagger or the reading-focused systems, especially Reader Companion and Saṃsādhanī.¹⁶ Thus, *arthah*, *arthau*, *arthāḥ*, *artham*, *arthān*, *arthena*, etc. remain distinct items here rather than all being abstracted to a single word, *artha*. However, whether this is a problem is again an empirical question; such stemming may itself result in the loss of some useful information, such as collocations of certain verbs with certain nouns in certain case endings, or genre-specific uses of certain verb tenses.¹⁷ The current Splitter, therefore, provides a sufficient starting point for experimentation.

5.2 Data Need #2: Sized and Coherent Documents

The second requirement for LDA is segmentation of a corpus into properly sized and suitably coherent documents. Whereas the importance of sizing is generally well-known, the necessity of document coherence, as with the issue of stemming just addressed, may depend on one’s specific goals.¹⁸ Toward this end, effort was made by Hellwig to “not transgress adhyāya bound-

¹²Code at <https://github.com/OliverHellwig/sanskrit/tree/master/papers/2018emnlp>.

Splitting the entire pramāṇa corpus took only a few hours on the average-strength personal computer used here: a 2017 MacBook Air with a 1.8 GHz Intel Core i5 processor and 8 GB RAM running macOS High Sierra 10.13.6. For another large-scale demonstration of the Splitter’s power, see Nehrdich’s visualization of quotations within the GRETIL corpus, based on fasttext vector representations of sequences with a fixed length of six tokens, at <https://github.com/sebastian-nehrdich/gretil-quotations>. For a descriptive introduction, see: http://list.indology.info/pipermail/indology_list.indology.info/2019-February/049348.html.

¹³This includes the token counts in Table 2 above. The largest pramāṇa text cleaned and splitted so far (but not yet included in the corpus discussed here) was Someśvara Bhaṭṭa’s *Nyāyasudhā*, on Kumāri Bhaṭṭa’s *Tantravārttika*, sourced from SARIT. It is roughly half a million words long, i.e., one-third the size of the *Mahābhārata*.

¹⁴The default error rate is summarized on the GitHub page as “~15% on the level of text lines”, meaning that “about 85% of all lines processed with the model don’t contain wrong Sandhi or compound resolutions.” For more on the theoretical accuracy limit, as well as on further limitations related to text genres and orthography, see §5.2 “Model Selection” and §5.3 “Comparison with Baseline Models” in Hellwig and Nehrdich (2018), including sentence-accuracies for non-standardized *Nyāyamañjarī* test sentences, esp. 60.2% for the model “*rcNN_{short}^{split}*”. Other immediate drawbacks of using the pre-trained model include: an input limit of 128 characters at a time (compensated for with chunking before splitting) and hyphens indifferently outputted for both intra-compound and inter-word splits (unimportant for LDA).

¹⁵Not yet tested is the possibility of using n-grams alongside segmented words in a “bootstrapping” effort; cp. Dr. Köntges’ upcoming work on LDA bootstrapping with morphological normalization and translation.

¹⁶Respectively: Hellwig (2009), Goyal et al. (2012), and Kulkarni (2009).

¹⁷Cp., e.g., the importance of the Spanish preterite form *fue* in an LDA topic concerned with time in Navarro-Colorado (2018). Cp. also use of the Sanskrit imperfect in narrative literature in Hellwig (2017, passim).

¹⁸For discussion of the importance of size constraints, see Tang et al. (2014), on which the range of words-per-document adopted here is based. For discussion of optimizing topic concentration by using paragraphs to segment documents, as opposed to foregoing all such structural markers (including chapter headings) in favor of simple fixed-length documents for a corpus of 19th-century English novels, see section 6.2 “What is a Document?” in

aries” (2017, p. 145). Here, too, despite the more diverse nature of the śāstric corpus, the challenge of using structural markup was accepted, in part to shed light on encoding issues in this developing body of material. In practice, this meant first seeking out any and all available structural markup — whether in the form of section headers, numbering, whitespace (especially indentation and line breaks), punctuation distinctions like double vs. single *daṇḍas*, or, in the case of SARIT, XML element types and attribute values — and operationalizing it with unique, machine-readable conventions in plain-text. In addition to basic sections, higher-level groupings thereof were also marked (see Section 6 for details).

These preliminary subdivisions of text, or document candidates, could then be automatically transformed into the final LDA training documents using a two-step resizing algorithm: 1) subdivide document candidates which exceed the maximum length, using punctuation and whitespace as lower-level indicators to guide where a safe split can occur; and 2) combine adjacent document candidates whose length is below the minimum, using the grouping markup as a higher-level indicator to guide which boundaries should not be transgressed. The target size range was set at approximately 50–200 words per document,¹⁹ or 300–1000 IAST characters (pre-cleaning), relying on a conservative average of 7 characters per word.²⁰ Finally, the resulting training documents each received a unique, machine-readable identifier automatically reformulated from identifiers manually secured during initial cleaning, so as to facilitate meaningful interpretation during analysis (see, e.g., Section 8).²¹

6 Data Cleaning

The above-described need for maximally useful word- and document-segmentation for LDA prompted the development of practical encoding standards as well as tools for enforcing these standards. This cleaning process involved the greatest amount of manual effort, relying heavily on regular expressions.

Content was standardized to IAST transliteration²² and stored as UTF-8. Orthographic variation, including “optional sandhis”, has unfortunately not yet been controlled for, which does result in systematic Splitter errors;²³ this should either be standardized in the future or else the Splitter model should be retrained for orthographic substyles.

Punctuation was standardized in certain respects, especially dashes and whitespace: em-dash was used only for sentential punctuation; en-dash only for ranges; hyphen only for pre-existing manual sandhi-splits;²⁴ and underscore only for new manual sandhi-splits in rare cases of compounds longer than 128 characters (for the sake of the pre-trained Splitter model). Tab was used only for metrical material; space only for separating words from each other and from punctuation marks; and newline only for marking the start of new sections.²⁵ In this way, these special characters could more effectively help guide document- and word-segmentation before

Boyd-Graber et al. (2017, pp. 70–71).

¹⁹Cp. the use of sections each containing “approximately 30 ślokas” and thus “an average length of 404 words (= lexical units)” in Hellwig (2017, p. 154).

²⁰Such a proxy is necessary because document resizing occurs before word segmentation in this workflow, since punctuation is used for the former and removed in the latter. It is also assumed here that use of IAST instead of, say, SLP1, with the latter’s theoretically preferable one-phoneme-one-character principle, is not problematic, since letters are relatively evenly distributed throughout documents, and since LDA treats words as simple strings.

²¹Cp. use of the Canonical Text Services protocol (<http://cite-architecture.org/>) by the Open Greek and Latin Project (<https://www.dh.uni-leipzig.de/wo/projects/open-greek-and-latin-project/>) for its identifiers. Here, a pragmatic decision was made to opt for simpler, more familiar title abbreviations for now.

²²Transliteration was performed, for reasons of familiarity and also for included meter detection features, with the author’s own small Python library, available on GitHub at <https://github.com/tylergneill/Skrutable>. Other transliteration toolkits, such as that at https://github.com/sanskrit-coders/indic_transliteration, should work equally well.

²³See fn. 14 above.

²⁴This occurred mostly in Ono’s Dharmakīrti texts, which were in any case mechanically re-sandhified during pre-processing in order to ensure more uniform Splitter results. These texts may eventually also prove useful for comparing manual and automatic splitting of *pramāṇa* material.

²⁵For metrical or *sūtra* texts with extensive structural markup, these “sections” could be verse-halves or smaller.

ultimately being filtered out in final preprocessing.

Finally, brackets were also allocated structural markup functions: square brackets were used only for identifying the beginnings of document candidates; curly brackets only for marking higher-level groupings of document candidates; angle brackets only for tertiary structural information useful for reading but not needed for the present purpose; and parentheses only for certain kinds of philological notes, for example on related passages, also not needed here. Other philological material, especially variant or unclear readings, whether found in-line or in footnotes, was either deleted from this corpus or flattened into a single, post-correction text. This required a surprising amount of tedious and often haphazard manual work, which should become more avoidable in the future (for more detail, see Section 9.2).

Cleaned Text	Note
<iti pratyakṣasyānumānatvaparīkṣāprakaraṇam> {avayaviparīkṣāprakaraṇam} [2.1.33] ("sādhyatvād avayavini sandehaḥ") kāraṇebhyo dravyāntaram utpadyata iti sādhyam etat. kim punar atra sādhyam. kim avyatiṛeko 'thāvayavīti. ... ataḥ "sādhyatvād avayavini sandehaḥ" ity ayuktam. itaś ca sādhyatvād avayavini sandeha iti na yuktam ...	End of Previous Prakaraṇa Document Group: New Prakaraṇa Document Candidate Editorial Markup Text Content (In-Line Sūtra Quotation) ...

Table 3: Example of Cleaned Text for NV_2.1.33

To more efficiently enforce these standards, a two-part validator script was written in Python, firstly to check for permitted structural patterns as indicated by bracket markup, and secondly to check for permitted characters and sequences thereof. In case of deviations, the script generated a verbose alert to assist in manual correction.

To recap: After e-texts had been collected and most useful versions chosen, usable structure was sought out and highlighted with in-house markup, including during plain-text extraction from XML where needed. Thereafter, structure and content were laboriously standardized for all texts with the help of a custom-built validator tool. Beyond this point, final preprocessing occurred automatically: Extraneous elements were removed, document candidates were resized, final documents were word-splitted, and the results were reassocated with appropriate identifiers in a two-column CSV file for use with the topic modeling software.

7 Modeling Topics with LDA and Visualizing Structure

One application of LDA topic modeling of philological interest is direct interpretation of the automatically discovered topics. This information is contained in the resulting ϕ table describing the word-topic distributions, and it lends itself well to visualization.

For example, using ToPān (Figure 1) to train an LDA topic model on 67 pramāṇa texts segmented into words and documents as characterized above and with near-default settings²⁶ resulted in fifty topics, all human-interpretable, of which half are presented here, identified both by the respective fifteen top words (adjusted for "relevance")²⁷ and by an interpretive label based on manual scrutiny of the ϕ table.

²⁶ $\alpha = 0.02$, $\eta = 0.02$, and seed = 73, but $k = 50$ and number of iterations = 1000. Twelve most frequent function words (indeclinables and pronouns) were also removed as stopwords for training, à la Schofield (2017), summarized at <https://mimno.infosci.cornell.edu/publications.html>. In addition, but only after training, a further eighty-two function words were removed for the sake of more meaningful interpretation of ϕ values.

²⁷ $\lambda = 0.8$. See Sievert & Shirley (2014), and note log normalization: $\lambda * \log(p(w|t)) + (1-\lambda) * \log(p(w|t)/p(w))$.

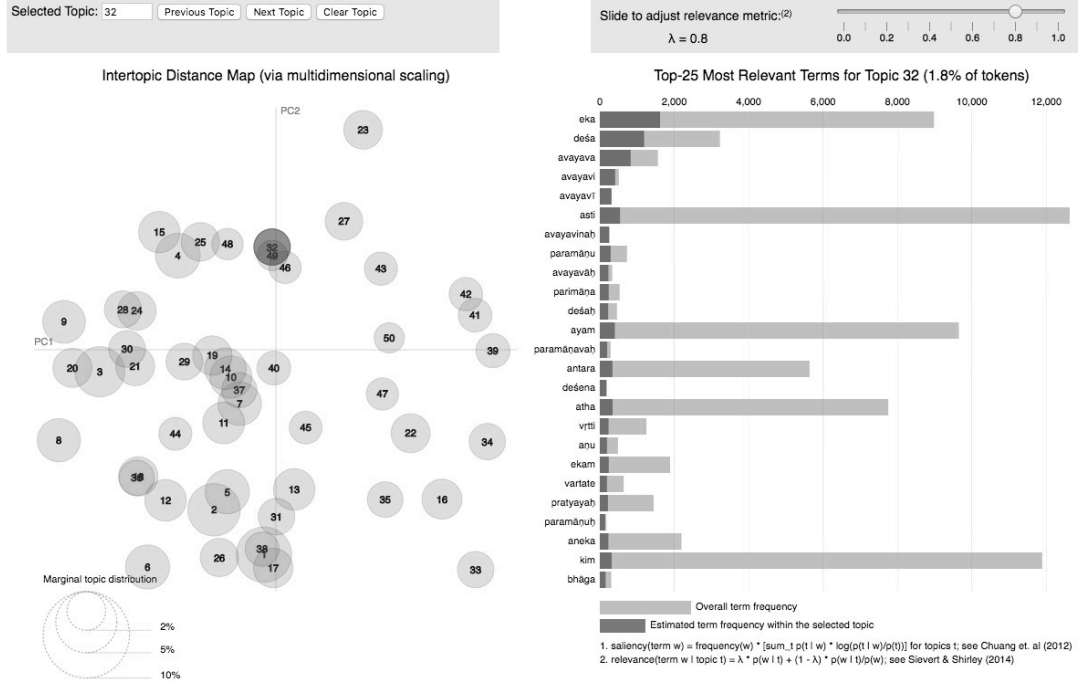


Figure 1: Visualization of Fifty Topics with *LDAvis* in ToPān.

Left: Marginal word-topic probabilities plotted against 2-D PCA of fifty topics.

Right: Top twenty-five words of Topic 32 ($\lambda = 0.8$), with topic and corpus frequencies.

Topic #	Top Fifteen Words	Interpretive Label
4	kārya kāraṇa sahakāri kāryam bīja sāmāgrī svabhāva janana aṅkura śakti śaktiḥ eka hetu janaka sāmāthyam	causation
10	prakāśa nīla prakāśaḥ rūpa ātma rūpam grāhya ātmā jñāna grāhaka ākāra saṃvid prakāśate nīlam ābhāsa	Bauddha non-dual perception
11	jñānam jñāna indriya viśaya pratyakṣam artha jñānasya pratyakṣa viśayam vijñānam akṣa jam rūpa kalpanā grahaṇam	perceptual cognitive process
14	vikalpa ākāra vastu artha ākāraḥ bāhya vikalpaḥ vāsanā rūpa pratibhāsaḥ pratibhāsa vikalpasya viśayaḥ sāmānya viśaya	images and conceptuality
15	bheda bhedaḥ eka bhedaḥ bhinna abheda bhede abhedaḥ bhedena dharma aneka ekam bhedasya bhedaḥ rūpa	difference
16	brahma mokṣa ānanda bhagavat maya śrutiḥ anna śruti viṣṇu jñāna mukti viṣṇuḥ arthaḥ sadā devānām	Dvaita soteriology
17	nigraha pakṣa sādhana sthānam pratijñā artham sthāna para kathā uttara artha tattva siddhāntaḥ doṣa jalpa	Nyāya method
20	abhāva abhāvaḥ bhāva vastu abhāvasya bhāvaḥ anya rūpa virodhaḥ vidhi niṣedha pratiṣedha abhāvayoḥ virodha niṣedhaḥ	affirmation and negation
22	duḥkha sukha rāga duḥkham sukham ātma tattva doṣa dveṣa saṃsāra nivṛttiḥ avidyā pravṛtti rāgaḥ janma	Nyāya soteriology
23	dravya saṃyoga guṇa vibhāga karma kāraṇa dvi saṃyogaḥ guru ākāśa dravyam mahat samavāyi parimāṇa kāraṇam	Vaiśeṣika ontology

Table 4: Philological Interpretation of Ten out of First Twenty-Five LDA Topics. Based on ϕ values, relevance-adjusted ($\lambda = 0.8$), excluding eighty-two further stopwords.

Topic #	Top Fifteen Words	Interpretive Label
26	pramāṇa artha pramāṇam pravṛtti jñānam prāmāṇyam prameya niścaya kriyā niścayaḥ phalam viśaya prameyam prāmāṇya pravṛtṭiḥ	pramāṇa
27	rūpa sparśa pṛthivī cakṣuḥ gandha indriya śabda rasa guṇa pradīpa śrotra grahaṇam tejaḥ śabdaḥ indriyam	sensation
29	sat asat kāraṇa kāraṇam kāryam kārya sattā asataḥ cit sarvam utpatti prak sataḥ utpattiḥ sattvam	Sāṃkhya pre-existent effect
32	eka deśa avayava avayavi avayavī avayavinaḥ paramāṇu avayavaḥ parimāṇa deśaḥ paramāṇavaḥ antara deśena vṛtti aṇu	atoms, parts, and wholes
35	phala svarga vidhi phalam karma hiṃsā kāmāḥ vidhiḥ sādhana putra yāga artha vidheḥ yajeta codanā	Vedic sacrifice
36	rajata mithyā bādha satya rajatam svapna bādhya sākṣi bādhaḥ sat śukti jñāna asat bhrānti mithyātvam	error
38	prāmāṇyam veda āpta prāmāṇya pramāṇa artha āgama aprāmāṇyam vākya pramāṇam puruṣa doṣa vakṛ apauruṣeya svatas	trustworthy speech
39	pañca prakṛti vyaktam rajaḥ pradhānam prakṛtiḥ avyaktam vikāra tamaḥ sattva mahat avyakta sargaḥ vṛtiḥ tanmātrāṇi	Sāṃkhya metaphysics
40	smṛti pūrva smṛtiḥ anubhava smaraṇam smaraṇa saṃskāra smṛteḥ anubhavaḥ kāla saṃskāraḥ anubhūta viśaya jñānam jñāna	experience and recollection
41	karma śarīra śarīram icchā īśvaraḥ īśvara prayatna dharma śarīrasya deha adharma phala karmaṇaḥ cetanā bhoga	karma
42	bhavanti viśeṣāḥ dharmāḥ sarve santi hetavaḥ syuḥ viśeṣa arthāḥ yeṣāṃ kecid śabdāḥ anye teṣu bhāvāḥ	plural words
43	indriya manaḥ ātma manasaḥ śarīra yugapad jñāna sukha viśaya artha icchā cakṣuḥ jñānam sannikarṣa indriyāṇām	Nyāya prameyas related to the self
45	kriyā kāraṇa kartṛ karma karaṇa artha vyāpāra vyāpāraḥ dhātu karaṇam arthaḥ bhāvanā kriyām karoti kriyāyāḥ	action
47	aham puruṣa puruṣaḥ buddhi puruṣasya ātmā artham buddhiḥ arthaḥ ātmanaḥ ātmānam buddheḥ prakṛtiḥ mama bhoktā	Sāṃkhya on self and other
48	viśeṣaṇa viśeṣya samavāyaḥ ghaṭa samavāya bhū sambandha ghaṭaḥ viśeṣaṇam viśiṣṭa ādhāra sambandhaḥ paṭa paṭaḥ guṇa	qualification

Table 5: Further Philological Interpretation of Fifteen out of Remaining Twenty-Five LDA Topics.

8 Using Topics for Information Retrieval

Another computational application of interest to philologists, that of calculating similarity among portions of text, can to some extent also be approached directly with these same topic modeling results, namely by vectorizing documents according to their topic distributions and measuring their distance from each other in topic-space.²⁸ The relevant information for this is found in the θ table describing the topic-document distributions.

For example, using *Metallo* with default settings²⁹ to compare documents according to their Manhattan distance in topic-space, one can query topics and documents of interest to a particular research question — here, say, the present author’s own dissertation topic: the ontological whole (*avayavī*) in Bhāsarvajña’s *Nyāyabhūṣaṇa*. Manual inspection of the fifty discovered topics quickly reveals that Topic 32 (see Table 5 above) will likely be relevant. *Metallo* then easily generates a list of arbitrarily many documents best exemplifying this topic, or in other words, documents closest to that particular basis vector in the topic-space (see Table 6). It also allows

²⁸Ideally, topic distribution would be only one among a number of linguistic features used to characterize documents for information retrieval. The implementation here is therefore mainly for the purpose of demonstration.

²⁹Significance parameter = 0.1. Note also that by default, all topics are weighted equally.

for direct querying of any desired document, say, $NBh\bar{u}_{104,6^{\wedge}1}$ ³⁰ (beginning of the *avayavī* discussion), for arbitrarily many documents closest to it in topic-space, as seen in Figure 2 and Tables 7 and 8.

Rank	Document Identifier	Topic 32
1	$NV_{4.2.7}$	98.8%
2	$NVTT_{4,2.10.1-4,2.10.2^{\wedge}2-4,2.11.1}$	98.7%
3	$NV_{2.1.31^{\wedge}2}$	98.4%
4	$NSV_{4.2.7}$	98.4%
5	$NV_{2.1.32^{\wedge}4}$	97.2%
6	$NV_{2.1.32^{\wedge}8}$	95.4%
7	$NBh_{2.1.36.1-2.1.36.2}$	95.1%
14	$NSV_{4.2.8-4.2.9}$	90.6%
15	$NSV_{4.2.16}$	90.3%
20	$NSV_{4.2.11-4.2.13}$	88.3%
21	$NBh_{2.1.36.3}$	87.9%
22	VVr_{12}	87.8%
24	$VVr_{14^{\wedge}2}$	87.0%
25	$VVr_{14^{\wedge}1}$	87.0%
26	$NBh_{4.2.16.1-4.2.16.3}$	86.6%
27	$NBh_{2.1.31.3-2.1.31.5}$	86.4%
35	$NVTT_{2,1.32.1^{\wedge}7}$	82.6%
39	$NM_{9,2.430.325}$	80.7%
40	VVr_{13}	80.6%
43	$NBh\bar{u}_{106,3}$	80.0%
46	$NVTT_{4,2.7.1}$	79.3%
48	$NTD_{4.2.7}$	79.3%
51	$NBh\bar{u}_{111,24^{\wedge}1}$	78.8%
52	$NVTT_{4,2.25.1^{\wedge}3}$	78.6%
56	$NTD_{4.2.10}$	77.0%
65	$PVV_{1.87,0-1.87,1}$	75.5%
72	$PVin_{1.38.3}$	74.2%
75	$NK_{59.4^{\wedge}2}$	74.1%
76	$NSu_{2.2.66cd.3-2.2.66cd.4}$	74.0%
81	$NTD_{2.1.39}$	72.9%
86	$NTD_{4.2.15}$	71.5%
91	$VNT_{80,1^{\wedge}2}$	70.5%
94	$NBh\bar{u}_{104,6^{\wedge}2}$	70.1%
97	$NM_{9,2.430.322}$	69.8%
100	$YŚ_{3.44.5-3.44.6}$	69.3%

Table 6: Selected Documents in which Topic 32 is Most Dominant.

Top four only shown for NV , $NVTT$, NSV , NBh , VVr , NTD . (Sixty-five more not shown.)

All shown for NM , $NBh\bar{u}$, PVV , NK , NSu , VNT , $YŚ$.

³⁰As seen here by the “ $\wedge 1$ ” notation marking a document automatically subdivided in resizing, queryable documents are currently limited to those somewhat artificial ones used in modeling. It is also possible to extrapolate to new data, but this has not yet been done here.

NBhū_104,6¹

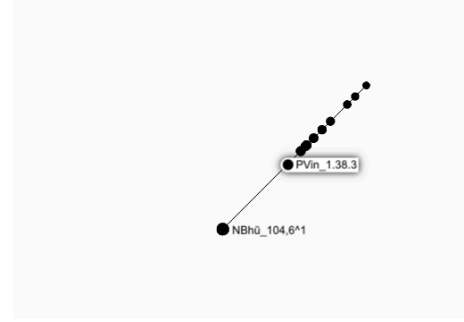
Significant distance set to: 0.1

Important Topics:

Topic32 eka_deśa_avayava_asti_avayavi_ayam_atha: 67.91%

Text:

nanu ca asthūlasya arthasya grāhakaṃ na tu jñāna ākārasya sthāulyam asti iti atas na jñāna ātmakam sthūlam grāhyam iti jñānāt artha antaram sthūlam sutarām na sambhavati tathā hi na tāvat ekaḥ avayavi tathā sati tasya pāṇi ādi kampe sarva kampa prāpteḥ akampāne vā cala acalayoḥ prthak siddhi prasaṅgāt vastra udaka vat ekasya ca āvaraṇe sarvasya āvaraṇa prasaṅgāt abhedāt na vā kasyacid āvaraṇam iti avikalama drśyeta avayavasya āvaraṇam na avayavinaḥ iti abhyupagame api arddha āvaraṇe api anāvṛta tvāt prak iva asya darśana prasaṅgaḥ avayava darśana dvāreṇa avayavi darśanam iti asmin api pakṣe sarvathā avayavinaḥ apratipatti prasaṅgaḥ sarva avayavānām draṣṭum aśakyavāt katipaya avayava darśanāt avayavi darśane yadvat atra avayava darśane api tathābhūtasya eva darśana prasaṅgaḥ rakte ca ekasmin avayave yadi avayavi raktaḥ tadā anya avayava sthaḥ api raktaḥ eva drśyeta no ced tadā sarva avayava rāge api avayavi araktaḥ eva upalabhyeta



PVin_1.38.3

na api sthūlaḥ ekaḥ viśayaḥ tathā avabhāsī pāṇi ādi kampe sarvasya kampa prāpteḥ akampāne vā cala acalayoḥ prthak siddhi prasaṅgāt vastra udaka vat ekasya ca āvaraṇe sarvasya āvaraṇa prasaṅgaḥ abhedāt na vā kasyacid āvaraṇam iti avikalama drśyeta avayavasya āvaraṇam na avayavinaḥ iti ced ardhā āvaraṇe api anāvṛta tvāt prāgyat asya darśana prasaṅgaḥ avayava dvāreṇa tad darśanāt adrṣṭa avayavasya asya apratipattiḥ iti ced na bheda abhāvena sarvathā apratipatti prasaṅgāt sarva avayavānām ca yugapad draṣṭum aśakya tvāt sarvadā ca asya adarśana prasaṅgaḥ katipaya avayava pratipattau darśane alpa avayava darśane api tathā sthūlasya darśanam syāt rakte ca ekasmin rāgaḥ araktasya vā gatīḥ avayava rāge vā avaya vi rūpam araktam iti rakta āraktam drśyeta tasmāt na ekaḥ kaścid arthaḥ yaḥ vijñānam sarūpayati

Rank: 1

Distance: 20.16

Important Topics:

Topic32 eka_deśa_avayava_asti_avayavi_ayam_atha: 74.22%

Topics with significant distance:

Figure 2: Screenshot of Metallo “view” Query on Document NBhū_104,6¹

Rank	PVin	NBh	NBhū	NV
0			104,6 ¹	
1	1.38.3			
7			104,6 ²	
13		4.2.24.3		
15			110,12	
17			106,3	
18		4.2.16.1–4.2.16.3		
20		2.1.36.7		
25				2.1.31 ¹⁰
26				2.1.33 ³⁰
27				2.1.32 ⁴
28				2.1.33 ³¹
30		2.1.36.4		
31				4.2.26
34				2.1.36 ³
35				2.1.33 ³³
36				4.2.25 ³
37			123,21	
41				2.1.31 ³
42				1.1.14 ¹⁴
43			130,15 ²	
45		2.1.36.3		
47		2.1.35.3–2.1.35.4		
49				4.1.13

Table 7: Selected Documents Closest to NBhū_104,6¹ in Topic-Space.

Emphasis on: PVin, NBh, NBhū, NV.

Not shown: NM, NSV, NSu, NTD, VVr, NK, NVTT, ĀTV, PVV.

Rank	Document Identifier	Text Preview (Segmented, Unproofread)
0	<i>NBhū</i> _104,6 ¹	... jñānāt artha antaram sthūlam sutarām na sambhavati tathā hi na tāvat ekaḥ avayavī tathā sati tasya pāṇi ādi kampe sarva kampa prāpteḥ akampane vā cala acalayoh pṛthak ...
1	<i>PVin</i> _1.38.3	na api sthūlaḥ ekaḥ viṣayaḥ tathā pāṇi ādi kampe sarvasya kampa prāpteḥ akampane vā cala acalayoh pṛthak siddhi prasaṅgāt vastra udaka vat ...
13	<i>NBh</i> _4.2.24.3	... uktam ca atra sparśavān aṇuḥ sparśavatoḥ aṇvoḥ pratighātāt vyavadhāyakaḥ na sāvayava tvāt sparśavat tvāt ca vyavadhāne sati aṇu saṃyogaḥ na āśrayam vyāpnoti ...
18	<i>NBh</i> _4.2.16.1–4.2.16.3	... niravayava tvam tu paramāṇoḥ vibhāgaiḥ alpatara prasaṅgasya yatas na alpīyaḥ tatra avasthānāt loṣṭasya khalu pravibhajyamāna avayavasya alpataram alpatamam ...
20	<i>NBh</i> _2.1.36.7	... bhavataḥ tena vijñāyate yat mahat tat ekam iti aṇu amahatsu samūha atīśaya grahaṇam mahat pratyayaḥ iti ced saḥ ayam aṇuṣu mahat pratyayaḥ atasmin tat iti pratyayaḥ bhavati ...
7	<i>NBhū</i> _104,6 ²	vṛtti anupapatteḥ ca avayavī na asti tathā hi gavi śṛṅgam iti laukikam śṛṅge gauḥ iti alaukikam tatas yadi avayavini avayavāḥ varttante tadā ...
15	<i>NBhū</i> _110,12	nanu eka avayava kampane api anya avayavānām akampanāt asti cala acala tvam tena bheda siddhiḥ tatas kim aniṣṭam yadi nāma avayavānām cala acala tvena bhedaḥ tatas ...
17	<i>NBhū</i> _106,3	itas ca na asti avayavī buddhyā vivecane anupalambhāt na hi ayam tantuḥ ayam tantuḥ iti evam buddhyā pṛthak kriyamāṇeṣu avayaveṣu tad anyaḥ avayavī pratibhāti ...
25	<i>NV</i> _2.1.31 ¹⁰	... atha manuse na asmābhiḥ avayavi dravyāṇi kāni cit pratipadyante kim tu teṣu eva parama aṇuṣu paraspara pratyāsatti upasaṃgrahaṇa saṃsthāna viśeṣa avasthiteṣu ...
26	<i>NV</i> _2.1.33 ³⁰	... na tantavaḥ tantūnām avayavāḥ iti viruddhaḥ artha antara pratyākhyānāt ca avayavaḥ avayavī iti etat na syāt yat api idam ucyate ye avayavāḥ avayavinaḥ artha antaram ...
27	<i>NV</i> _2.1.32 ⁴	tasmāt ekasmin na kārtsnaḥ vartate iti na api eka deśena vartate na hi asya kāraṇa vyatirekeṇa anye eka deśāḥ santi sa ayam eka deśa upalabdhau avayavi upalabhyamānaḥ na kṛtsnaḥ upalabhyate ...

Table 8: Detail on Ten Documents Close to *NBhū*_104,6¹ in Topic-Space. In this case, *PVin*_1.38.3, ranked first, is in fact the direct source of the non-verbatim quotation.

9 Data Consistency Issues

These tentative results, encouraging though they may be, stand to be improved not only through more sophisticated application of NLP methods, but also through increased attention to data consistency. Besides systematic tokenization and orthography issues (addressed in Section 5.1) and unsystematic typographical or even editing errors (not yet prioritized here), three additional sets of systematic data consistency issues were revealed through the process of preparing this corpus. These are advanced here as the low-hanging fruit of improving textual data for future Sanskrit NLP work. The first issue applies at the level of documents and relates to being able to effectively manipulate these through meaningful identifiers, while the second and third are concerned with data loss at the level of individual words. In each case, special attention is paid to the SARIT texts so as to further encourage their use for NLP purposes.

9.1 Structural Markup and Identifiers

The essential structural challenge in such corpus-level computational work is to be able to refer to every single piece of text in the corpus with a unique and, if at all possible, meaningful identifier, in order to be able to effectively coordinate retrieval and human use after processing. In the texts used here, however, structural markup for the purpose of creating such identifiers was often less than easily available. Sometimes, only physical features of the edition, rather than logical features of the text, were found to be marked, even when the latter might have been possible (e.g., the digitization of Durveka Miśra’s *Hetubinduṭīkāloka* lacking the structure of the underlying *Hetubindu* or *Hetubinduṭīkā*). Sometimes, numerical structural markup was only found mixed in among textual content (e.g., Abhinavagupta’s *Īśvarapratyabhijñāvivṛtivimarśinī*). Sometimes, important section information was marked only with the verbal headers or trailers of the printed edition rather than with numbers (e.g., Vinītadeva’s *Nyāyabinduṭīkā*).

Of course, some markup issues may reflect citation difficulties within the philological field itself; for example, citation conventions for texts with continuously interwoven prose and metrical (or aphoristic) material may be more varied than for other texts.³¹ Similarly, when (or if) creating paragraphs in such prose texts, editors must often make a substantial interpretive departure from the available manuscript evidence. Thus, as the philological understanding of the interrelationships among parts of a given text gradually improves, so too might the corresponding structural markup in digitized texts also be expected to do so.³²

In other cases, however, it seems that basic encoding work has just been left undone, whether for lack of time or resources, or through a preference for adhering literally to the source edition, which, for better or worse, allows one to postpone further questions concerning structural annotation. Looking forward, insofar as these digitizations can receive more attention, and as more computational projects are attempted with them, the field should continue³³ to gradually move in the direction of the Canonical Text Services protocol. This protocol encourages explicit and usually numerical reference conventions for the sake of unambiguous citation and automatic processing, and its implementation has been admirably exemplified in recent years (also with TEI/XML markup) by the Open Greek and Latin Project (OGL).³⁴

Structural Markup and Identifiers in SARIT

The existing SARIT stylesheet transforms proved difficult to understand and adapt for the current purposes, and thus it was decided to utilize the situation as an exercise in understanding the diversity of structures encoded in that corpus. Experimentation quickly revealed that, in contrast to texts in the OGL corpus, where a single XPath expression in the <TEIheader> explicitly identifies the depth at which textual information will be found, the texts in the SARIT corpus varied so much in their use of main structural elements — <div>, <p>, <lg>, <quote>, <q>, etc. — that it was not possible to write and use straightforward XSL transforms that could apply to multiple files, much less to use the XML library of a given programming language (e.g. Python or Golang) to easily unmarshal the structure and expose the textual data.³⁵ For example, while for some texts, logical structure was encoded using only a single level of <div> elements (e.g., sūtra sections in Vātsyāyana’s *Nyāyabhāṣya*), for others, any number of levels of nested <div>s could be used for the same purpose (e.g., Jñānaśrīmitra’s *Nibandhāvali* and Prajñākaragupta’s *Pramāṇavārttikālaṅkāra*). Meanwhile, still other texts were structured not

³¹Take, for example, Prajñākaragupta’s *Pramāṇavārttikālaṅkāra*. It’s not always clear whether one should refer to a piece of the prose commentary with the help of a numbered Dharmakīrti verse quoted nearby, or with Prajñākaragupta’s own nearby and numbered verses, or simply with the edition page and line numbers.

³²Cp., e.g., *Nyāyabhāṣya* topical headers and paragraph divisions by editor Yogīndrānanda (1968) with those of S. Yamakami (2002) for the avayavī section at <http://www.cc.kyoto-su.ac.jp/~yamakami/synopsis.html>.

³³For thoughts so far, see, e.g., Ollett (2014).

³⁴See, e.g., the OGL texts in the Scaife Viewer online reading environment: <https://scaife.perseus.org/>.

³⁵Cp. such a mass unmarshalling script for OGL texts at <https://github.com/ThomasK81/TEItoCEX>.

Cp. also the simple, two-level, chapter-verse structure of DCS data as exported from the SanskritTagger in XML form, reflecting top-down, NLP-driven decision making from the very beginning. (A version of the Tagger capable of performing this export was secured with the kind help of Oliver Hellwig.)

according to logical structure but rather according to physical structure of the edition. For example, Jayantabhaṭṭa’s *Nyāyamañjarī*, printed on the top halves of pages in the book, was therefore encoded as <quote> elements inserted at unpredictable depths, i.e., within <p> or <q> elements, within the supervening modern *Ṭippanī* commentary, following page breaks. This proved especially difficult to understand and deal with from a perspective seeking natural language. Thus, new transforms had to be individually crafted for each of the fifteen SARIT texts used. While this does provide temporary access to the plain-text information, suggestions will be made to modify the SARIT source files so that they adhere to a smaller number of structural patterns that can be explicitly noted in their respective headers.

9.2 Editorial Markup

Also reflecting a still-developing state of editing and understanding, many digitizations of printed editions literally reproduce or add editorial markup — especially variant readings, including additions, deletions, and substitutions of variable length — which can be quite idiosyncratic and not always thoroughly explained in accompanying digitization metadata. For example, see the table below, based on Durveka Miśra’s *Hetubinduṭṭhāloka* (parenthetical editorial notes turn out to be reporting on the corresponding text in Arcaṭa):

Page	Text (with Editorial Note)	Suggested Change
254	... tadutpattāv eveti(tpattyā veti) vivakṣitam	replacement
279	a(nya)thā “nirvikalpakabodhena...	insertion
280	anadhigacchann iti (gaṃcchadi)ti	none?

Table 9: Examples of Inconsistent Editorial Markup

Insofar as it is not possible to automatically flatten such alternatives into a single text, the flow of natural language will be compromised, and words lost. The straightforward solution is to anticipate such flattening — either through XML transforms or simple search-and-replace routines — with consistent use of some unambiguous notation. This does, however, of course require substantial additional investment of time and expertise. Extensive notes taken during the corpus cleaning here should hopefully contribute to such improvements for the future.

Editorial Markup in SARIT

The use of <choice> elements in XML is a perfect way to address this situation, yet the SARIT texts were found to apply this solution only unevenly, leaving many instances of editorial markup uninterpreted as found in the printed edition. For example, as reported in the metadata of Karṇakagomin’s *Pramāṇavārttikavṛttiṭīkā*, although many round brackets (i.e., parentheses) and square brackets have been successfully interpreted — as <ref>, <note type=‘correction’>, and <supplied resp=‘#ed-rs’> — others have simply been left as is: “All other round brackets (227 occurrences) were encoded as <hi rend=‘brackets’>” and “All other square brackets (19 occurrences) were encoded as <hi rend=‘squarebrackets’>”. In other cases (e.g., Vācaspati Miśra’s *Tattvavaiśārādī*), these editorial notes were left untouched. Such cases require further philological scrutiny in order to allow for consistent extraction of natural language.

9.3 Whitespace

In the printed representation of Sanskrit texts, one can distinguish between two basic conventions, or perhaps styles, of using whitespace between words: 1) maximal use of whitespace, usually associated with Roman transliteration and prioritizing separate phonemes and words, and 2) conservative use of whitespace, usually associated with Indic scripts and prioritizing ligatures as found in the underlying manuscript tradition. Each style has its strengths and weaknesses, e.g., assuming more work on the part of the editor or digitizer and less on the part of the reader (first style) or vice versa (second style). The point of distinguishing these two

styles, however, is not to advocate for one over the other,³⁶ but rather to distinguish both from outright spacing errors. That is, it should be trivial for an NLP researcher to quickly filter out all markup and obtain a clean, consistent representation of either one style or the other.

In practice, however, this was often found not to be the case, suggesting that whitespace has not yet been conceived of as containing as much information as other character types. To take but one small example from the digitization of Candrakīrti’s *Prasannapadā* (prose section preceding 27.19):

... saṃsāraprabandhamupalabhya śāśvata mātmanaṃ parikalpayāmaḥ |

Here, the “conservative” style is found, but with a spurious space. Each such instance represents the effective loss of one or more words in segmentation. Many of these errors do follow certain patterns, such that regular expressions can be part of a standardization solution, but there are limits to what such language-blind methods can detect.³⁷

Whitespace in SARIT

For its own part, SARIT experiences this same whitespace consistency issue, but it also introduces novel difficulties with its handling of in-line annotations, i.e., XML node() elements placed within text() elements. For example, consider the following six representative examples in the digitization of Mokṣākaragupta’s *Tarkabhāṣā* (transliterated, XML elements simplified):

Space	Proper	Improper
Left	kumbhakārasya <note n=“45-1”/>kartṛtvam	pratyakṣa <note n=“4-1”/>mabhidhīyate
Right	-mataśrutyai<note n=“1-1”/> tarkabhāṣā	balāda<note n=“5-2”/> bhyupagatam
None	parokṣatva<note n=“18-1”/>pratipādanāya	-pādaiḥ<note n=“41-0”/>kāryatvasya

Table 10: Examples of Inconsistent Whitespace in SARIT Texts

It thus becomes impossible to systematically extract the expected result.

Particularly problematic were <lb> (and to a lesser extent <pb>) elements containing the break=“no” attribute, as these were not infrequently found to occur adjacent to other <lb> or <pb> elements not possessing this attribute, as well as adjacent to simple whitespace, thereby rendering the attribute ineffective and compromising word segmentation. A particularly dramatic example is found in Jñānaśrīmitra’s *Nibandhāvalī* (complex whitespace simplified):

... pariṇāma<lb break=“no”/> <lb/> <pb n=“257”/> <lb/>paramparāparicayasya ...

In such cases, ensuring proper segmentation necessitates removal of competing elements, which can then cause problems of its own, e.g., if line number counts are required for constructing identifiers. On the other hand, this break=“no” attribute was sometimes simply not used when it should have been. For example, in Śāntarakṣita’s *Vādanyāyāṭikā* (67,4–5; element simplified) (also observe not one but two whitespaces):

sadādyaviśeṣavi <lb/> ṣayā ...

Fortunately, once identified, fixing such problems is relatively easy with the help of regular expressions and SARIT’s recommended Git-based workflow, although again, expertise and time are required. The XSLT workflow described above can also be further modified to help diagnose such issues and assess how much progress has been made in this direction at any given point.

10 Conclusion

This demonstration of working through a certain subset of Sanskrit pramāṇa texts with LDA topic modeling has been of a preliminary character. Nevertheless, it provides a valuable window

³⁶From the perspective of NLP, machine-learning-based systems, ever more the rule rather than the exception, can be made to handle both separately, just as OCR systems can be trained for multiple fonts.

³⁷E.g., a regex built to find a final consonant migrating to the beginning of the next word, as in the example given, would fail to distinguish between “-m ucyate” and “mucyate”, both valid sequences, depending on context.

onto the state of digitization of a large number of e-texts of ever-increasing importance to the scholarly community and shows what potential they have for further computational research. Moreover, issues encountered with LDA and pramāṇa texts in particular should generalize well to many other NLP methods and Sanskrit subgenres. Until a database of supervised word-segmentation, such as found in the DCS, is secured also for such specialized texts, perhaps with the help of a collaborative, online annotation system, the remarks here will hopefully help interested parties continue to improve digitization workflows in ways that anticipate the kind of accessible, citable, machine-actionable text — to be processed, for instance, with an unsupervised segmenter — that will be most needed for a variety of corpus-linguistic and information retrieval applications in the future.

References

- S. Margret Anouncia and Uffe Kock Wiil. 2018. *Knowledge Computing and its Applications: Knowledge Computing in Specific Domains*, volume 2. Springer Nature Singapore.
- David Blei, Andrew Ng, and Michael Jordan. 2003. Latent Dirichlet allocation. *The Journal of Machine Learning Research*, 3:993–1022.
- David Blei. 2012. Topic modeling and digital humanities. *Journal of Digital Humanities*, 2(1), Winter.
- Sharon Block. 2016. Doing more with digitization. *Common-place.org*, 6(2), January.
- Jordan Boyd-Graber, Yuening Hu, and David Mimno. 2017. Applications of topic models. *Foundations and Trends® in Information Retrieval*, 20(20):1–154.
- Pawan Goyal, Gérard Huet, Amba Kulkarni, Peter Scharf, and Ralph Bunker. 2012. A distributed platform for Sanskrit processing. In *24th International Conference on Computational Linguistics (COLING), Mumbai*.
- Oliver Hellwig and Sebastian Nehrdich. 2018. Sanskrit word segmentation using character-level recurrent and convolutional neural networks. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2754–2763, Brussels, Belgium, October-November. Association for Computational Linguistics.
- Oliver Hellwig. 2009. SanskritTagger: A stochastic lexical and POS tagger for Sanskrit. In Gérard Huet, Amba Kulkarni, and Peter Scharf, editors, *Sanskrit Computational Linguistics*, pages 266–277.
- Oliver Hellwig. 2010–2019. DCS - The Digital Corpus of Sanskrit. <http://www.sanskrit-linguistics.org/dcs/index.php>.
- Oliver Hellwig. 2017. Stratifying the Mahābhārata: The textual position of the Bhagavadgītā. *Indo-Iranian Journal*, 60:132–169, January.
- Thomas Koentges and J. R. Schmid. 2018. ThomasK81/ToPan: Rbiter. January. <http://doi.org/10.5281/zenodo.1149062>.
- Thomas Koentges and Jeffrey C. Witt. 2018. ThomasK81/Metallo: HumboldtBonpland. October. <http://dx.doi.org/10.5281/zenodo.1445773>.
- Amba Kulkarni and Devanand Shukl. 2009. Sanskrit morphological analyser: Some issues. *Indian Linguistics*, 70(1–4):169–177.
- Borja Navarro-Colorado. 2018. On poetic topic modeling: Extracting themes and motifs from a corpus of Spanish poetry. *Frontiers in Digital Humanities*, 5.
- Robert K. Nelson. 2011. Of monsters, men — and topic modeling. *The New York Times*, May.
- Andrew Ollett. 2014. Sarit-prasāraṇam: Developing SARIT beyond ‘Search and Retrieval’. Posted on Academia.edu. Slides from a talk given in Oxford (‘Buddhism and Digital Humanities,’ organized by Jan Westerhoff).
- Lisa M. Rhody. 2012. Topic modeling and figurative language. *Journal of Digital Humanities*, 2(1), Winter.

- Jacques Savoy. 2013. Authorship attribution based on a probabilistic topic model. *Information Processing & Management*, 49:341–354, 01.
- Alexandra Schofield, Måns Magnusson, and David Mimno. 2017. Pulling out the stops: Rethinking stopword removal for topic models. pages 432–436, April.
- Yanir Seroussi, Ingrid Zukerman, and Fabian Bohnert. 2014. Authorship attribution with topic models. *Computational Linguist*, 40(2):269–310, June.
- Carson Sievert and Kenneth Shirley. 2014. LDAvis: A method for visualizing and interpreting topics. In *Proceedings of the Workshop on Interactive Language Learning, Visualization, and Interfaces*, pages 63–70, Baltimore, Maryland, USA, June. Association for Computational Linguistics.
- Jian Tang, Zhaoshi Meng, XuanLong Nguyen, Qiaozhu Mei, and Ming Zhang. 2014. Understanding the limiting factors of topic modeling via posterior contraction analysis. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, pages I–190–I–198. JMLR.org.
- Svāmī Yogīndrānanda and Bhāsarvajña. 1968. *Nyāyabhūṣaṇam: śrīmadācāryabhāsarvajñāpraṇītasya nyāyasārasya svopajñāṇ vyākhyānam*. Śaddarśana Prakāśana Pratiṣṭhānam : Prāpti-sthānam Udāsīna Samskr̥ta Vidyālaya, Vārāṇasī.

Vedavaapi: A Platform for Community-sourced Indic Knowledge Processing at Scale

Sai Susarla Damodar Reddy Challa
School of Vedic Sciences Vedavaapi Foundation
MIT-ADT University, Pune Bangalore
sai.susarla@gmail.com

Abstract

Indic heritage knowledge is embedded in millions of manuscripts at various stages of digitization and analysis. Numerous powerful tools and techniques have been developed for linguistic analysis of Samskrit and Indic language texts. However, the key challenge today is employing them together on large document collections and building higher level end-user applications to make Indic knowledge texts intelligible. We believe the chief hurdle is the lack of an end-to-end, secure, decentralized system platform for (i) composing independently developed tools for higher-level tasks, and (ii) employing human experts in the loop to work around the limitations of automated tools to ensure curated content always. Such a platform must define protocols and standards for interoperability and reusability of tools while enabling their autonomous evolution to spur innovation.

This paper describes the architecture of an Internet platform for end-to-end Indic knowledge processing called Vedavaapi that addresses these challenges effectively. At its core, Vedavaapi is a community-sourced, scalable, multi-layered annotated object network. It serves as an overlay on Indic documents stored anywhere online by providing textification, language analysis and discourse analysis as value-added services in a crowd-sourced manner. It offers federated deployment of tools as microservices, powerful decentralized user / team management with access control across multiple organizational boundaries, social-media login and an open architecture with extensible and evolving object schemas. As its first application, we have developed human-assisted text conversion of hand-written manuscripts such as palm leaf etc leveraging several standards-based open-source tools including ones by IIIT Hyderabad, IIT Kanpur and University of Hyderabad.

We demonstrate how our design choices enabled us to rapidly develop useful applications via extensive reuse of state-of-the-art analysis tools. This paper offers an approach to standardization of linguistic analysis output, and lays out guidelines for Indic document metadata design and storage.

1 Introduction

There is growing interest and activity in applying computing technology to unearth the knowledge content of India's heritage literature embedded in Indic languages due to its perceived value to modern society. This has led to several research efforts to produce analysis tools for Indic language content at various levels – text, syntax, semantics and meaning Goyal et al. (2012; Kumar (2012; Huet (2002; Kulkarni (2016; Hellwig (2009). Many of these efforts have so far been addressing algorithmic issues in specific linguistic analysis problems. However, as the tools mature and proliferate, it becomes imperative to make them interoperable for higher order document analytics involving larger document sets with high performance. We categorize existing tools for Indic knowledge processing into three buckets - media-to-text (e.g., OCR (image to text), speech recognition (audio to text)), text-to-concept (e.g., syntax-, semantics- and discourse analysis), and concept-to-insight (e.g., knowledge search, mining, inference and decision-making). For instance, though several alternative linguistic tools exist for Samskrit text

analysis (morphological analysis, grammatical checking), they use custom formats to represent input text and analysis outcome, mainly designed for direct human consumption, and not for further machine-processing. This inhibits the use of those tools to build end-user applications for cross-correlating texts, glossary indices, concept search etc.

On the other hand, the number of Heritage Indic documents yet to be explored is staggering. Data from National Mission for Manuscripts NAMAMI (2012) indicate that there are more than 5 million palm leaf manuscripts that are scanned but not catalogued for content, let alone converted into Unicode text to facilitate search. This is in contrast to less than a million in the rest of the world combined before the advent of print era. In addition, The Internet Archive project Archive.org (2019) has a huge collection of scanned printed Indic books. Very few of them have been converted to text. There are also thousands of online Unicode Samskrit documents yet to be analyzed linguistically for knowledge mining. Use of technology is a must to address this scale.

We believe that to take Indic knowledge exploration to the next level, there needs to be a systematic, end-to-end, interoperability-driven architectural effort to store, exchange, parse, analyze and mine Indic documents at large scale. Due to lack of standardized data representation and machine interfaces for tools, Indic document analysis is unable to leverage numerous advances in data analytics that are already available for English and other languages.

Moreover, Indic documents pose unique challenges for processing compared to other ancient document collections due to the unbroken continuity of Indic knowledge tradition spanning more than two thousand years. First, a vast majority of them are handwritten or in often poorly scanned archaic printed modes in dozens of languages, more than thirty evolving scripts and diverse media. Existing linguistic platforms are inadequate to handle their complexity and diversity. Second, human feedback and correction in a community-sourced mode is essential to curate Indic document content at scale for further machine processing. But the architecture of many existing tools is not amenable to incorporating human input and adapting to it. Finally, Indic knowledge collections and processing tools are fragmented across multiple organizations and administrative boundaries. Hence a centralized approach to user authentication, access control and accounting will not be acceptable.

To overcome these challenges, this paper presents Vedavaapi, a novel platform architecture for community-sourced Indic document processing to transform digitized raw Indic content into machine-interpretable knowledge base. Through Vedavaapi this paper makes the following contributions to facilitate large-scale Indic knowledge processing:

1. A federated RESTful service architecture to support dynamic Indic knowledge processing workflows by leveraging independently evolving services, where each service can be deployed and scaled independently to handle load.
2. A canonical object model to represent document analytics output that enables interoperability between multiple tools in the document processing pipeline and also transparent integration of human feedback at each stage without modifying the tools themselves.
3. A NoSQL-based object store that supports self-describing, versioned schemas to help tool and data evolution over time.
4. A uniform, hierarchical security and access control model for users and object collections that supports decentralization of policies for flexible management across organizational boundaries. This model also allows individual tool providers to meter usage for chargeback to end-users.

The rest of the paper is organized as follows. In Section 3, we define the problem of Indic Knowledge processing, its requirements and the scope of our work. In Section 4, we illustrate the challenges in the use of existing tools for Indic knowledge processing to motivate our work. In

Section 5, we present the principles that guide the design of our solution Vedavaapi. In Section 6, we describe the key architectural aspects of Vedavaapi including its object model, security model and deployment. In Section 7, we present an overview of our current implementation and a qualitative evaluation against our objectives. In Section 8, we outline ideas for future work and conclude.

2 Related Work

Existing work on language and knowledge processing can be viewed from three aspects - Natural Language Processing (NLP) algorithms and tools, human-assisted adaptation techniques around those tools to accelerate curation of content, and end-to-end platforms that compose NLP tools into higher-level workflows. This paper’s focus is on the third aspect namely, how to build a platform that enables composing NLP tools into effective workflows that lower human effort and improve productivity in processing large, diverse document collections. NLP tools exist for each stage of the language processing pipeline shown in Figure 1. Crowd-sourcing is well-known as an effective way to rapidly curate or annotate content and is employed in multiple successful knowledge projects such as Wikipedia Wikipedia (2019). For example, the Bodleian Library at Oxford Libraries (2019) enables crowd-annotation of music collections to describe their content. For Indic document processing, some of the metadata is layered on other metadata and also machine-generated and hence might be inaccurate. Hence it needs manual curation, but should have mechanisms to reduce repetitive corrections. The architecture proposed here enables such flexibility. Workflows to handle archaic document collections are custom-built for individual scripts. In contrast, for Indic document collections we need a system that is geared to handle script diversity as well.

3 Indic Knowledge Processing: Overview and Status

By Indic knowledge, we refer to the practices, techniques and principles that evolved in Ancient India over centuries across all disciplines. Some of that knowledge has been documented in written form via manuscripts, while some got transmitted down to the present via oral, craft and cultural traditions. The objective of Indic Knowledge Processing (IKP) is to recover, preserve, paraphrase and leverage Indic knowledge sources for contemporary applications.

Heritage Indic documents come in all media formats and sizes. They include palm leaf and other manuscripts containing hand-written text preserved over millennia, books printed over the last 2 centuries, audio/video recordings of discourses/renderings by traditional scholars, and thousands of Unicode texts available over the web. Some of these have been digitized, but not yet converted to machine-processable text. They come in dozens of Indic scripts, languages and fonts in multiple combinations NAMAMI (2016), making their organization and processing an engineering challenge. In addition, much of Indic tribal knowledge is still locked up as regional traditions yet to be recorded and captured from their practitioners. Many Indic documents use languages with similar grammatical structure to Samskrit. Samskrit literature is well known to have a rigorous linguistic discipline that makes it more amenable to machine-processing and automated knowledge extraction than other natural languages Goyal et al. (2012). IKP involves creating services to explore Indic knowledge content at various levels – text extraction, syntactic and semantic analysis, knowledge search, mining, representation and inference.

The potential for automated mining of Indic knowledge due to its linguistic base of Samskrit, coupled with the sheer size of Indic document corpus yet to be examined, opens the opportunity to pursue Scalable Indic Knowledge Processing as an impactful research area in computing. This area is inherently multi-disciplinary, and involves rich media analytics (of audio, video, images), machine-learning, computational linguistics, graph databases, knowledge modeling and scale-out cloud architecture.

Figure 1 illustrates the various stages of a typical IKP pipeline covering three distinct transformations: media to text, text to concept, and concept to insight. Each of these stages produces a

high volume of metadata in the form of analysis output, content indexes and user feedback that need to be persisted. Currently, there is a huge corpus of digitized content to feed the pipeline and numerous tools for various stages of the pipeline, but disjointed and not usable in tandem.

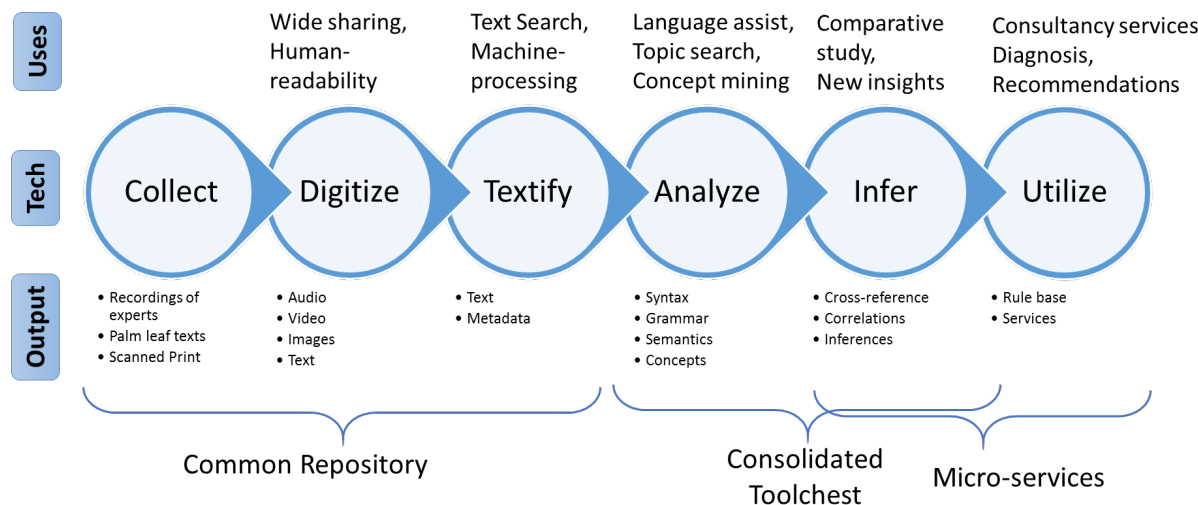


Figure 1: The irregularity of text layouts in Palm leaf manuscripts.

This paper presents the architecture of a novel software platform that bridges the gaps in the IKP pipeline to help rapidly transform digitized Indic knowledge content into useful applications. Some of our target applications include an E-reader for Indic texts that provides search within scanned or audio/video documents, glossary of technical terms used in a book, concept map and knowledge map views and semantic queries. The scope of this paper is restricted to architectural issues and not the algorithmic details of specific stages of the pipeline or the end-user applications.

3.1 Requirements of an IKP Platform

In addition to scalable performance to handle millions of documents by thousands of simultaneous users, an IKP platform must have the following properties:

Durability: It must provide both data and metadata persistence, so users or services can build on prior analysis by others.

Extensibility: The platform must support functional extensions to its services via APIs. It should also provide well-documented data formats and interfaces to incorporate available knowledge sources and analytics tools into its fold. This allows existing analysis tools to be reused in larger contexts than anticipated originally.

Crowd-sourcing: Ambiguity is inherent in natural language understanding. To help resolve ambiguity in analysis and enable users to enrich each other's knowledge through the platform, it must accept human feedback (analogous to Wikipedia) and adapt to it. However to reduce user burden of repetitive corrections, the IKP system must have built-in intelligence to auto-apply suggested corrections to similar contexts.

4 Architectural Considerations for IKP

We now discuss several architectural implications of the above requirements and how existing solutions handle them.

4.1 Handling OCR Errors

First, consider the conversion of digitized content into text, referred to as the “textify” stage in the IKP pipeline of Figure 1. Optical Character Recognition (OCR) technology has matured to extract printed text in many Indic languages from high quality scanned images. Google offers a paid Vision API service Google (2019) that is more than 95% accurate on scanned images of resolution higher than 100 DPI. Open source alternatives also exist (Tesseract Tesseract (2019), Sanskrit OCR by Hellwig Hellwig (2019)), but are not found to be as effective on low-resolution or skewed scans of printed text. The accuracy levels of these services is adequate for direct human consumption for text search purposes, but not for further machine processing. Proof-reading of even a 95% accurate OCR output is a tedious manual effort. Existing OCR services do not have feedback-driven correction in their workflow. Such an adaptation facility would greatly enhance the utility of OCR by reducing repetitive manual work over time.

An IKP system must leverage these OCR tools but also facilitate building human feedback collection and tool re-training workflows around them. Another problem is that the bulk of Indic texts are in handwritten manuscripts with irregular layouts, (see Figure 2 for examples) and existing text segmentation and layout detection schemes are poor at handling them. A more effective alternative for designing an OCR solution would be to separate layout detection and text recognition into modular services and employ the best tools for each service. This enables one to handle printed as well as hand-written text recognition that improves over time, leveraging state-of-the-art tools. In Section 7, we discuss how Vedavaapi achieves that.



Figure 2: The irregularity of text layouts in Palm leaf manuscripts.

4.2 Human-assisted Language Analytics

Machine processing of Indic documents is inherently prone to errors due to ambiguity, For instance, morphological analysis Kulkarni (2016; Huet (2002) of a Samskrit sentence produces alternative semantic trees sometimes running into hundreds. Text segmentation to detect words from a punctuation-free Indic character sequence can also generate multiple alternative segmentations. Such tools still need human intervention both to supply the context to prune the choices during analysis, and to select a meaningful option from analysis output. Further, system adaptation needs to be built in to create self-improving analyzers. All this requires a mechanism to capture human feedback persistently and incorporate it into future analysis tasks. The IKP architecture should provide user-feedback-driven adaptation as a value-addition on top of individual analysis tools, and define standard interfaces to exchange that information with the tools.

4.3 Handling Data Diversity

The input data for an IKP workflow are source documents, which are mostly read-only content. The document analysis tools augment original content with one or more alternate views (e.g.,

morphological analysis of a sentence, a concept map, an OCR output). When a user annotates those views, some of them become irreplaceable and hence must be stored durably. From a mutability standpoint, an IKP system must deal with three types of content with different rates of churn:

Read-only Source Content that is never updated after creation,

Mutable System-inferred Content that can be reproduced by re-running analytics, and

Mutable Human-supplied Content including user annotations and corrections to system-inferred content.

IKP's data store should clearly demarcate these three types and treat them differently to avoid imbalance in storage performance. Also, for the same source content, there could be multiple alternate views at multiple levels of semantics and granularity that need to be tracked as such. For instance, there could be a sentence-level analysis, paragraph-level analysis and global analysis that coexist for a document.

4.4 Implications of Crowd-sourcing

When human input is solicited for correction, there needs to be a facility to track multiple alternate suggestions, rank them by user reputation and provide a consolidated view that represents the most acceptable suggestion. Similarly, the user feedback can be used as training data for machine-learning tools to minimize the need for subsequent corrections. Hence an IKP system must maintain version histories for content updates.

Resolving competing suggestions in a crowd-sourcing situation is a well-understood phenomenon with numerous solutions. The IKP platform must enable the use of such solutions in IKP use cases by facilitating persistent capture of the appropriate data.

5 IKP Architecture: Guiding Principles

Based on the considerations discussed in the previous section, we outline a set of guiding principles for the design of an IKP architecture as follows:

- **Federation:** The architecture must adopt an open platform approach that enables services to be independently developed, deployed and maintained by multiple organizations.
- **Interoperability:** The architecture must allow existing tools to be leveraged in larger Indic document analytics workflows which the tool developers might not have anticipated.
- **Community-sourcing:** The architecture must support overlaying of human input and correction to the output of any of the services transparently.
- **Decentralized security and Accounting:** The architecture must allow single-sign-on across multiple services while allowing them to independently meter resource consumption by end-users for chargeback. For Indic knowledge processing to be accelerated, participation of thousands of scholars and enthusiasts across multiple organizational boundaries is essential. Decentralized authentication and authorization ensures that. Decentralized accounting allows the development of value-add services to enrich the platform in an economically viable manner.

6 Architecture of Vedavaapi

In this section, we describe the architecture of Vedavaapi, a platform we are building to facilitate large-scale IKP workflows. Vedavaapi is a web-based platform that offers rich, multi-layered annotated views of document collections stored natively or elsewhere (such as at archive.org). Figure 3 illustrates the architecture of Vedavaapi. It is organized as a set of loosely coupled web

services and web applications interacting via RESTful APIs. Each such service is packaged as a cluster of Docker containers Docker (2019) for ease of deployment and scaling. A web service only responds to API requests, whereas a web application offers end-user interaction as well, via a GUI.

6.1 The Vedavaapi Ecosystem

One of the core web services is a Vedavaapi site that provides secure controlled access to annotated Indic document collections of an organization. There could be many Vedavaapi sites, and each of them offers an administrative boundary with its own user and document collection management. A Vedavaapi dashboard web application orchestrates end-user interaction with one or more Vedavaapi sites. This application handles single-sign-on user login via social media, user and team management, document collection management and launching IKP workflows via invoking other Vedavaapi web services.

To facilitate third-party IKP tools (e.g., OCR and linguistics tools) to operate on document collections of Vedavaapi sites securely, Vedavaapi provides an adapter library to be bundled with those tools. This adapter provides user authentication and secure access to any Vedavaapi site. A third-party IKP tool can be converted into a Vedavaapi IKP service by wrapping it with a RESTful API frontend along with the adapter library. Using the adapter library, IKP services interact with Vedavaapi sites to retrieve their data and store IKP output on behalf of logged in users.

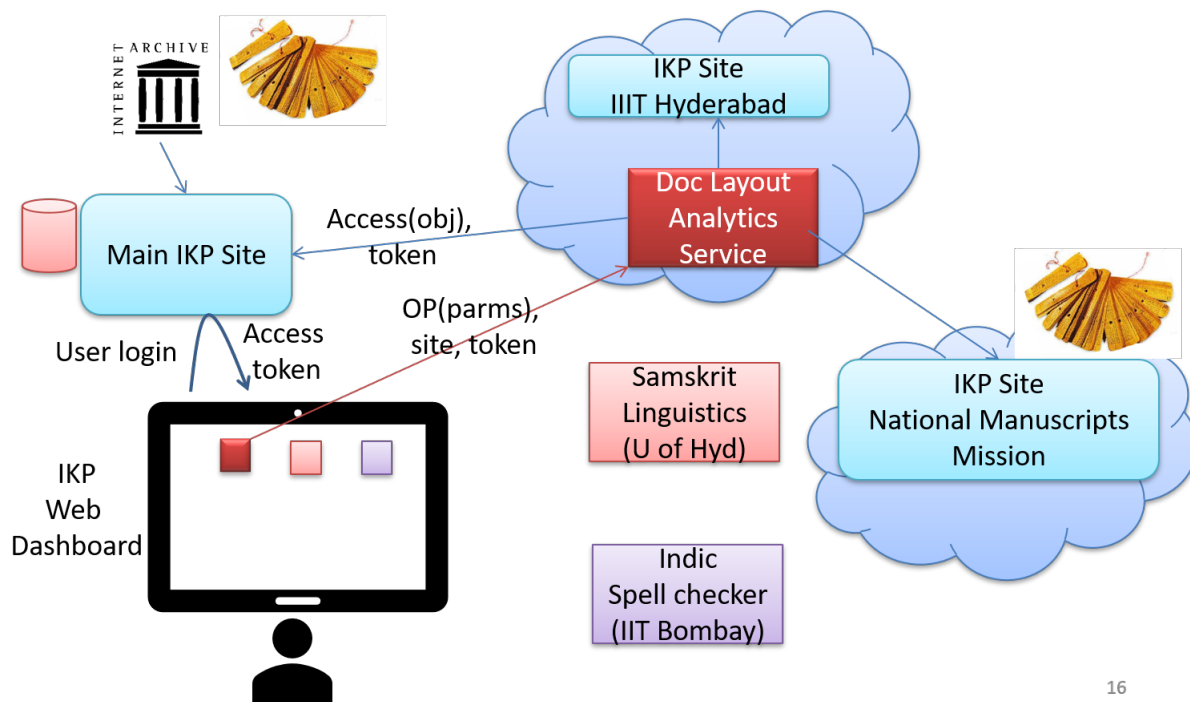


Figure 3: Vedavaapi Federated Architecture. Example IKP services that are active in this illustration are Samskrit Linguistics, Indic Spell Checker and Doc Layout Analytics services.

An IKP service can be registered with multiple Vedavaapi sites to offer its services via specific API endpoints or to manipulate specific document types. When an end-user requests an IKP operation on an Indic document at a site, he/she is presented with a list of registered IKP services available for that operation. For instance, multiple OCR tools can be made available to extract text from a scanned page.

A Vedavaapi site consists of a persistent object store, user and team management service, access control service, and an OAuth service. The object store service houses all of the site's

persistent metadata in a NoSQL database as JSON objects, and provides a powerful navigational query interface. The document source images are stored in the local file system.

6.2 User Authentication

Each Vedavaapi site maintains its own user accounts, teams and access control permissions for its document collection, and exports itself as an OAuth service provider. The Vedavaapi dashboard application authenticates a user via social media login and registers a new user to a site upon first access. Soon after login, it procures an OAuth access token to represent the user for subsequent operations at the site. Unlike cookies, the access token can be passed around to other IKP services to represent the user when accessing Vedavaapi documents.

When a third-party IKP service needs to access or update a document at a Vedavaapi site, it simply passes on the access token it received from its caller (usually the Vedavaapi dashboard application). Thus IKP tool developers are relieved from performing user authentication and access control. IKP services can also invoke other IKP services recursively while representing the same user transparently throughout the delegation chain. Moreover, given the access token, any IKP service provider can retrieve the user profile for accounting / metering the user's operations against his/her quota. This enables the service to chargeback based on usage regardless of where it is the invocation chain.

6.3 Vedavaapi Object Model

A Vedavaapi site stores and manages Vedavaapi objects, which are of three types - agents, resources and annotations. An agent is either a user (either human or bot) who has an account with the site and needs to be authenticated, or a collection of users called a team. Resources are the objects whose access by users needs to be regulated, and which can be annotated by users. Annotations are pieces of information tagged to resources or other annotations, such as the output of an IKP analysis. Every object is referred to by its unique UUID generated by the underlying object store (in our case, MongoDB MongoDB (2016)).

Examples of resources include scanned books, text documents, videos, and collections of other resources such as libraries. IKP applications can define their own resource types. Vedavaapi recognizes a special type of resource called SchemaDef, which describes the schema of any Vedavaapi object using the JSONSchema description language standard Schema (2019). Resources form a strict parent-child hierarchy, whereas an annotation can refer to multiple resources and hence induces a directed acyclic graph. Examples of annotations include transcript, translation, commentary, linguistic analysis output etc.

Vedavaapi object model allows object relationships to be captured via three types of links - source / parent object, target / referred object and members list of a collection object. All objects are referred by their UUIDs issued by the underlying object store (MongoDB in our case). The source / parent link is used to link a resource to its container or parent resource such as books to their library or pages to their book. The target / referred link is used to link an annotation to its referred object such as a transcript to a paragraph. Figure 5 illustrates a network of Vedavaapi objects generated in a typical OCR workflow.

Often, an IKP workflow needs to persist the ordering of objects in a collection, e.g., pages in a book or words in a page. To facilitate that, we define a Sequence resource object as one that enumerates its child resources via their numeric index field. Sometimes, we also need to persist different orderings of the same set of objects, e.g., a user's bookmarked pages in a book. To support that, we define a sequence annotation object as one that explicitly enumerates a set of arbitrary object ids in a specific order via its own "members" field. To capture multiple alternatives produced by an IKP analysis output, we define a choice annotation object as one that returns one of its referring annotations according to a selection strategy such as first, random, vote, etc. For instance, when a morphological analysis produces multiple alternatives, they can be persisted under a choice annotation to be presented to users for voting. Finally, to represent arbitrary semantic linkage among concepts or among sentences in a discourse, we need

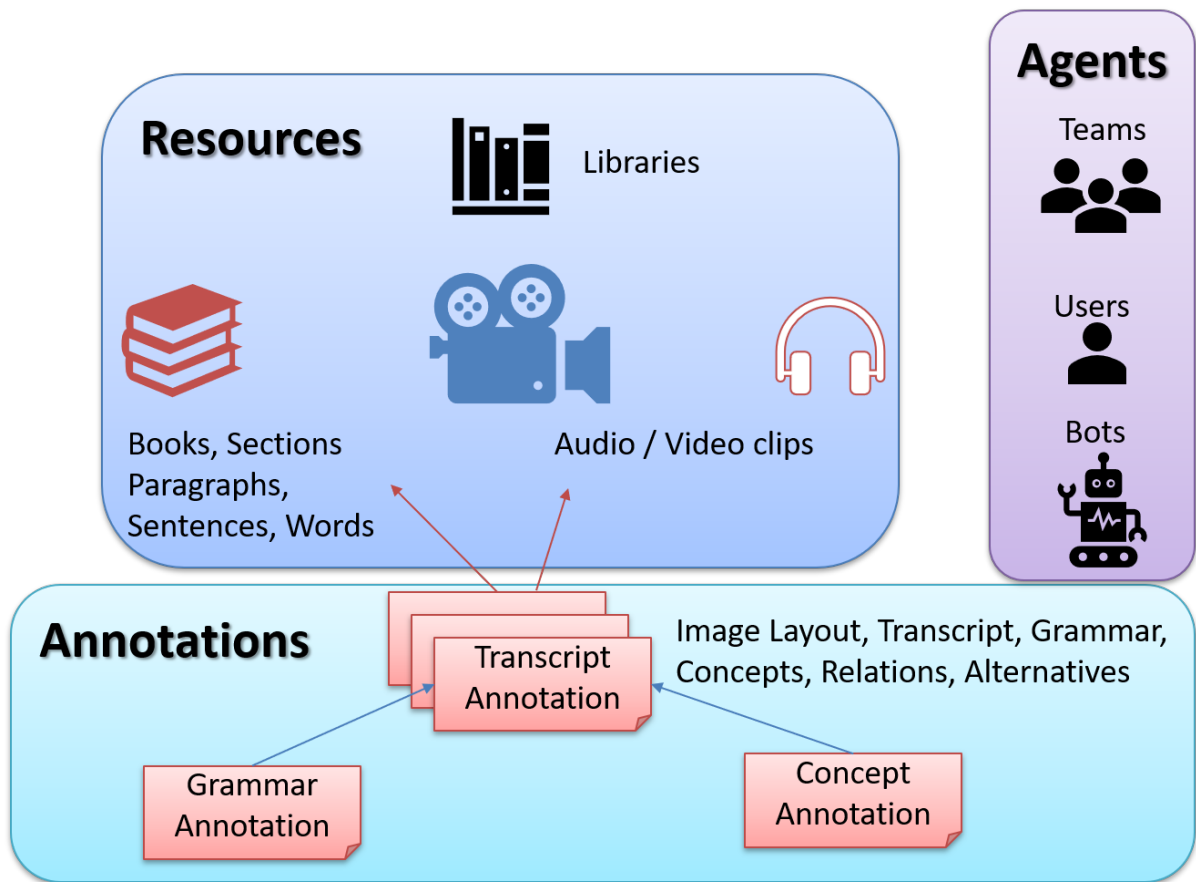


Figure 4: Vedavaapi Object Model.

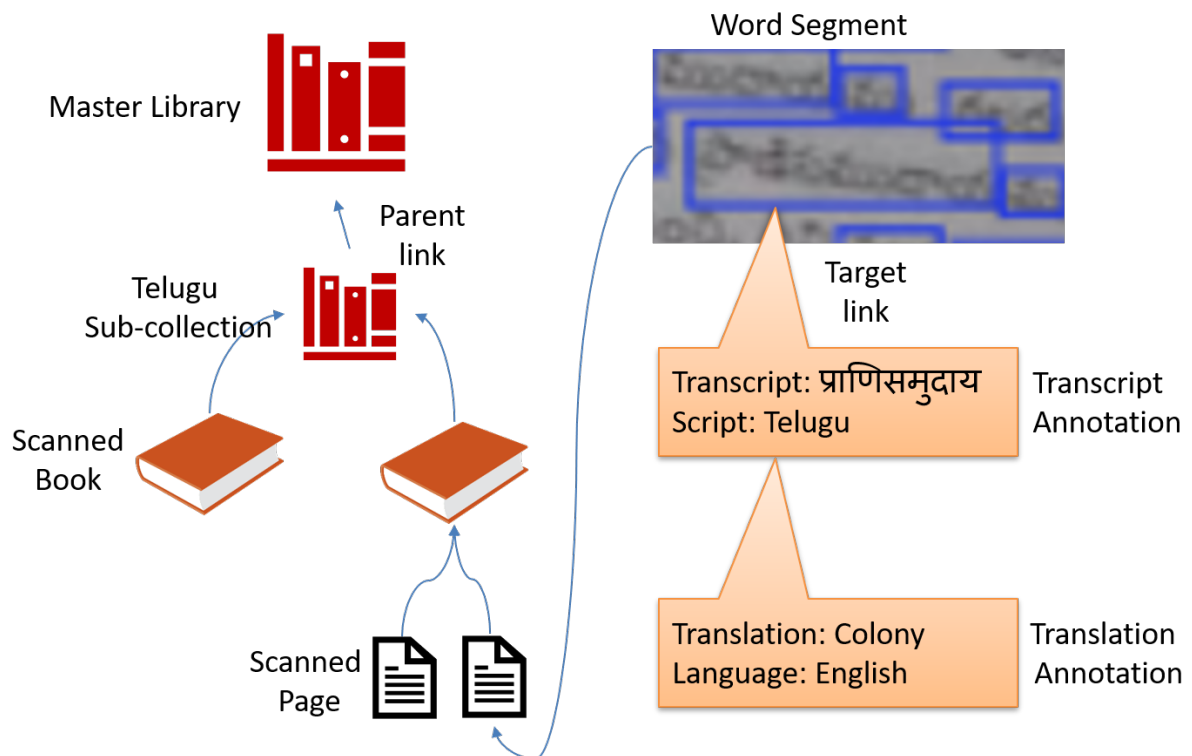


Figure 5: An Example Vedavaapi Object Network showing both resources and annotations.

a generic way to explicitly annotate the relation among groups of objects. We define a relation annotation object as one that captures the semantic relations among two or more resources or annotations.

Figure 6 illustrates the entire class hierarchy of Vedavaapi objects along with their inter-linkage conventions.

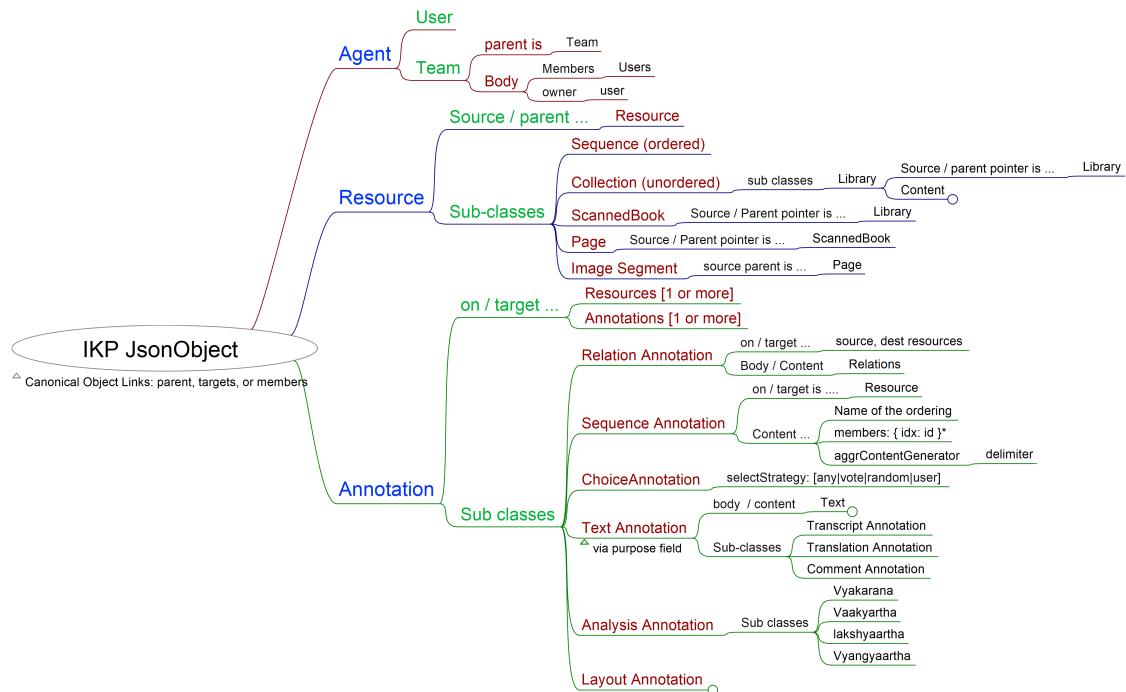


Figure 6: Class Hierarchy of Vedavaapi Objects.

6.4 Vedavaapi Access Control

A large-scale IKP platform must allow different teams the flexibility to manage access to their own document collections independently, while providing administrative override when required. Vedavaapi provides fine-grain control over operations on object content as well as the inter-object network by users and teams. To do so, Vedavaapi recognizes the following operations on objects:

- read: allows reading this object’s content, i.e, metadata attributes
- updateContent: allows updating the object’s attributes
- delete: allows deleting the object and delinking it from its network.
- linkAnnos: allows creating annotations on this object.
- linkChildren: allows linking child resources to this resource.
- updateLinks: allows re-parenting a resource, re-targeting an annotation or changing the members of a collection
- updateAcls: allows updating the access control list of this object.

Vedavaapi uses an ID card approach to authorizing user operations on objects. A user can be part of multiple teams. Hence a user “carries” i.e., inherits the IDs of all the teams to which one belongs.

A Vedavaapi access control list (ACL) is a persistent attribute of the object and applies to all objects - user and teams objects as well as resources and annotations. Moreover, resources inherit the ACLs from their parent resources and annotations inherit ACLs from the objects they target. The ACL comprises three lists for each operation type:

- Granted IDs: the list of user and team IDs that are allowed this operation
- Revoked IDs: the list of user and team IDs that are not allowed this operation.
- Prohibited IDs: the list of user and team IDs that are prohibited this operation

A wildcard “*” in a list matches any ID. Vedavaapi authorizes user operations on objects using ACLs as follows: a user is allowed an operation on an object if at least one of the IDs one possesses is allowed that operation, and none of the IDs is prohibited that operation.

Access control based on ID cards avoids the need to check a user for team membership at access control time, which happens frequently. ACL inheritance offers a convenient and intuitive way for administrators to control access to large object networks. Prohibited IDs feature allows quarantining a user or team in an emergency security breach situation.

As a concrete example, if management of a library and its book collection needs to be delegated to a team, that team can be given update ACLs for the library’s child resource hierarchy while revoking the updateLinks operation to prevent the team from changing how the library connects to the parent document collection.

6.5 Vedavaapi API Overview

Table 1 outlines the APIs exported by Vedavaapi site to client applications. It consists of user and team management, authentication, object store access and ACL management. The APIs mainly support create, read, update and delete (CRUD) operations on various resources. In addition, the uniform object model of Vedavaapi allows a single API to manage diverse object types while also providing a powerful bulk operation interface on object graphs for efficiency. Specifically, the object store offers a versatile graph traversal API that not only is used for retrieving object networks but also upload or modify them. The query for graph traversal takes an attribute-based selection criterion to pick the initial objects and a list of hop criteria to guide the navigation from those objects to others via selected links.

API Cluster	APIs
Accounts	OAuth login and portable access tokens
Accounts	CRUD operations on users and teams
Object Store	CRUD operations on objects (resources, annotations, schemas, services)
Object Store	Object graph traversal, queries, updates and deletes
ACLs	CRUD operations on ACLs for given resource
ACLs	resolve permissions for currently logged in user

Table 1: Vedavaapi API Overview

7 Implementation and Evaluation

We have implemented most of the core Vedavaapi functionality in Python using Flask web services framework to provide RESTful API access. The object store is implemented as a python wrapper around MongoDB. The wrapper provides schema validation, user access control and multi-hop navigational queries on the raw objects stored in MongoDB database. We have implemented the Vedavaapi web dashboard as a standalone AngularJS application that can connect to multiple Vedavaapi sites via their API.

The objective of the Vedavaapi platform is to facilitate leveraging existing tools to rapidly create larger and effective IKP workflows. To evaluate how well our architecture achieves this

objective, we have repackaged several existing open-source and private software modules to create an image-to-text conversion pipeline for scanned Indic documents - both printed and handwritten ones. Unlike existing OCR solutions, our solution enables human intervention to compensate for machine errors as well as OCR retraining for improved effectiveness. To do so, we have ported the following existing tools to run as IKP services in the Vedavaapi ecosystem:

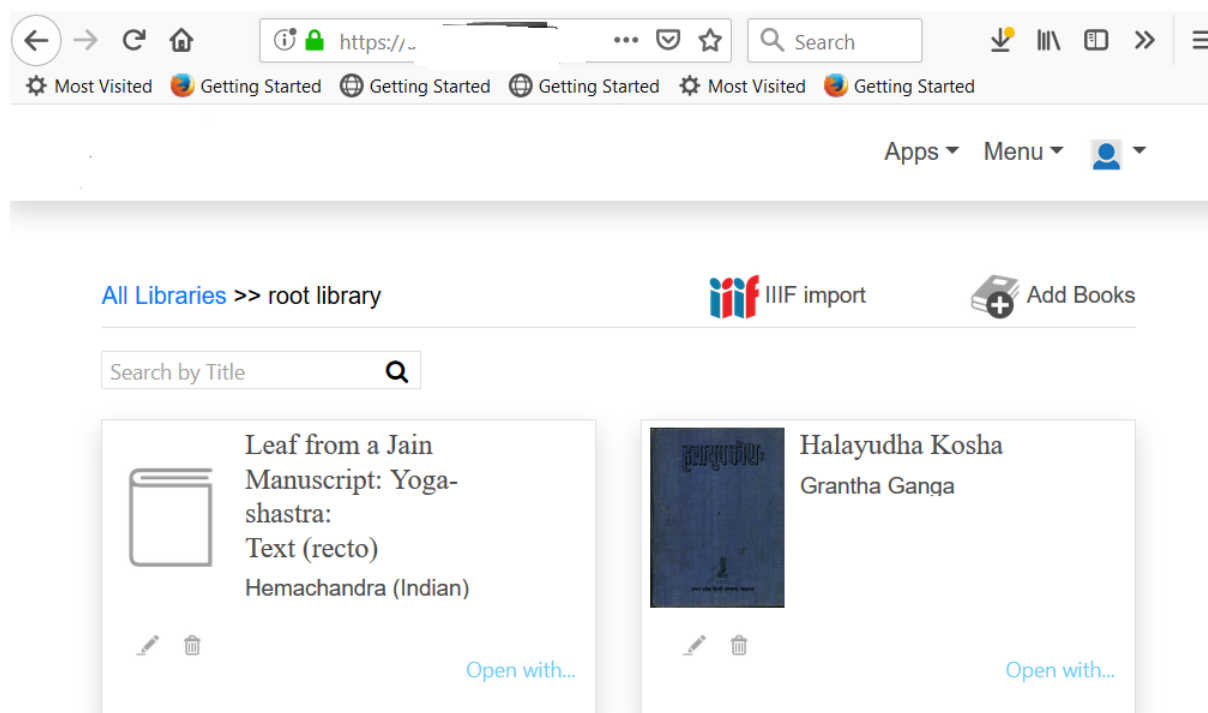


Figure 7: Screenshot of Vedavaapi library view showing books imported from archive.org via IIF importer.

- **IIF Book importer:** This service imports layout and page information of scanned books uploaded to large digitized archives including <https://archive.org/>. We wrote a python library with a Flask API to import an entire scanned book from archive.org from its url as a Vedavaapi resource hierarchy. This way, we can offer IKP services on scanned books stored elsewhere. This took a couple of days of development effort, as Vedavaapi object schema was expressive enough to incorporate their metadata. Figure 7 shows a screenshot of a book imported via this service.
- **Mirador Book Annotator:** Then we ported a sophisticated open-source book viewer and annotator web application (written in JavaScript) called Mirador to operate on Vedavaapi-hosted books. We achieved this by using our Vedavaapi client-side adapter library in JavaScript as a plugin to Mirador to source its book information and serve it from our site. Mirador has a built-in annotation facility that lets users manually identify text segments and also optionally transcript the text. We added persistence by storing those annotation on Vedavaapi backend site. This took one week of effort.
- **Indic OCR Tools:** OCR tools such as Tesseract and Google Vision API service provide both segmentation as well as text recognition from images in an XML-based standard format called hOCR. We created a wrapper service around them to import and export hOCR formatted data as annotations in Vedavaapi. We added a plugin to Mirador to invoke a user-selected OCR service to pre-detect words of a scanned page. This took one

week of effort and greatly helped jumpstart text conversion for many printed texts available publicly.

- hOCR Editor: We ported an open-source web-based text editor for HOCR-formatted output to ease user experience in text conversion compared to Mirador. With our hOCR importer and exporter libraries already in place, this step took a day of effort, mainly to persist edits incrementally on Vedavaapi site. Figure 8 shows a screenshot of a post-OCR editing session. We imported an 800-page book called “Halayudha Kosha” from archive.org using IIIF importer application into Vedavaapi. We then invoked Tesseract OCR on the 10th page. We opened the OCR output using the hOCR editor as shown in the figure.

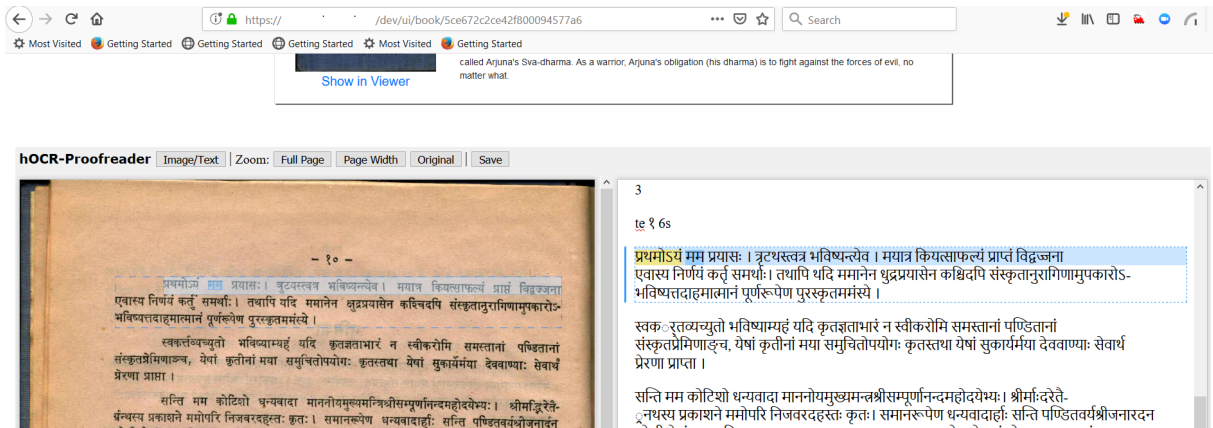


Figure 8: hOCR Editor running within Vedavaapi dashboard for proofreading Tesseract OCR output on a printed page from archive.org. The original image is shown on the left and the word editor is on the right. The yellow is corrected word.

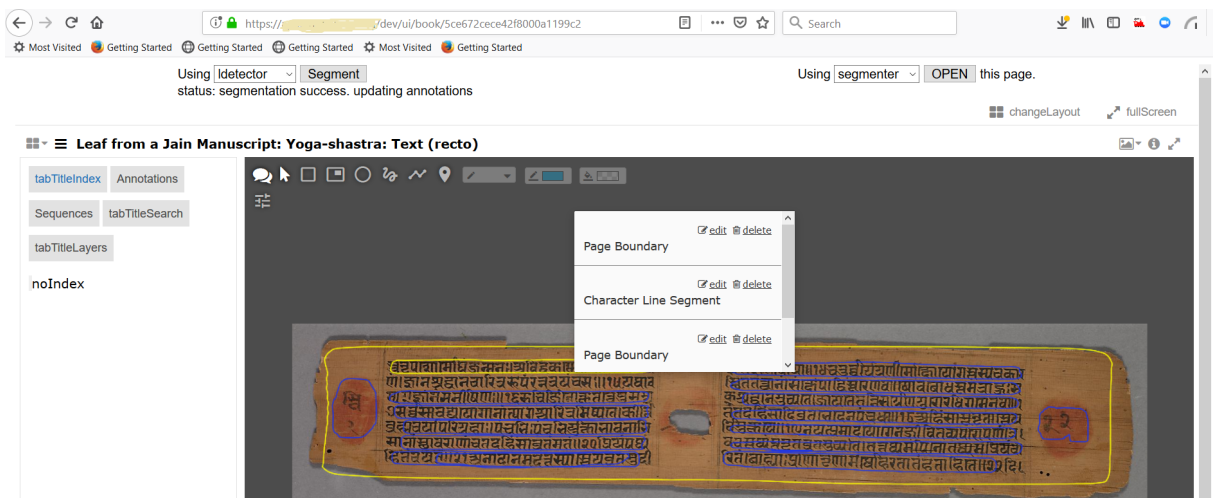


Figure 9: Palm leaf manuscript’s lines detected with IIIT Hyderabad’s palm leaf layout detector and edited through Mirador viewer within Vedavaapi dashboard. The blue contours around the lines were auto-detected and labeled by the tool as “Character Line Segments”.

With these applications integrated with Vedavaapi platform, we got a complete solution for text conversion of archive.org books using OCR tools as well as crowd-sourced human correction working within 2 weeks. However, the layout detection of existing OCR tools on hand-written palm leaf manuscripts is poor due to irregular and overlapping lines in such documents. In parallel, a research group at IIIT Hyderabad developed a deep-learning-based layout detector for

palm leaf manuscripts called Indiscapes Prusty et al. (2019) that automatically draws polygons around lines of text, holes, images and other artifacts by training on manual shape annotations. It requires a machine with GPU for the training step.

- Layout Detector for Palm leaf Manuscripts: Hence we have created a palm leaf layout detector based on IIIT Hyderabad tool. It takes a page image URL from a Vedavaapi site, detects line segments and posts them back as annotations to that page on Vedavaapi with empty text label. The training model file is maintained at IIIT Hyderabad, while the detector runs as Vedavaapi service. Subsequently, we were able to use the hOCR editor to type the text manually, thereby creating a crowd-sourced workflow for online transcription of hand-written text. Porting the tool to Vedavaapi took 2 days of effort as most of the functionality was in place. Figure 9 shows the screenshot of this service running from within Vedavaapi dashboard.
- Samsaadhanii Linguistic Toolkit: We are currently in the process of incorporating Samsaadhanii toolset as an IKP service to be invoked on Vedavaapi-hosted Sanskrit text data. This will test Vedavaapi’s ability to leverage community-sourcing to eliminate ambiguity in linguistic analysis output. This is still a work in progress.

8 Lessons Learnt and Future Directions

Our experience with devising and leveraging the Vedavaapi platform to create IKP workflows indicates that a carefully designed object model that takes the data needs of existing tools can greatly enhance the ability to reuse these tools in providing useful end-to-end IKP solutions. While many of the design choices we had made got validated through the OCR pipeline experiment, we need to work on incorporating the higher order linguistic analysis tools to fully validate the design. During this journey of developing the IKP platform, we realize that there are a lot of popular, well-designed tools already developed and used in different contexts. To really facilitate widespread adoption of such a platform, it should be simple to adapt them to fit into its ecosystem.

Hence the next steps in this effort would be to incorporate tools for text segmentation, Sanskrit linguistics and knowledge mapping to pave the way for a robust, popular platform for innovation around Indic knowledge.

9 Conclusion

In this paper, we made the case for ensuring interoperability of tools and services to accelerate the pace of Indic knowledge processing. While numerous point solutions exist, we have identified that the lack of end-to-end systems approach hinders rapid progress in this field. We present a novel platform approach to IKP architecture that combines the best practices of scale-out cloud computing, careful metadata design and flexible security protocols to significantly accelerate progress in this field.

References

- Archive.org. 2019. Internet Archive: Digital Library of free and borrowable books. <http://www.archive.org/>.
- Docker. 2019. Docker: Enterprise Container Platform. <https://www.docker.com/>.
- Google. 2019. Google Cloud Vision API. <https://cloud.google.com/vision/>.
- Pawan Goyal, Gérard Huet, Amba Kulkarni, Peter Scharf, and Ralph Bunker. 2012. A distributed platform for Sanskrit processing. In 24th International Conference on Computational Linguistics (COLING), Mumbai.

- Oliver Hellwig. 2009. Extracting dependency trees from sanskrit texts. *Sanskrit Computational Linguistics 3*, LNAI 5406, pages 106–115.
- Oliver Hellwig. 2019. Sanskrit OCR. <http://www.sanskritreader.de/>.
- G erard Huet. 2002. The Zen computational linguistics toolkit: Lexicon structures and morphology computations using a modular functional programming language. In *Tutorial, Language Engineering Conference LEC'2002*.
- Amba Kulkarni. 2016. Samsaadhanii: A Sanskrit Computational Toolkit. <http://sanskrit.uohyd.ac.in/>.
- Anil Kumar. 2012. Automatic Sanskrit Compound Processing. Ph.D. thesis, University of Hyderabad.
- Bodleian Libraries. 2019. What’s the score at the Bodleian? <https://www.bodleian.ox.ac.uk/we-ston/our-work/projects/whats-the-score>.
- MongoDB. 2016. MongoDB NoSQL Database. <http://www.mongodb.com/>.
- NAMAMI. 2012. Performance Summary of the National Mission for Manuscripts, New Delhi, India. <http://namami.org/Performance.htm>.
- NAMAMI. 2016. National manuscript mission, new delhi, india. <http://namami.org/>.
- Abhishek Prusty, Sowmya Aitha, Abhishek Trivedi, and Ravi Kiran S. 2019. Indiscapes: Instance segmentation networks for layout parsing of historical indic manuscripts. In *Accepted for publication in ICDAR 2019*.
- JSON Schema. 2019. JSON Schema Standard. <http://json-schema.org/>.
- Tesseract. 2019. Tesseract OCR. <https://opensource.google.com/projects/tesseract>.
- Wikipedia. 2019. Wikipedia: The Free Encyclopedia. <http://www.wikipedia.org/>.

On Sanskrit and Information Retrieval

Michaël Meyer

Paris Diderot university

École pratique des hautes études / Paris

michael.meyer@etu.univ-paris-diderot.fr

Abstract

Many Sanskrit texts are available today in machine-readable form. They are of considerable help to philologists, but their exploitation is made difficult by peculiarities of the language which prevent the use of traditional information retrieval systems. We discuss a few possible solutions to improve this situation and present as well a number of strategies to increase retrieval efficiency.

1 Searching Sanskrit Corpora: Purposes and Difficulties

1.1 The Sanskrit Electronic Corpora

Philologists have nowadays at their disposal many digital resources for the study of Sanskrit literature. Among these resources, electronic texts are of peculiar importance. At the time of this writing, about 1,500 such texts are publically available, all electronic archives included,¹ for a total size of around 350 Megabytes of plain text data.²

The most comprehensive of these collections, both in terms of quantity and in the variety of subjects embraced, is probably the Göttingen Register of Electronic Texts in Indian Languages [GRETIL].³ Of importance is also the digital library of the Muktabodha Indological Research Institute [MIRI],⁴ which focuses on Tantric literature from the Medieval era. The Thesaurus Indogermanischer Text- und Sprachmaterialien [TITUS],⁵ by contrast, mainly focuses on Vedic and Brahmanic literatures.

To our knowledge, the most recent digital library is the Search and Retrieval of Indic Texts [SARIT] repository,⁶ managed by Dominik Wujastyk, Patrick McAllister and a few other scholars. At the time of this writing, it offers access to about sixty Sanskrit texts, all of which are encoded in XML format, according to the guidelines of the Text Encoding Initiative [TEI] standard.⁷ Peter Scharf also provides texts in this format in his Sanskrit Library.⁸ By contrast, most other online repositories typically provide their electronic texts in plain text format, in obscure *ad hoc* formats, or in HTML with very light markup.

Of a different genre are part-of-speech-tagged corpora. We know of only one, the Digital Corpus of Sanskrit [DCS],⁹ elaborated by Oliver Hellwig. However, Huet and Lankri (2018) recently developed a Sanskrit corpus manager that provides access to a number of annotated sentences.

¹It is difficult to give a reliable estimate of the number of *unique* texts input electronically, for two reasons. Firstly, because several scholars have input the same text, sometimes using the same edition, sometimes not, and under various formats. Secondly, because some texts are actually subsets of larger ones, such as the *Bhagavadgītā* relative to the *Mahābhārata*.

²By plain text, we here mean Sanskrit text in the International Alphabet for Sanskrit Transliteration [IAST], encoded in UTF-8 and devoid of markup data such as XML tags.

³<http://gretil.sub.uni-goettingen.de>.

⁴http://muktalib5.org/digital_library.htm.

⁵<http://titus.uni-frankfurt.de>.

⁶<http://sarit.indology.info>.

⁷<https://tei-c.org>.

⁸<https://sanskritlibrary.org>.

⁹<http://www.sanskrit-linguistics.org/dcs>.

1.2 The Importance of Electronic Corpora for Sanskrit Studies

Electronic texts are very useful to philologists. Indeed, philological work in its two forms—edition on the one hand, interpretation and exegesis on the other—requires to discover textual parallels.¹⁰ This is particularly important in the case of Sanskrit literature, because it is rife with citations and glosses. Indeed, scholiasts often cite excerpts from the literature, or paraphrase them, when commenting a text. Identifying the source of these citations can prove difficult, if only because merely vague references to the quoted work are often provided.¹¹

Broadly speaking, we can distinguish two types of philological enquiries.

Firstly, searching for textual parallels, i.e., finding the source of a citation, or, conversely, checking whether a passage from a given text is cited elsewhere. These enquiries usually help to reconstruct corrupt passages or to amend them. They are also useful to obtain a better understanding of the meaning of obscure passages, because the original context of the passage or its exegesis in the scholastic literature generally provide crucial information. Finally, they are of considerable importance to estimate the dates of an author or of a text: checking which texts an author cites, and, conversely, which texts cite him, is one of the most effective ways to estimate his date.

Other inquiries are more linguistic in nature. They typically aim at understanding the meaning of rare syntagms, or the meaning of syntagms that are somewhat common in the literature but possess a technical signification in specific texts. This type of philological work is at the origin of the *Tāntrikābhīdhānaśośa* project, which aims at creating a lexicon of the Tantric terminology. The editors themselves take notice of the importance of electronic texts in their preface to the third volume of the work (Goodall and Rastelli, 2013, 9):

Whereas the initiators of this project worked with notes and card-indices that they had compiled over a life-time of reading, we are faced with dozens, hundreds, or sometimes even thousands of usages of a given tantric expression at the touch of a search-button. Many instances are therefore inevitably unfamiliar to us, but we must at least attempt to take what is relevant into account. Searching through an electronic library with “grep” thus has considerable and obvious advantages, but carries with it an obligation to take into account more passages than we would otherwise encounter. Furthermore “grepping” is especially helpful for revealing the contours of evolutions in usage for certain expressions.

1.3 Limits of Pattern Matching Tools

To assist philologists in their work, the development of full-text retrieval systems is important. A few have been written over the years, usually to provide search interfaces to specific text collections.¹² Despite the existence of these systems, most researchers generally use pattern matching tools such as `grep`, if only because they are more readily available and allow them to search into their private collection of documents or in their research notes.

These tools, however, are not practical for searching Sanskrit texts. Not so much because of speed issues, since pattern matching engines are nowadays highly optimized and since the volume of data is small enough to be fully cached in-memory, even on a low-end computer. But because their matching strategy, as well as their display facilities, are closely tied to the input data format. Searching Sanskrit documents in several formats, not even talking about distinct transliteration schemes, is generally very messy. Many possible query matches are usually missed because of intricacies of the text representation, such as the use of whitespace or the introduction of special symbols and annotations within the text. If the IAST is used, queries can also return more

¹⁰We borrow this distinction from Pollock (2018).

¹¹For instance, a considerable number of citations are introduced with the words *tad uktam* “this has been said,” which tell us absolutely nothing about the origin of the citation.

¹²We present two of them below in section 2.2.

matches than expected when they start or end with a phoneme which textual representation is a substring of another phoneme, as is the case of the simple vowel *i* relative to the diphthong *ai*.

These issues could be alleviated by preprocessing documents and transliterating them to a simplified version of the Sanskrit Library Phonetic Basic encoding scheme [SLP1] (Scharf and Hyman, 2012, 151–158), which possess the useful property that it needs a single code point—in fact, a singly byte—to represent a Sanskrit phoneme. Amending the original documents would however likely cause problems, if, for instance, annotations in English appear within Sanskrit passages; furthermore, the encoding itself, while convenient for machines, is noticeably hard to decipher for a human reader. A better solution would be to write a pattern matching engine that runs its matching algorithm, not on the actual text, but on a logical representation derived from it on-the-fly. To our knowledge, however, this approach is almost never chosen, probably because any non-trivial preprocessing would considerably impede the performance of the engine. Complicated queries are usually relegated to database systems, or, less frequently, information retrieval systems.

1.4 Difficulties in Indexing Sanskrit Texts

It would thus be beneficial to use a real information retrieval system, both for the sake of efficiency and for the sake of flexibility. But the Sanskrit language does not lend itself easily to text retrieval, because indexing a document generally presupposes that it is possible to recognize its lexical units. This process is straightforward in a lot of languages, but is however highly ambiguous in Sanskrit. The difficulties involved are due to two principal reasons: the scarcity of explicit word boundaries, and the existence of euphony phenomena.

In Indic scripts, word boundaries are indeed not necessarily made explicit with whitespace or punctuation characters. Sequences of graphemes thus do not represent words, properly speaking, but rather sequences of one or more words. We call these *clusters*, by analogy with the meaning of the term in musical terminology, where it designates a group of adjacent sounds.

To facilitate reading, researchers usually introduce, while transliterating a text, as much boundaries as possible, typically by adding whitespace characters, as shown in table 1. We call this process *ungluing*. More specifically, we say that a text is *unglued* if no more boundaries can be introduced into it without altering the graphemes that represent phonemes.

Devanāgarī	सोममय इति दर्शडमङ्गीकृतमत्र ।
Transliteration (glued)	<i>somamaya iti darśaṇamaṅgīkṛtamatra</i>
Transliteration (unglued)	<i>somamaya iti darśaṇam aṅgīkṛtam atra</i>

Table 1: Several possible representations of a Sanskrit passage. Excerpt of Vṛṣabhadeva’s commentary on the first chapter of Bhartṛhari’s *Vākyapadīya* (Iyer, 1966, 201, l. 9).

Despite the advantages of this ungluing process, some researchers or copists do not introduce, in transliterated texts, boundaries that were not present in the original text. The rationale for this practice, if any, is unclear to us. A possible explanation could be that preserving the transliterated text in its original glued form eases roundtrip conversions between the original Devanāgarī and its transliterated representation.

A more serious issue resides in the fact that phonemes around word boundaries and at the end of a phrase, before a punctuation mark, can be modified as a result of euphony phenomena (*sandhi*). These transformations obey a set of rules, which can be compactly represented with the notation $\alpha|\beta \rightarrow \gamma$;¹³ the vertical bar here stands for a word boundary, the variable α represents the end of the left word, β the start of the right one, and γ the result of the application of the rule. To be more accurate, we represent here with α and β the shortest possible strings of phonemes that need to be considered for applying the rule.

¹³We borrow this notation from Gérard Huet.

For our purpose, it is convenient to distinguish three basic types of *sandhi* rules. A few of them, all of which have in common that they operate on vowels, produce as output a single phoneme. This is the case of the rule $a|\bar{i} \rightarrow e$, for instance, which dictates that the words *deva* ‘god’ and *īśvara* ‘lord,’ when written in sequence, form the string *maheśvara*. But most other rules produce as output a sequence of two or more phonemes that can be unglued in transliterated texts. We write these rules with the notation $\alpha|\beta \rightarrow \alpha'|\beta'$; the vertical bar on the right side of the arrow indicates that it is possible to unglue the text at this point, while α' and β' denote the transformation of α and β , respectively. The rule $h|c \rightarrow ś|c$, for instance, belongs to this category; it dictates that the words *devah* ‘god’ and *ca* ‘and,’ when written in sequence, form either the string *devāśca* or *devāś ca*. Finally, a few rules do not involve any gluing. They are applied at the end of a phrase, before a punctuation mark. We represent them with the notation $\alpha|\emptyset \rightarrow \alpha'|\emptyset$, where the symbol \emptyset represents the absence of a phoneme.

Despite the difficulties involved in segmenting Sanskrit texts, programs have been developed to address the issue. Gérard Huet (2003; 2005) thus elaborated an unsupervised parser based on finite-state technologies. By contrast, Oliver Hellwig (2009; 2010) developed a supervised parser based on a hidden Markov model, which he trained on manually annotated sentences from the DCS. These tools are of considerable help for computer-assisted linguistic tasks, but it does not seem to us that they are currently robust enough to be used autonomously, without human supervision, for indexing tasks. Gérard Huet’s segmenter—the only one that can be used programmatically at the time of this writing—indeed operates on a finite vocabulary and with a finite set of *sandhi* rules and inflection rules, so that a single unknown word, peculiar form or typing error prevents the segmentation of a full cluster. This is aggravated by the fact that the strings that are worth looking for in an index are typically rare words or syntagms, names of persons, etc., which are the most likely to not be recognized correctly by a tokenizer. Furthermore, many electronic texts contain corrupt passages—either because of typing errors, or because the original manuscript from which the text was copied is itself damaged or corrupt.

Most issues involved in indexing Sanskrit texts would go away if the electronic texts at our disposal were all exhaustively segmented. We do have access, in fact, to segmented electronic texts, most notably the word-reading (*padapāṭha*) of the *Ṛgvedasamhitā* and the annotated texts of the DCS. But they only form a small subset of the available electronic texts, and it is unreasonable to assume that this situation is going to evolve significantly in the near future. For the time being, we should thus be content with the data available, and try to make the best of it.

2 State of the Art

2.1 Basic Structure of an Information Retrieval System

An information retrieval system, at the very least, consists in an index that maps a set of strings to lists of sorted integers that represent the documents these strings occur in. Generally, the offsets at which each string occurs within each document are recorded as well, so as to make possible phrase searches. These lists of occurrences, technically called *postings lists*, are typically represented as arrays of variable-length integers. The index proper is represented as a dictionary-like data structure, a B⁺ tree for instance.

When the documents to index are texts, as is the case for us, an index typically stores the terms that appear in the documents collection, or at least some useful representation of them, such as their stem. But this is in no way mandatory. In particular, a few experimental XML retrieval systems (Büttcher and Clarke, 2005; Strohman et al., 2005) index as well the structure of the document, typically by treating XML tags as if they were terms. This makes possible structured queries with arbitrary nesting of the kind supported by XPath expressions.

Nevertheless, most text retrieval systems available today use a flat data model where structural information is encoded as part of each term, typically by prefixing the term with a binary string that represents the section of the document the term occurs in. This approach is more convenient

to implement and generally reduces the time necessary for evaluating a query. We will soon see that segregating structural information from terms is nonetheless very useful in practice, even for flat text documents that do not have an explicit structure.

2.2 The Existing Sanskrit Information Retrieval Systems

To our knowledge, two Sanskrit text retrieval systems use an information retrieval architecture instead of a pattern matching tool or a traditional database system.

The Gaveṣikā system (Srigowri and Karunakar, 2013) allows searching for the inflected forms of a nominal or verbal stem and supports as well spelling variations. This functionality is implemented by ungluing the text at indexing time¹⁴, and, at search time, by expanding the stem submitted as query to its inflected forms and to alternate spellings of these forms, with the help of a morphological generator. This expansion process does not cover phonetic transformations that result from the application of *sandhi*, so that a number of results are typically missed. Nevertheless, the recall of the system is very high, on par with the DCS word retrieval facilities.

The SARIT corpus also makes use of an information retrieval system, the most interesting feature of which is the support of document attributes. Its indexing strategy is not described anywhere, but we can reasonably assume, by looking at the website documentation and at search results, that the unit of indexing is a cluster. Searching for a string in such a way that all its occurrences are returned thus requires adding wildcards on each side of it, as in **mukha** ‘face,’ for instance. This somewhat defeats the purpose of using an information retrieval architecture, if only because of efficiency reasons. Indeed, searching for a query string with a leading wildcard typically involves a full traversal of the terms dictionary, followed by a costly merge operation.¹⁵ Searching for the string *mukha* in the GRETEL corpus, for instance, would require examining a dictionary of about 2,785,000 clusters and merging about 7,000 postings lists.

We initially wrote our own retrieval system in 2017, as a practical and convenient replacement for traditional string matching tools. It supports searching for arbitrary substrings, while being aware of gluing phenomena and of phonetic transformations that result from the application of *sandhi*. The indexing strategy we chose at the time was elaborated to maximize recall, in such a way that no potential match can possibly be missed, provided that *sandhi* application between the words in the query is deterministic. We were primarily concerned with this completeness guarantee because it is of the utmost importance when searching for textual parallels. However, we did not pay much attention to the precision of the system. Improving it while still maintaining this completeness guarantee indeed creates a host of new difficulties, as will be evident from our discussion below.

3 Adapting Information Retrieval Techniques to Sanskrit

3.1 Substring Search

We explained above in section 1.4 that segmenting a Sanskrit text accurately is a difficult task. For the sake of retrieval, however, it is not necessary to segment texts in a way that is linguistically meaningful. We can make possible arbitrary substring searches, without tokenizing the text in lexical units. This is usually done by indexing the n -grams of a document, that is to say, all substrings of length n this document contains.

In information retrieval, the item n stands for is usually a character, sometimes a word. In our case, n represents Sanskrit phonemes, which map to variable-length sequences of bytes in the source text. Within the index, we represent phonemes as code points in one of the Unicode private-use areas,¹⁶ so that it is possible to index both phonemes and assigned code points together while using the UTF-8 encoding for compressing strings. We currently support as input

¹⁴This detail is not mentioned in the paper, but is patent from the actual implementation: <http://scl.samsaadhanii.in:8080/searchengine>.

¹⁵It is however possible to make wildcards lookup run in sublinear time, for instance by using the technique described in section 3.1.

¹⁶http://www.unicode.org/faq/private_use.html.

transliteration scheme all the variants of the IAST that are actually used in electronic texts.¹⁷ Support for new transliteration schemes could easily be added, by writing a new transliteration state machine or by extending the existing one.

We use trigrams ($n = 3$) as basic retrieval units, as a compromise between speed, usability and simplicity. Searching for strings which include less than three phonemes is not very useful in practice, except for monosyllabic *mantras* (*bījamantra*) which comprehend exactly two phonemes, such as AIM.

Searching with n -grams is conceptually equivalent to matching phrases. For instance, the query *mantra* can be evaluated by retrieving the postings lists of the trigrams *man*, *ant*, *ntr* and *tra*, and by examining them concurrently so as to find a window of length 6 where these n -grams occur, in this order. Given that n -grams overlap when $n > 1$, it is in this case unnecessary to look for all n -grams in the query string to satisfy the query. To retrieve the documents that match the query string *mantra*, for instance, searching for a window of length 6 where the trigrams *man* and *tra* occur in this order is sufficient.

3.2 Handling of Cluster Boundaries

We have so far explained how to support matching strings of phonemes. We should also discuss what to do with cluster boundaries, i.e., the substrings of the indexed text that do not represent phonemes—most notably, whitespace characters. Given that Sanskrit electronic texts are not necessarily unglued, we want cluster boundaries to be treated as optional during matching. For instance, we want the query string *punar api* to match documents that do contain exactly the string *punar api*, but also those that contain the string *punarapi*.

To address this issue, we first used a crude, but somewhat effective, approach: ignore all cluster boundaries in both the indexed text and the query, thus treating *punar api* and *punarapi* as equivalent, for instance. However, this results in false positives when it happens that two or more clusters, when joined together, contain as a substring what could be a valid word in another context, but is not in this peculiar case. For instance, searching for the term *nara* ‘man’ returns documents that contain the string *punar api*, because this string is interpreted internally as *punarapi*, which contains *nara* as a substring. If the text is originally glued, as in the string *punarapi*, it is of course impossible to prevent such false positives without a full-fledged segmentation module. But we can at least prevent them when boundaries were introduced in the text that allow us to do so. For this to be possible, the query string itself should be unglued. In the remainder of this paper, we will assume that this is necessarily the case.

A possible way to prevent this kind of false positives is to include cluster boundaries in the n -grams generated at indexing time, and to amend the user query in such a way that the cluster boundaries it contains need not be present in the text for a document to be considered matching. Representing cluster boundaries inside the n -grams generated at indexing time involves interpreting sequences of characters that do not represent phonemes as a single character, say *_*, and to emit n -grams as usual. The string *punar api*, for instance, thus results in the trigrams *pun*, *una*, *nar*, *ar_*, *r_a*, *_ap* and *api*. Interpreting the user query in such a way that cluster boundaries are optional at search time is, however, more involved. The simplest solution would be to generate all possible forms the query string could take on inside a document, and search for the union of these strings. Following this process, the query string *punar api ca*, for instance, would be expanded to the union of the strings *punar_api_ca*, *punar_apica*, *punarapi_ca* and *punarapica*, all of which would then be segmented into n -grams for evaluating the query.

This method, however, would produce a highly redundant query. A convenient way to improve it becomes readily apparent if we observe that we ultimately want to produce a query that is semantically equivalent to a regular expression where cluster boundaries are optional, such as *punar_?api_?ca*. In other words, we want to produce a query that recognizes the language of

¹⁷A few phonemes can be written in different ways, for instance the *anusvāra*, which is represented as *ni* or *ṁ* depending on the source text. We systematically ignore Vedic accents.

the finite-state automaton denoted by such a regular expression. Instead of selecting n -grams from the query string proper, we can thus convert it to a finite-state automaton, amend this automaton to make cluster boundaries optional, determine and minimize it so as to obtain an automaton similar to the one presented in figure 1, and finally extract n -grams directly from this representation.

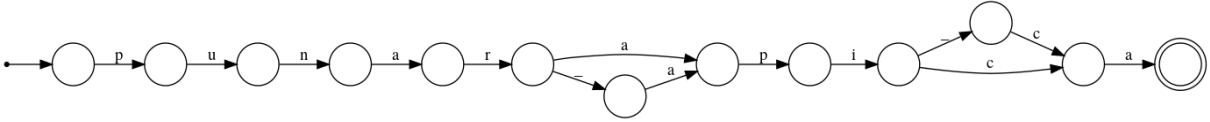


Figure 1: Minimal acyclic finite-state automaton denoted by the regular expression $punar_?api_?ca$

There is however a simpler solution to the problem of cluster boundaries. Instead of representing cluster boundaries inside the n -grams generated at indexing time, we can instead keep indexing texts as if they did not contain any cluster boundaries, and index separately cluster locations. To do that, it is necessary to amend the n -grams tokenizer so as to make it emit a special token C each time a new cluster is encountered, with the same position as the first n -gram in the cluster. The resulting postings list C thus records the start offset of all clusters in the document collection. Query strings must be processed in a similar way, so as to obtain a list of n -grams to look for, on the one hand, and a list of cluster start offsets, on the other. The evaluation of the query follows the process we described above in section 3.1, up to the point where an interval $[a, b]$ of the document that matches the query n -grams is delimited. At this point, an additional test is required to determine whether the segmentation of the text is compatible with the one of the query: if all cluster boundaries c that occur within the delimited interval of the document such that $a < c < b$ also appear at the same relative position in the query, the delimited passage can be considered a match; otherwise, it must be discarded.

Compared to the first solution, this approach presents the disadvantage that it requires additional storage. But it can also be much faster to execute, provided that the index layout is modified in the way described below in section 4.3.

3.3 Handling of *Sandhi*

The most vexing difficulty to take care of is however the handling of *sandhi*. Briefly put, we want a query string to match, not only its original form, but also all the forms it could take on inside a text as a result of the application of *sandhi*. For instance, we want the query *devaḥ* to match, not only the string *devaḥ*, but also the strings *devaś* in *devaśca* ‘and the god,’ *devo* in *devo’pi* ‘but the god,’ and so on.

3.3.1 Noncontextual *Sandhi* Expansion

In our first attempt at the task, we used a simple generation approach that only takes into account a single side of each *sandhi* transformation rule, treating the other as if it did not matter for the application of the rule. To be more explicit, we treated rules of the type $\alpha|\beta \rightarrow \gamma$ as if they could be read as $\alpha|^* \rightarrow \gamma$ or $*|\beta \rightarrow \gamma$, where the wildcard symbol $*$ stands for an arbitrary sequence of zero or more phonemes; similarly, we treated rules of the type $\alpha|\beta \rightarrow \alpha'|\beta'$ as if they could be read as $\alpha|^* \rightarrow \alpha'|^*$ or $*|\beta \rightarrow *|\beta'$; and we treated rules of the form $\alpha|\emptyset \rightarrow \alpha'|\emptyset$ as $\alpha|^* \rightarrow \alpha'|^*$.

To implement this approach, we wrote a *sandhi* application module in the most straightforward way,¹⁸ and systematically exercised it so as to generate two lookup tables T_{left} and T_{right} . T_{left} maps a given phoneme to the set of forms it could take on as a result of *sandhi* application when it occurs at the beginning of a word. Conversely, T_{right} maps sequences of one or two phonemes

¹⁸At the time we started developing our system, Gérard Huet’s *sandhi* engine was not yet publicly downloadable.

to the forms they could take on when they occur at the end of a word. A record of each table is reproduced in table 2, together with sample rules from which each record entry was derived.

T_{left}		T_{right}	
Entry	Sample rule	Entry	Sample rule
\bar{u}	$u u \rightarrow \bar{u}$	v	$u a \rightarrow v a$
o	$a u \rightarrow o$	uv	$u a \rightarrow uv a$
u	$k u \rightarrow k u$	\bar{u}	$u u \rightarrow \bar{u}$
		u	$u k \rightarrow u k$

Table 2: *Sandhi* expansion of the phoneme u in T_{left} and T_{right}

At query time, we look up the beginning and the end of the query string submitted by the user in the tables T_{left} and T_{right} , respectively, and use the retrieved data to construct a query that matches all possible forms the original query string could take on inside a text. This generation process is performed by creating, from the user query and the data retrieved in T_{left} and T_{right} , a minimal acyclic finite-state automaton similar to the one presented in figure 2, before extracting n -grams from this representation, in the manner described above in section 3.2.

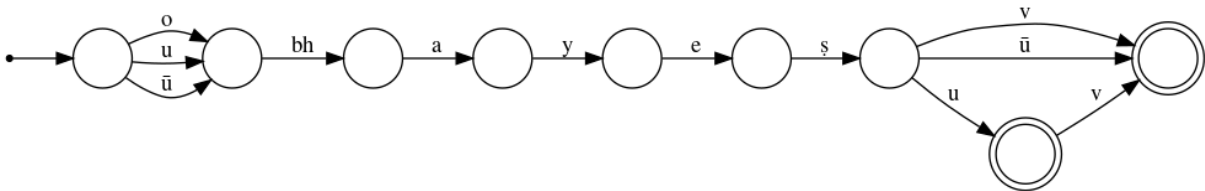


Figure 2: The query string *ubhayesu* after *sandhi* expansion

This *sandhi* expansion process does not require additional storage and is fast enough to be performed online, because the number of forms α and β can take on after *sandhi* application is small in practice. It is, however, incomplete, since it does not attempt to apply *sandhi* between the words of a query string, essentially ignoring the fact that *sandhi* application is a nondeterministic process and could thus produce several distinct query strings. Furthermore, it also returns many false positives, since it does not take into account the phonetic context of the query string within a document.

3.3.2 Contextual *Sandhi* Expansion

The errors generated by our *sandhi* expansion technique fall into three basic categories:

1. To begin with, *sandhi* expansion is performed even at the very beginning of a phrase, where no *sandhi* can possibly occur. The query string *iti* ‘thus’ thus ends up erroneously matching the verb *eti* ‘he goes’ when this verb appears at the beginning of a phrase, because of rules such as $a|i \rightarrow e$.
2. Similarly, false positives can occur at the end of a phrase, before a punctuation mark. In this configuration, the only rules that should be taken into account are those of the form $\alpha|\emptyset \rightarrow \alpha'|\emptyset$.
3. But the vast majority of false positives occur within clusters. For instance, the query string *devah* ends up incorrectly matching the string *devasabda* ‘divine sound,’ because *devah* is expanded to *devas* due to the rules $h|c \rightarrow s|c$ and $h|ch \rightarrow s|ch$, and because *devas* is a substring of *devasabda*.

False positives of the types 1 and 2 could be addressed to by implementing a filtering mechanism similar to the one we described above for cluster boundaries. To make this possible, the start and end positions S and E of each *daṇḍas*-delimited text segment should be recorded in the index, and n -grams in the query tree should be annotated to reflect the conditions under which they can be considered to match. For instance, the trigram *eti* generated from the query string *iti* should be annotated with a flag that dictates that it can only be considered to be a match if no postings in S possess the same position. Similarly, the trigram *evo* generated from the query string *devas* and the rule $as|a \rightarrow o|'$ should be annotated with a flag that dictates that it can only be considered to be a match if E does not occur three positions ahead of it.

This solution is feasible for false positives of the types 1 and 2, but not so much for those of the type 3. The main problem is that performing contextual checks becomes in this case prohibitively expensive. To test whether the transformation of a k to a g at the end of a word is appropriate in a given context, for instance, we would have to check the postings list of about 27 phonemes. The cost of query evaluation could be improved by indexing phoneme classes so as to reduce the number of postings list that need to be examined concurrently. If, say, all sonants except nasals were indexed under a single postings list S , we could determine whether the transformation of a k to a g at the end of a word is appropriate by examining just S .

However, it seems to us preferable to take the reverse approach, that is to say, to resolve possible results of *sandhi* application at indexing time. To test this approach, we wrote a transducer that maps each possible string that could result from the application of *sandhi* to the set of rules that could have generated it. Instead of attempting to determine whether a given reading is correct, as would a full-fledged tokenizer, we assume it necessarily is, and index it as such. To be more specific, we generate two extra tokens α_{right} and β_{left} that represent respectively the values α and β of *sandhi* rules of the form $\alpha|\beta \rightarrow \gamma$ or $\alpha|\beta \rightarrow \alpha'|\beta'$, each time such a rule is recognized in the source text. These extra tokens are affected the same position as the string γ , α' or β' they stand for. For instance, the phoneme \bar{a} in the word *uvāca* triggers the generation of the tokens a_{left} , a_{right} , \bar{a}_{left} and \bar{a}_{right} , with duplicates removed. Similarly, we emit one extra token α_{right} each time a *sandhi* rule of the form $\alpha|\emptyset \rightarrow \alpha'|\emptyset$ is recognized in the source text.

With this approach, the query process is greatly simplified. We start by looking up the first phoneme of the query string in the index so as to retrieve its postings list of the form β_{left} , if any. The same operation is performed for the last one or two phonemes of the query string, so as to retrieve the corresponding postings list of the form α_{right} . The remainder of the string is then segmented into trigrams as usual, and a phrase query is finally constructed from these three types of elements. For instance, the query string *ubhayatas* results into the four tokens u_{left} , *bhay*, *yat* and as_{right} , which are then combined to construct a phrase query.

This approach is appropriate for a small number of documents, but might be less feasible for a large documents collection. Indexing *sandhi* rules indeed considerably increases the size of the index and produces huge postings lists, an effect that is compounded by the fact that, to make possible searching for strings which comprehend between 3 and 6 phonemes included, bigrams ($n = 2$) and unigrams ($n = 1$) must also be indexed. To support a real workload, it might be necessary to prune bigrams and unigrams that are useless for matching the text; if, for instance, a string of three phonemes that cannot possibly involve phonetic modifications—say *abhi*—appears at the beginning of a verse, indexing its bigrams and unigrams is unnecessary, because its trigram will necessarily be selected over them at search time.

3.4 Searching for Inflected Forms Given a Stem

When searching for a peculiar syntagm, as opposed to a phrase, it is often desirable to obtain, in the set of search results, documents that contain various forms of this syntagm, such as its plural form, instead of just the specific form that was submitted in the query. This functionality is important for the Sanskrit language, because its morphology is particularly rich.

To make possible this type of functionality, it is customary to use a stemmer, that is to say,

a program that maps an inflected form to its stem, or at least to a string that is not a valid stem but that could stand for it in some way.¹⁹ This approach is of course only feasible if lexical units can be distinguished in the first place. It is thus not suited to the indexing framework we described above.

However, it is possible to use the reverse approach, that is to say, to generate, at query time, all possible inflected forms of a stem submitted as query, and to search for the union of the resulting strings. This is the approach used by Srighowri and Karunakar (2013). The same strategy can be used with our indexing scheme, save for the fact that each generated inflected form must itself be segmented into n -grams. To generate a compact query tree, we can construct a minimal acyclic finite-state automaton such as the one presented in figure 3 and extract n -grams from this representation, as done above in section 3.3.1.

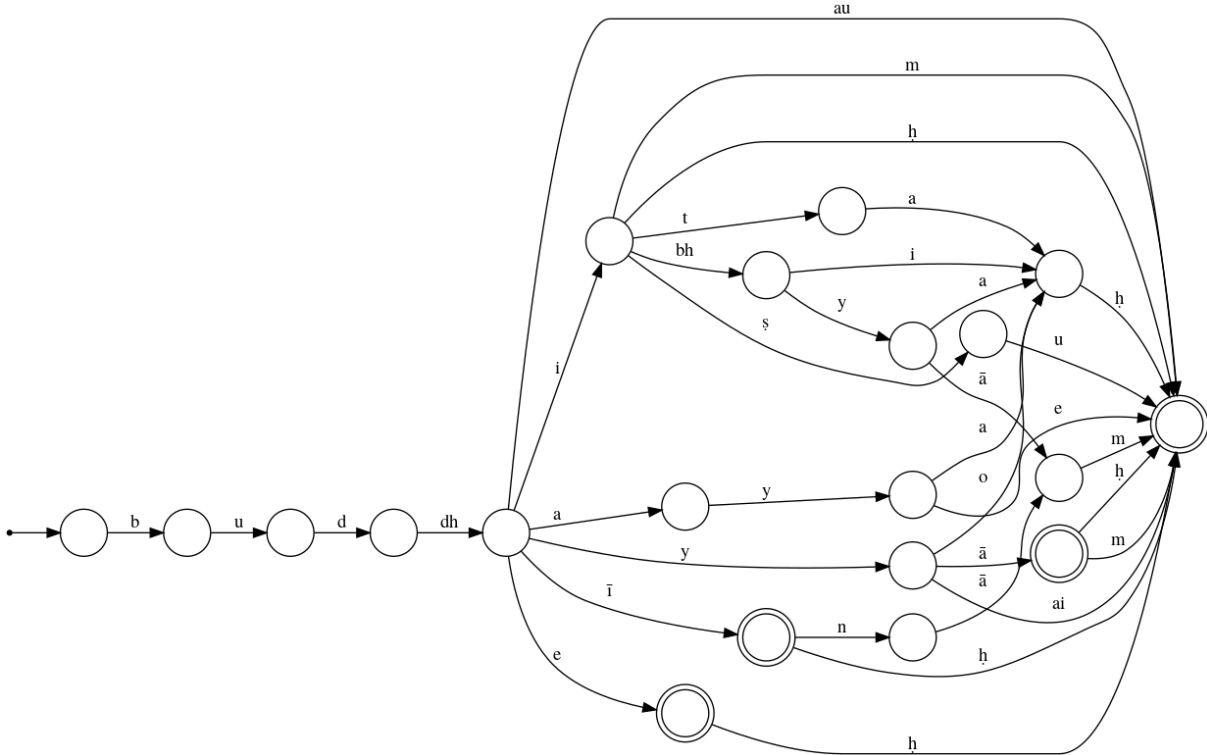


Figure 3: Inflected forms of the nominal stem *buddhi* (fem.) ‘intelligence’ represented as a finite-state automaton

4 Increasing the Efficiency of the System

4.1 Combining Documents Representations

We assumed so far that existing tokenizers for the Sanskrit language are not yet robust enough to be used autonomously for indexing tasks. This does not imply, however, that they should not be used at all. Relying exclusively on the output of a real tokenizer would not suffice, for the reasons given above in section 1.4. But we could still index both the output of such a tokenizer and the n -grams generated as described above.

This strategy has been used with some success in Chinese information retrieval (Zhang et al., 2000), and would thus probably benefit us too. However, the very process of indexing several readings of the same text is in itself technically challenging, because *sandhi* application often modifies the length of a word, which complicates phrase matching. We have not yet elaborated a robust enough strategy to address this issue. Furthermore, it is not yet clear to us how exactly

¹⁹Algorithmic stemmers such as the one of Porter (1980) do not necessarily return valid stems.

the output of a full-fledged tokenizer could be used to produce a better ranking of possible query matches. At the very least, we could provide to the user a visual cue of the estimated correctness of a match by highlighting possible matches with different colors, or maybe different shades of the same color.

4.2 Optimizing the Selection of N -grams

In all the information retrieval systems we have studied so far, no special attention is given to the fact that query strings which contain a number of characters that is not a multiple of n when $n > 1$ can be segmented in different ways while minimizing the number of n -grams in the query. Furthermore, it is implicitly assumed that pruning as many n -grams as possible in the query string is necessarily beneficial, as far as retrieval time is concerned.

This assumption, however, is not necessarily correct. For the time of retrieval is by far dominated by the decoding of postings lists, not by the lookup of n -grams in the index. A more accurate estimation of the cost of the evaluation of a query can thus be obtained by examining the frequency of each query n -gram in the collection, an information that is usually readily available in the index data structure. Instead of arbitrarily pruning n -grams, we should thus attempt to select the combination of n -grams that minimizes the overall number of postings to be decoded while satisfying the query. This amounts to interpreting the user query as if it was a directed acyclic graph, where vertices represent n -grams and edges represent the length of the postings list of the target n -gram, so as to find the shortest path from the first n -gram to the last one.

An example is given in figure 4, for the query string *mantraḥ* and $n = 3$. Edges are annotated with the actual frequency of the n -gram they point to in the GRETIL corpus. The cost of decoding the postings list of the first trigram *man* can be ignored, just as the cost of decoding the postings list of the last trigram *rah*, since both trigrams must necessarily be selected for the query to match. In this case, the optimal path is $0:man \rightarrow 2:ntr \rightarrow 4:rah$, which involves decoding $203,356 + 37,885 + 76,636 = 317,877$ postings. By contrast, choosing the path $0:man \rightarrow 3:tra \rightarrow 4:rah$, which also involves the minimum number of trigrams necessary to match the query, would lead to decoding nearly two times more postings.

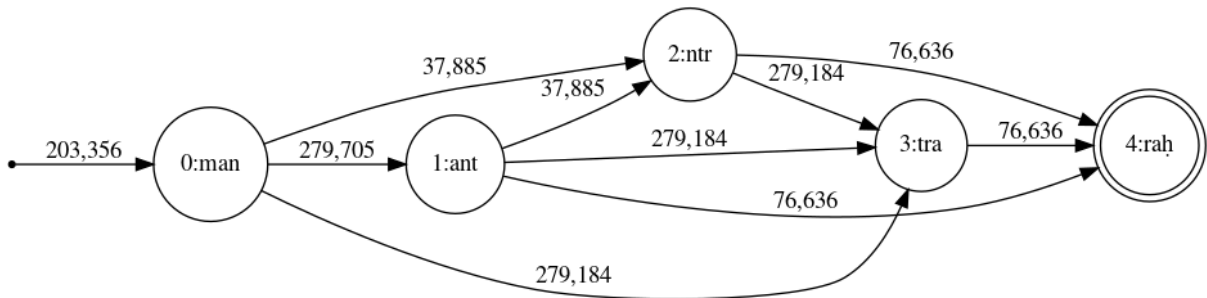


Figure 4: The query string *mantraḥ* represented as a directed acyclic graph of trigrams

A further optimization is also possible, this time for all $n > 0$, when a query string contains several occurrences of a given n -gram. In this situation, query tree nodes can be shared, so that the postings list of a n -gram that occurs several times in the query need to be decoded only once. Accordingly, the contribution of a given n -gram to the cost of query evaluation should be taken into account only once during the n -grams selection process.

4.3 Optimizing the Representation of Postings Lists

The indexing strategies we discussed so far are very costly in terms of processing time if a traditional index layout is used for representing postings lists. Indeed, lists of documents identifiers are usually separated, both conceptually and physically—on disk or in-memory—from the lists of

integers that represent the positions of a given term within a given document. The primary motivation for this data layout is the idea that the queries that do not involve positional matching should not incur the cost of decoding lists of positions. In our case, however, positional queries are almost always necessary. We would thus benefit from inlining lists of positions within lists of documents.

It seems however beneficial, in terms of implementation complexity and in terms of expressiveness at the very least, to go one step further and merely index positions, essentially treating the documents collection as if it was a single long document. Doing this allows better granularity at retrieval time. If documents boundaries are indexed in the way we proposed to index clusters previously in section 3.2, it indeed becomes possible to constrain matching, not merely to a single document, but also to a sequence of documents. Other structural information could be indexed as well, such as verse boundaries or paragraph boundaries, to make possible more expressive queries.

But the main advantage of this index layout lies in the fact that it lends itself more conveniently to the optimization of positional queries. For these queries can often be evaluated without reading in full the postings lists of the terms they are made of. To give but one example, if a document contains the string *a a a a a a b* and we are looking for the phrase *a b*, all postings of *a* up to the last one are irrelevant, and thus we can jump directly to the last occurrence of *a*. For this to be possible, postings lists must be made addressable, at least to some degree.

To address this problem, Moffat and Zobel (1996) propose to split each postings list into several chunks, and to prefix each of these chunks but the last one with the identifier of the first document in the next chunk, together with a pointer to this chunk. This solution can of course be extended to positions lists. It saves processing time, since chunks that are irrelevant for the evaluation of a query need not be decoded, and can be skipped over. However, it does not save much disk or memory bandwidth, if at all, because a chunk, or at least its initial part, must necessarily be fetched from disk or from memory in order to retrieve the location of the next one. This suggests that we should store chunks pointers separately from the chunks themselves.

To do that in a way that is amenable to disk storage, we propose to store postings lists contiguously on disk, while introducing a logical separation in fixed-size pages. A single postings list can thus cross several pages, and, conversely, a single page can hold several postings lists. The postings lists that cross several pages can then be indexed by allocating new pages and store there pointers to each page the list occupies at the lower level, together with a copy of the first position of each delimited chunk of the list at the lower level. This process can be repeated again to introduce further levels of indexing. Briefly put, this amounts to constructing a kind of skip list (Pugh, 1990) that is laid out similarly to a B⁺ tree, over the whole set of postings lists in the index.

An example of this data layout is given in figure 5. It describes a two-levels index that contains six pages, on which are laid out the postings lists of three distinct terms *x*, *y* and *z*. Positions that belong to the same postings list are represented with the same color.

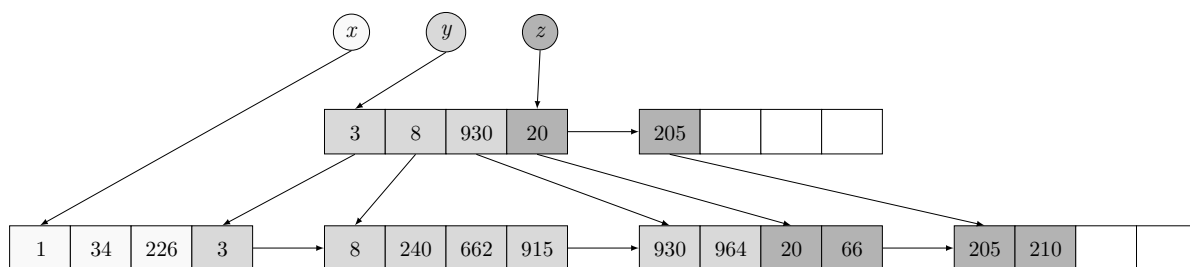


Figure 5: Representation of postings lists

Conclusion

We have discussed in the above several possible ways to model, in an information retrieval system, a few peculiarities of Sanskrit phonetics and morphology, and described as well a number of possible optimizations to reduce processing time. This however only scratches the surface of the functionalities an information retrieval system is expected to provide.

The most pressing goal, for the time being, is to elaborate an architecture that strikes a good balance between the system’s precision, its recall, and its efficiency in terms of time and space. In particular, the interaction of the strategies we described above deserves special consideration, because their compounding effect can easily lead to excessively complicated queries. It might be necessary to adopt several distinct retrieval strategies depending on the query and the user’s expectations. To help alleviate the issue, it is desirable to give more control to the user over the query process, so that he can choose whether a quick but possibly incomplete or inaccurate answer is preferable to a more accurate, but slower one. Accordingly, it is necessary to elaborate an evaluation methodology to test the time and space efficiency of the system.

Much also remains to be improved in the area of query expressivity. We did not discuss how to support, within our framework, Boolean operators, proximity operators and containment operators—searching within a given number of documents, of paragraphs or of lines, for instance. It is also desirable to formalize a query syntax that gives full control to the user over the search process; most notably, over its linguistic features: the handling of *sandhi* and the expansion of stems to their inflected forms.

Acknowledgements

I am grateful to the anonymous reviewers for their helpful corrections and suggestions.

References

- Stefan Büttcher and Charles L.A. Clarke. 2005. Indexing time vs. query time trade-offs in dynamic information retrieval systems. In *University of Waterloo Technical Report CS-2005-31*, Waterloo, Canada.
- Dominic Goodall and Marion Rastelli. 2013. *Tāntrikābhidhānakośa III: Dictionnaire des termes techniques de la littérature hindoue tantrique / A Dictionary of Technical Terms from Hindu Tantric Literature / Wörterbuch zur Terminologie hinduistischer Tantrén*. Sitzungsberichte, Österreichische Akademie der Wissenschaften, Philosophisch-Historische Klasse; Beiträge zur Kultur- und Geistesgeschichte Asiens. Verlag der Österreichische Akademie der Wissenschaften, Wien.
- Oliver Hellwig. 2009. *SanskritTagger*. In Gérard Huet, Amba Kulkarni, and Peter Scharf, editors, *Sanskrit Computational Linguistics: First and Second International Symposia* (Institut national de recherche en informatique et en automatique, Rocquencourt, France, Oct. 29–31, 2007 and Brown University, Providence, RI, USA, May 15–17, 2008), number 5402 in Lecture Notes in Artificial Intelligence, pages 266–277, Berlin and Heidelberg. Springer.
- Oliver Hellwig. 2010. Performance of a lexical and POS tagger for Sanskrit. In Girish Nath Jha, editor, *Sanskrit Computational Linguistics: 4th Symposium* (Jawaharlal Nehru University, New Delhi, India, Dec. 10–12, 2010), number 6465 in Lecture Notes in Artificial Intelligence, pages 162–172, Berlin and Heidelberg. Springer.
- Gérard Huet and Idir Lankri. 2018. Preliminary design of a Sanskrit corpus manager. In Gérard Huet and Amba Kulkarni, editors, *Computational Sanskrit & Digital Humanities: Selected Papers Presented at the 17th World Sanskrit Conference* (University of British Columbia, Vancouver, July 9–13, 2018), pages 259–276, New Delhi. D K Publishers Distributors Pvt. Ltd.
- Gérard Huet. 2003. Lexicon-directed segmentation and tagging of Sanskrit. XIIth World Sanskrit Conference, Helsinki, Finland, Aug. 2003. URL: <http://gallium.inria.fr/~huet/PUBLIC/wsc.pdf> (accessed 2018/10/10).
- Gérard Huet. 2005. A functional toolkit for morphological and phonological processing, application to a Sanskrit tagger. *Journal of Functional Programming*, 15(4):573–614.

- K.A. Subramania Iyer. 1966. *Vākyapadīya of Bharṭṛhari with the Commentaries Vṛtti and Paddhati of Vṛṣabhadeva: Kāṇḍa I*. Number 32 in Deccan College Monograph Series. Deccan College, Postgraduate and Research Institute, Poona.
- Alistair Moffat and Justin Zobel. 1996. Self-indexing inverted files for fast text retrieval. *Transactions on Information Systems*, 14(4):349–379.
- Sheldon Pollock. 2018. “Indian philology”: Edition, interpretation, and difference. In Silvia D’Intino and Sheldon Pollock, editors, *L’espace du sens: Approches de la philologie indienne / The Space of Meaning: Approaches to Indian Philology*, number 84 in Publications de l’Institut de civilisation indienne, pages 3–45, Paris. Collège de France.
- Martin F. Porter. 1980. An algorithm for suffix stripping. *Program*, 14(3):130–137.
- William Pugh. 1990. Skip lists: A probabilistic alternative to balanced trees. *Communications of the ACM*, 33(6):668–676.
- Peter M. Scharf and Malcolm D. Hyman. 2012. *Linguistic Issues in Encoding Sanskrit*. Motilal Banarsidass, Delhi.
- Srigowri and Karunakar. 2013. Gaveṣikā: A search engine for Sanskrit. In Malhar Kulkarni and Chaitali Dangarikar, editors, *Recent Researches in Sanskrit Computational Linguistics: Fifth International Symposium* (IIT Mumbai, India, Jan. 4–6, 2013), Delhi. DK Printworld.
- Trevor Strohman, Donald Metzler, Howard Turtle, and W. Bruce Croft. 2005. Indri: A language model-based search engine for complex queries. In *Proceedings of the International Conference on Intelligent Analysis, Technical Report*.
- Jian Zhang, Jian-Yun Nie, Jianfeng Gao, and Zhou Ming. 2000. On the use of words and n -grams for Chinese information retrieval. In *IRAL ’00: Proceedings of the Fifth International Workshop on Information Retrieval with Asian Languages*, pages 141–148, Hong Kong, China. ACM.

Framework for Question-Answering in Sanskrit through Automated Construction of Knowledge Graphs

Hrishikesh Terdalkar Arnab Bhattacharya
hrishirt@cse.iitk.ac.in arnabb@cse.iitk.ac.in
Dept. of Computer Science and Engineering,
Indian Institute of Technology Kanpur,
India.

Abstract

Sanskrit (saṃskṛta) enjoys one of the largest and most varied literature in the whole world. Extracting the knowledge from it, however, is a challenging task due to multiple reasons including complexity of the language and paucity of standard natural language processing tools. In this paper, we target the problem of building knowledge graphs for particular types of relationships from saṃskṛta texts. We build a natural language question-answering system in saṃskṛta that uses the knowledge graph to answer factoid questions. We design a framework for the overall system and implement two separate instances of the system on human relationships from mahābhārata and rāmāyaṇa, and one instance on synonymous relationships from bhāvaprakāśa nighaṇṭu, a technical text from āyurveda. We show that about 50% of the factoid questions can be answered correctly by the system. More importantly, we analyse the shortcomings of the system in detail for each step, and discuss the possible ways forward.

1 Introduction and Motivation

Sanskrit (IAST¹: saṃskṛta, Devanagari: संस्कृत) is one of the most ancient and richest languages in the world. Its literature boasts of text spanning every facet of life and contains works on mathematics, arts, sciences, religion, philosophy, etc. Unfortunately, the large volume of such works and the relative lack of proficiency in the language have kept treasures in those text hidden from the common man. Unraveling information from these texts in a targeted and systematic manner can not only help in enhancing the knowledge systems but can also revive an interest in the language.

Many of these texts are technical in nature, prime examples of which include āyurveda (आयुर्वेद) texts such as bhāvaprakāśa (भावप्रकाश). The nighaṇṭu (निघण्टु) portion of bhāvaprakāśa is compiled as a glossary of the various substances (dṛavya, द्रव्य) and their properties (guṇa, गुण). Although the information is generally provided in a format that enables scholars to study and analyse it systematically, the large volume of such texts makes it harder for any individual to extract all the information. An automated system can, therefore, greatly aid this processing of information. However, to the best of our knowledge, there does not exist any system that can query this knowledge trove directly and automatically.

While it can be argued that English translations of bhāvaprakāśa nighaṇṭu are available, and building information retrieval (IR) systems for it is a routine for today's IR/NLP tools, there are two main shortcomings of it. First, there are many such nighaṇṭu texts and translations in English are available for only a minuscule number of them. Second, and more importantly, many of the translations of saṃskṛta texts had been done without a proper understanding of the context and culture in which they were composed in the first place. They may have been forced to use English words and phrases that are not a true reflection of the spirit of the original

¹Entire paper uses the IAST encoding scheme for writing Sanskrit words in romanized format. https://en.wikipedia.org/wiki/International_Alphabet_of_Sanskrit_Transliteration

meaning. A notable case in point, as mentioned by Swami Vivekananda himself, is the word śraddhā (श्रद्धा), for which the English translation “regards” is not enough.

Thus, it is always best to rely on the original language. The need of the hour, hence, is to use natural language processing (NLP) of saṃskṛta itself to understand the texts in saṃskṛta.

Our aim in this work is to take the first step towards a concrete NLP task, namely, natural language question-answering in saṃskṛta. In particular, we aim to design a *framework* that processes saṃskṛta texts, extracts the information in it, and stores it in a format that can be queried using questions posed in saṃskṛta.

We propose to store the knowledge base (KB) in a knowledge graph (KG) format. KGs have a rich structure and store the information in the form of entities (as nodes) and relationships (as edges between the nodes). The edges are directed, and both the nodes and edges can store labels describing their attributes. There are multiple off-the-shelf tools available for storing and querying KGs, including graph databases², Property Graphs³, Resource Description Framework (RDF) (Lassila et al. (1998)), Gremlin queries⁴, SPARQL queries⁵, etc. The popularity of KBs such as YAGO (Suchanek et al. (2007)), DBpedia (Auer et al. (2007)) and Freebase (Bollacker et al. (2008)) is a testament to their success.

We also propose question-answering as a concrete example of the use of such KGs and a way of measuring the effectiveness of the system. Various online question-answering fora such as Quora⁶ and quizzes serve as a motivation. We particularly choose the two epics of India, namely, mahābhārata and rāmāyaṇa, categorized as itihāsa in saṃskṛta literature, and questions on human relationships within them, as examples for our framework due to their popularity and ease of establishment of the ground truth. We also work with bhāvaprakāśa nighaṇṭu to highlight the usage for technical texts.

The framework brings to the fore multiple challenges. First, the state of the art of natural language processing in Indian languages, unfortunately, is not as advanced as that in English or some other European languages. Indian languages, and in particular saṃskṛta, are morphologically richer. Therefore, tasks such as lemmatization and parts-of-speech tagging are harder and more error-prone in these languages. Second, some technical texts use their own jargon where certain words may be used in a specific meaning. For example, aṣṭādhyāyī, a work on saṃskṛta grammar by pāṇini uses specific combinations of grammatical cases (vibhakti) to denote which action is to be performed.⁷ Third, names in saṃskṛta are meaningful words and, therefore, identifying named entities is particularly hard. An extremely interesting example in rāmāyaṇa is janaka (जनक), which means “father” in general, but is also the name of a prominent character. Fourth, synonyms are often used to refer to the same person. In many cases, higher-order grammar rules are required to parse the meaning of a word and understand that it is a synonym. For example, it is not mentioned anywhere in the rāmāyaṇa text that dāśarathī is the son of daśaratha and, hence, synonymous to rāma. However, saṃskṛta grammar rules make it obvious to someone who understands the language. Unfortunately, automatic language processing tools are incapable of using such higher-order rules at present.

Nair and Kulkarni (2010) have proposed a model for extracting implicit knowledge from amarakośa and storing it in a structured manner, and have constructed a tool for answering queries using this knowledge. Kulkarni et al. (2010) have built a Sanskrit WordNet⁸ by expanding the

²https://en.wikipedia.org/wiki/Graph_database

³https://en.wikipedia.org/wiki/Graph_database#Labeled-property_graph

⁴<https://docs.janusgraph.org/latest/gremlin.html>

⁵<https://www.w3.org/TR/rdf-sparql-query/>

⁶<https://www.quora.com>

⁷The presence of nominative (prathamā), genitive (ṣaṣṭhī) and locative (saptamī) cases in the same sentence might not convey any special meaning in a normal text, but, in aṣṭādhyāyī, it specifies a process to be followed to transform words, e.g., rule 6.1.77 from aṣṭādhyāyī (iko yaṇaci, इको यणचि) contains words ikaḥ (ṣaṣṭhī), yaṇ (prathamā), aci (saptamī), which is to be interpreted as “an इक् letter which is followed by an अच् letter is converted to a corresponding यण् letter”.

⁸http://www.cfilt.iitb.ac.in/wordnet/webswn/english_version.php

Hindi WordNet. A production grammar for human relationships in *saṃskṛta* was proposed in Bhargava and Lambek (1992). It works for solitary words and cannot be directly used for text. Automatic translation tools, if available, can also be used where the entire text is translated to English and the KG is built from the translated text. However, we could not find any such tools. Although Sanskrit-English dictionaries⁹ provide a word-level translation of selected words from *saṃskṛta* to English, word-level translation often does not produce meaningful or grammatically correct text. We, thus, decided to use only the text as available in *saṃskṛta*. In future, we will explore the use of such tools and methods.

The rest of the paper is organized as follows. In Section 2, we explain the generic framework of the question-answering system. There exist some excellent tools for *saṃskṛta* that aid us in the analysis. For other cases, we build our own heuristic rule-based systems. In Section 3, we describe the automatic construction of the knowledge graph while the details of the various modules of the system are described in Section 4. Since *bhāvaprakāśa nighaṇṭu* is a technical text, we highlight its specialized processing in Section 5. In Section 6, we analyse the results of our experiments. Finally, in Section 7, we discuss the lessons learnt and future directions.

2 Proposed Framework

2.1 Knowledge Graphs (KG)

Knowledge graphs (KG) model real-world entities as nodes. Relationships among the entities are modelled as (directed) edges. For example, in a KG about human relationships in *mahābhārata*, *arjuna* and *abhimanyu* are nodes. They are connected by a directed edge from *arjuna* to *abhimanyu* labelled by the relationship “has-son” (*putra*).

In English, there have been several efforts in automated KG construction, notable among them being YAGO, DBpedia, Freebase, etc. Suchanek et al. (2007) built the YAGO ontology by crawling the Wikipedia and uniting it with WordNet using a combination of both rule-based as well as heuristic methods. Auer et al. (2007) built DBpedia that extracts knowledge present in a structured form on Wikipedia by template detection using pattern matching coupled with post-processing for quality improvement. Bollacker et al. (2008) designed Freebase, a database of tuples that is created, edited and maintained in a collaborative manner. Unfortunately, however, none of the above techniques are applicable for automatically building knowledge graphs in *saṃskṛta*.

Processing of text for YAGO depends on many IR/NLP tools that are available only in English and a handful of other languages, mostly European. The state of the art of these tools in *saṃskṛta* is still not standardized and may not be directly useful. Sanskrit Wikipedia¹⁰ also is not as resourceful as its counterpart in English. Hence, the amount of structured information available there is minuscule compared to the vast *saṃskṛta* literature that is developed over several millennia. Thus, a system such as DBpedia is not possible. A collaborative effort such as Freebase is also ruled out due to a paucity of active *saṃskṛta* users adept in digital technologies. To the best of our knowledge, there is no work that directly builds a knowledge graph from *saṃskṛta* texts.

2.2 Triplets

A common way of encoding the relationship information is in the form of *semantic triplets*. A triplet has the structure [**subject**, **predicate**, **object**] which indicates that the entity **subject** has the relationship **predicate** with the entity **object**. Hence, the fact that *arjuna* has a son *abhimanyu* is encoded as the triplet [*arjuna*, has-son (*putra*), *abhimanyu*] ([अर्जुन, पुत्र, अभिमन्यु]).

The KG is built automatically by extracting such triplets from the text. We target KGs on specific types of relationships, namely, human relationships for epics, and synonymous relation-

⁹<https://www.sanskrit-lexicon.uni-koeln.de/>

¹⁰<https://sa.wikipedia.org/wiki>

ships in *nighaṅṭu*. One of the foremost jobs, therefore, is to identify the relationship words. This is a corpus-independent set and depends only on the language. However, since the text is free-flowing (except in technical texts where there is a structure) and almost always written in poetry in the form of *śloka*, even when a relationship word is identified, the subject and object words may be anywhere around it (both before and after). *śloka* (श्लोक) is a semantic unit in *saṃskṛta* and is equivalent to a *verse*. Sometimes, one or both of these entities may not be even in the same *śloka*. Hence, a *context* window around the relationship word must be defined and searched for the relevant entities. Specifying the length of such a context window is not easy; if it is too short, relationships may be missed, while if it is too long, too many spurious relationships may be inferred. Even identifying the *śloka* boundaries may not always be trivial. Fortunately, however, these boundaries are clearly marked in the texts that we have worked on.

The details of how such triplets are extracted are explained in Section 3. The knowledge graph is maintained in an RDF format as a set of all such extracted triplets.

2.3 Questions

The next important task in the pipeline is to parse the natural language question. Since the question is also in *saṃskṛta*, we adopt similar processing as the text to extract triplets. In this work, we assume only factoid based questions such as “Who is the son of arjuna?” (अर्जुनस्य पुत्रः कः?) The triplet extracted from the above question will be [arjuna, has-son, X] ([अर्जुन, पुत्र, किम्]).

Since *saṃskṛta* is quite free with word ordering, the above question may be asked in different manners, such as अर्जुनस्य पुत्रः कः? or कः अर्जुनस्य पुत्रः? or अर्जुनस्य कः पुत्रः? All of these should yield the same triplet [अर्जुन, पुत्र, किम्].

The *inverse* question may also be asked: “Who is the father of abhimanyu?” (कः अभिमन्योः पिता?) The above can be answered only if it is known that the inverse of “has-father” is the relationship “has-son”. This, again, is a property of the language and must be explicitly mentioned.

Hence, we maintain a map of such inverse relationship rules. Note that it is not always one-to-one. For example, “has-mother” is also the inverse of “has-son”, and “has-father” is the inverse of “has-daughter” as well. Gender information, therefore, becomes important.

We augment the initially built knowledge graph by adding appropriate inverse relationship edges. It is ensured that an inferred inverse relationship does not contradict a directly inferred relationship from the text. The details are in Section 3.4.

Even though the questions are simple and short, they may contain *multiple* triplets. For example, a question पाण्डोः पत्न्याः भ्राता कः? may be asked by someone who does not know what the relation brother-of-wife is called in *saṃskṛta*. This question contains two relationships, पत्नी and भ्राता. The triplet form of these relationships would be [पाण्डु, पत्नी, किम्] corresponding to the subquestion ‘Who was the wife of pāṇḍu?’ and [पत्नी, भ्राता, किम्] corresponding to the subquestion ‘Who was the brother of wife (of pāṇḍu)?’. All of these must be extracted correctly.

Further, they must be linked properly. In the example above, we must ensure that the object of the first triplet is the subject of the second triplet, that is, the correct triplets are [पाण्डु, पत्नी, X] and [X, भ्राता, किम्]. Here, a variable is used to denote the person that satisfies both the triplets.

Once these are correctly linked, a SPARQL query pattern is formed. The SPARQL query equivalent for the above question is

```
SELECT ?A
WHERE {
  :पाण्डु :पत्नी ?X .
  ?X :भ्रातृ ?A .
}
```

This is finally directly queried against the KG, and the answer is returned. Section 4 describes in detail the intricacies of the different steps of the question-answering system.

Figure 1 describes the overall framework. The final accuracy of the system is dependent on each of the modules of the architecture. For example, if the extracting triplets component is

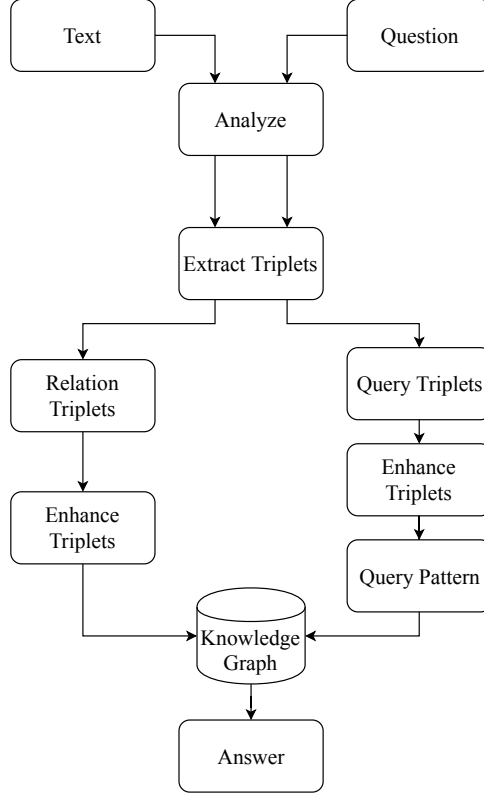


Figure 1: Overall framework of the system.

very erroneous, then neither the KG information is captured correctly, nor is the intention of the question understood. The overall error is a cascading effect of the errors in each of the individual components. Thus, for a successful system, each component must be reasonably accurate.

3 Construction of Knowledge Graph

In this section, we describe in detail the automated construction of knowledge graph (KG). The input consists of *saṃskṛta* text (in digital Unicode format) of an entire work (such as *mahābhārata*, *bhāvaprakāśa nighaṇṭu*, etc.) and the *type* of relationships intended (e.g., human relationships, synonymous words, etc.). The output is a set of triplets in the form [subject, predicate, object] where the predicate is of the relationship type intended and subject and object are entities. If [a, R, b] is an output triplet, then it implies that *object* b is *relation* R of *subject* a.

3.1 Pre-Processing of Text

saṃskṛta is a morphologically rich language. A single word root, called *prātipadika* (प्रातिपदिक), can yield many forms depending on the case, gender and number. Similarly, a single verb root, called *dhātu* (धातु), can lead to many forms as well depending on the tense, person and number. In addition, various prefixes (*upasarga*, उपसर्ग) and suffixes (*pratyaya*, प्रत्यय) get affixed to these forms to generate thousands of other forms.

Further, words are very often joined together to form compound words using either pronunciation rules through a process called *sandhi* (सन्धि) or semantic rules through a process called *samāsa* (समास). Often, both are invoked together, and a series of words are joined together to form one big compound word.

Splitting these compound words into their base words is a highly complicated procedure and may not always be unambiguous. For this step, we make use of the *Sanskrit Sandhi and*

Compound Splitter, a tool¹¹ by Hellwig and Nehrdich (2018). For example, if the input text is कर्णार्जुनयोः को श्रेष्ठः the output is कर्ण-अर्जुनयोः कः श्रेष्ठः.

The next task is to semantically analyze the *form* of the word. Again, we use a third-party *analyser tool*, *The Sanskrit Reader Companion*¹² from *The Sanskrit Heritage Platform* by Goyal et al. (2012). This tool outputs the case (vibhakti, विभक्ति), number (vacana, वचन) and gender (liṅga, लिङ्ग) for each word. The tool uses various abbreviations¹³ to convey the linguistic information.

For the running example, the analysis yields

कर्ण [‘voc.’, ‘sg.’, ‘m.’]

अर्जुन [‘loc.’, ‘du.’, ‘m.’]

किम् [‘nom.’, ‘sg.’, ‘m.’]

श्रेष्ठ [‘nom.’, ‘sg.’, ‘m.’]

Here, ‘nom.’, ‘loc.’ and ‘voc.’ are abbreviations used to denote nominative case (प्रथमा), locative case (सप्तमी) and vocative case (सम्बोधन) respectively. Similarly, ‘sg.’ and ‘du.’ indicate singular and dual number (एकवचन and द्विवचन). While ‘m.’ denotes the masculine gender (पुलिङ्ग).

The word श्रेष्ठ gets correctly analysed: it is in the nominative case, is in singular number, and masculine gender. However, the other words require some more adjustments. For example, the word अर्जुन is shown to be in dual number. This is output since the original compound word consisted of two persons. However, now that they are separated, it should no longer be in dual number, but adjusted to be in singular number. Similarly, the case analysis for कर्ण is wrongly output to be vocative. The reason for this again is the fact that the original structure of the compound word was lost. We adjust the case of previous words in a compound word by adopting the case of the last word in the compound word. Thus, the case for कर्ण is changed to locative, since that is the case for अर्जुन .

3.2 Identifying Relationship Words

Given a particular relationship type, the set of words pertaining to it is corpus-independent and is a property of the language. For example, if human relationships are targeted, in saṃskṛta, the (roots of the) relevant words are pitṛ (father, पितृ), mātṛ (mother, मातृ), putra (son, पुत्र), putrī (daughter, पुत्री), pati (husband, पति), patnī (wife, पत्नी), etc. Of course, these words can appear in various forms. More importantly, their synonyms can also appear. For example, all the words दुहितृ, तनया, आत्मजा mean पुत्री.

While these can be learned, since the set is mostly fixed, we have employed a key-value based approach where we have listed many of such relationship words along with their synonyms. For each such group of synonyms, there is a canonical word (e.g., पुत्री for the group of words indicating daughter) that is used in the KG.

The identification of a relationship word is simply a match from this entire set of words.

3.3 Identification of Triplets

Once a relationship word is identified, it forms the predicate of a triplet. The next task, therefore, is to identify the subject and object corresponding to it.

It is expected that the subject and object entities will not be too far off from the predicate word. To bound the sphere of influence or context, we use śloka (श्लोक) boundaries. Each śloka considered as a semantic unit and is akin to a verse. Fortunately, for the texts we have used, the śloka boundaries are clearly marked. In this work, we restrict the context to be *one* śloka before and after the one where the predicate is found, i.e., a total of 3 śloka.

Since subjects and objects are entities, they generally occur as nouns in a language. The analyser tool (*The Sanskrit Reader Companion*) described earlier marks the parts-of-speech tags

¹¹<https://github.com/OliverHellwig/sanskrit/tree/master/papers/2018emnlp>

¹²<https://sanskrit.inria.fr/DICO/reader.fr.html>

¹³All the abbreviations used by the tool are listed at <https://sanskrit.inria.fr/abrevs.pdf>.

of words. It, however, does not distinguish between nouns, pronouns and adjectives. Since there is a fixed set of pronouns for *saṃskṛta*, we use that set to correct some of the nouns. We, however, fail to distinguish the adjectives from the nouns in a satisfactory and consistent manner. This is a major future work.

Within the nouns (and adjectives), we look for those that are in the *genitive* case (षष्ठी विभक्ति). The genitive case pertains to the *ṣaṣṭhī vibhakti* (genitive case) and denotes *sambandha* (सम्बन्ध). The word *sambandha* in *saṃskṛta* literally means relationship and, therefore, a noun exhibiting genitive case is the most likely candidate for a subject. For example, the अर्जुनस्य पुत्रः अभिमन्युः आसीत् means *abhimanyu* was son of *arjuna*. Here, ‘of *arjuna*’ is expressed by the genitive case of the word (अर्जुन), i.e., अर्जुनस्य. Hence, all such nouns in the genitive case are marked as subjects.

The relationship word or the predicate can be in different cases, numbers and gender, though. Since the object follows the predicate, according to *saṃskṛta* grammar, it must be in the same case, number and gender as the predicate. We use this rule to extract objects. To be precise, an object is a noun that exhibits the same case, number and gender as the predicate word. In the sentence अर्जुनस्य पुत्रः अभिमन्युः आसीत्, word पुत्रः is the predicate word and the word अभिमन्युः is the object and both of these words are in the nominative case (प्रथमा विभक्ति).

We insert all such extracted triplets in the KG. We assume that if an entity appears multiple times, it refers to the *same* person. The above assumption is almost always correct barring some exceptional cases.¹⁴

3.4 Enhancement of Relationships

As explained earlier (in Section 2), just the base relationships may not always be enough to answer a question. If the triplet [*arjuna*, *has-son*, *abhimanyu*] ([अर्जुन, पुत्र, अभिमन्यु]) is stored, the question “Who is the father of *abhimanyu*?” (कः अभिमन्योः पिता?) cannot be answered, even though the information is present.

To be able to answer such queries, we have enhanced the KG with inverse relationships. For example, the inverse of “*has-father*” is “*has-son*”. This, again, is a property of the language and are explicitly stored.

As discussed earlier, the inverse relationships are not always one-to-one. For example, “*has-mother*” is also the inverse of “*has-son*”, and “*has-father*” is the inverse of “*has-daughter*” as well. Hence, we use the gender information of the subject and the object to disambiguate.

The complication does not end here. Imagine a question “Who is maternal uncle of *Nakula*?” (नकुलस्य मातुलः कः). This information may not be directly stored in the KG. The relationship *मातुल* is a composition of *मातृ* and *भ्रातृ*. These components [नकुल, मातृ, माद्री] and [माद्री, भ्रातृ, शल्य] may be present in the KG. Again, the situation is that the KG contains the information but cannot answer the question.

To solve this, derived relations could be broken into their component base parts. Thus, “*has-maternal uncle*” is stored as “*has-mother*” and “*has-brother*” with an additional (possibly unnamed) node in between. In particular, from the triplet [नकुल, मातुल, शल्य], two more triplets [नकुल, मातृ, X] and [X, भ्रातृ, शल्य] could be generated. If there is already such a node X, it could be used; otherwise, a new node could be created. However, addition of such *dummy* nodes has not been explored in this work.

We achieve the same result by handling this issue at the time of querying. This is discussed in Section 4.2. We maintain a list of relationships and their possible derivations from base relationships. Once more this mapping is rarely one-to-one. For example, “*brother-of*” can be composed of “*son-of-father*” and “*son-of-mother*”. Also, the gender must be taken care of.

A particularly interesting case is “*has-ancestor*” and “*has-descendant*”. These are recursive relationships, and the depth of recursion can be anything, i.e., a ‘father’ is an ancestor, so is an ‘ancestor-of-father’, and so on. We do not handle these cases in the current work.

¹⁴*karna* was the son of *kunti*, and one of the *kaurava* was also named *karna*.

4 Question-Answering

We now describe one application, that of question-answering. We assume that the questions are asked directly in *saṃskṛta* and are about factoids, i.e., about a single piece of information. We also assume that the questions are only about the relationships that the knowledge graph encodes. If not, the question is ignored, since clearly the KG is incapable of answering it. Further, the questions are assumed to be short and consist of a single sentence only.

The question is first pre-processed in the same manner as the text (Section 3.1). To be more precise, compound words are split using *Sanskrit Sandhi and Compound Splitter* a tool by Hellwig and Nehrdich (2018), the component words are analysed using *The Sanskrit Reader Companion* from *The Sanskrit Heritage Site*, and relationship words and nouns are identified. Next, triplets are extracted.

4.1 Identifying Triplets

A blank triplet is initialized. The question words are scanned one by one. For each word, it is determined if it can be a subject word, a predicate word or an object word. If the word is a noun in genitive case but is not a relationship word, then it is likely to be a subject word. The relationship words directly give the predicates. The object word is generally in the nominative case. For example, consider the question अर्जुनस्य पुत्रः कः? (“Who is the son of arjuna?”). Since अर्जुन is in genitive case, it is the subject. The word पुत्र is the predicate. The object is किम्. The triplet formed, therefore, is [अर्जुन, पुत्र, किम्].

Once a triplet is filled up, another new triplet is initialized. This is necessary since there may be chain questions of the form अर्जुनस्य पुत्रस्य पुत्रः कः? The triplets generated from this are [अर्जुन, पुत्र, X] and [X, पुत्र, किम्].

The process goes on till all the words in the question are processed.

At the end of this phase, the triplets thus formed are called *query triplets*.

4.2 Enhancing Triplets

Each query triplet is next enhanced to a set of triplets, called the *enhanced triplet set*. The rules for enhancing the relationship of a query triplet is the same as that used in processing the KG triplets. In particular, each complex relation is broken into its constituent parts and new triplets are created using the aforementioned mapping of relationships to its constituents.

Suppose, a predicate (i.e., relation) R can be decomposed to two base predicates R1 and R2. Then, if a query triplet is of the form [A, R, B], then two triplets of the form [A, R1, X] and [X, R2, B] are generated. Note that {[A, R, B]} and {[A, R1, X], [X, R2, B]} are *equivalent expressions* and either of them can return the correct answer from the KG. However, since it is not known which information is stored in the KG, *both* are used.

Thus, each query triplet QT_i is replaced by its enhanced triplet set $ET_i = \{QT_i\} \cup IT_i^j$ where IT_i^j is a set of triplets inferred from QT_i , as shown in the example below.

For the question अर्जुनस्य मातुलस्य पिता कः, we first obtain the triplets {[अर्जुन, मातुल, X], [X, पितृ, किम्]}. These triplets are then enhanced by appropriately splitting the relationship मातुल using the rule मातुल = मातृ + भ्रातृ. Here, $QT = [अर्जुन, मातुल, X]$ and $IT = \{[अर्जुन, मातृ, Y], [Y, भ्रातृ, X]\}$. As a result, we get two triplet sequences for this question, {[अर्जुन, मातृ, Y], [Y, भ्रातृ, X], [X, पितृ, किम्]} and {[अर्जुन, मातुल, X], [X, पितृ, किम्]}.

4.3 Query Pattern

If the question contains only one query triplet, then members of its enhanced triplet set form the alternate query patterns. Suppose, however, the question contains n query triplets with their corresponding n enhanced triplet sets ET_1, ET_2, \dots, ET_n . The Cartesian product of the elements of these sets form the *alternate query patterns*. Thus, if there are 2 enhanced sets with 2 and 3 elements in them, the total number of alternate query patterns is $2 \times 3 = 6$.

Each of these alternate query patterns are posed to the KG and answer triplets are returned. The correct field of the answer triplet is returned as the factoid answer.

We have not encountered a case where alternate query patterns return different answers. If, however, such a situation arises, a further disambiguation step (possibly using majority voting, etc.) is required.

5 Technical Texts

We have chosen a technical text *bhāvaprakāśa* which is one of the important texts from *āyurveda*. *bhāvaprakāśa nighaṇṭu* is a glossary chapter from this text, which contains detailed information about the medicinal properties of various plants, animals and minerals written in a *śloka* format. There are 23 *adhyāya* in this chapter. Being a technical text, *bhāvaprakāśa nighaṇṭu* has more structure than *rāmāyaṇa* or *mahābhārata*.

5.1 Structure

The text *bhāvaprakāśa nighaṇṭu* loosely adheres to the following structure.

- Substances (*dravya*, द्रव्य) with similar properties or from the same class occur in the same chapter. For example, all the flowers are in one chapter, all the metals are in another chapter.
- Each chapter consists of various *blocks* (sets of consecutive *śloka*), where each block speaks about one substance.
- Each block generally has the following internal components:
 - Synonyms of the concerned substance
 - Where that substance can be found
 - Properties of the substance. e.g., colour, smell, texture, composition and other medicinal properties
 - Differences between the different varieties of the substance

While the blocks are structured to some extent, the following deviations exist.

- The length of each block is not fixed.
- The number of synonyms of each substance are not fixed.
- The order of the components of the block varies from substance to substance to a certain extent.
- Some of the internal components may, at times, be absent such as the varieties of a substance.

Importantly, the *separation* between two consecutive blocks is not marked in the text.

These points of deviation from the pattern act as hurdles in the process of understanding and exploiting the structure of a text to extract information. Understanding the structure of a text can be a challenging task. We have taken the help of domain experts¹⁵ to form our understanding of the structure described above.

Properties (*guṇa*, गुण) are of the form (*name*, *value*). A property value can be directly attached to a substance, or it can be attached through a *property-name*. For example, a substance is “red”, or, a substance has *colour* “red”.

Relationships of interest can be of a number of types. Some of them are: (*substance-1*, *is-synonym-of*, *substance-2*), (*substance*, *property-name*, *property-value*), (*substance*,

¹⁵We acknowledge Dr. Sai Susarla, Dean at Maharshi Veda Vyas MIT School of Vedic Sciences, Pune, India, and his team for sharing their expertise with us.

Words			Nouns		
adhyāya 1	adhyāya 2	All adhyāya	adhyāya 1	adhyāya 2	All adhyāya
(च, 127)	(च, 56)	(च, 946)	(कफ, 53)	(तिक्त, 39)	(पित्त, 461)
(तद्, 85)	(तिक्त, 39)	(तद्, 786)	(उष्ण, 47)	(कफ, 31)	(कफ, 438)
(किम्, 55)	(लघु, 37)	(पित्त, 461)	(पित्त, 45)	(विष, 22)	(गुरु, 254)
(कफ, 53)	(कफ, 31)	(कफ, 438)	(तिक्त, 34)	(उष्ण, 21)	(उष्ण, 240)
(उष्ण, 47)	(तु, 24)	(तु, 394)	(वात, 32)	(पित्त, 19)	(तिक्त, 237)
(पित्त, 45)	(किम्, 24)	(लघु, 321)	(शूल, 29)	(कुष्ठ, 18)	(वात, 204)
(तु, 39)	(तद्, 22)	(वा, 278)	(कुष्ठ, 28)	(अस्त्र, 18)	(स्मृत, 194)
(तथा, 35)	(विष, 22)	(अपि, 268)	(कास, 25)	(स्मृत, 17)	(कुष्ठ, 177)
(अपि, 34)	(उष्ण, 21)	(किम्, 266)	(कटु, 25)	(कण्डु, 16)	(गुण, 160)
(तिक्त, 34)	(हृत्, 20)	(गुरु, 254)	(श्वस, 24)	(कटु, 16)	(लघु, 160)

Table 1: Top-10 most frequent words, nouns and their frequencies from bhāvaprakāśa nighaṇṭu.

Counts	Words, Nouns, Properties, Non-Properties, Special Words, Pronouns, Verbs, Case- i Nouns ($i = 1, \dots, 8$), Number- j Nouns ($j =$ singular, dual, plural)
Ratio to Words	Nouns, Properties, Non-Properties, Special Words
Ratio to Nouns	Properties, Non-Properties, Special Words, Case- i Nouns ($i = 1, \dots, 8$), Number- j Nouns ($j =$ singular, dual, plural)
Other Ratios	Properties to Non-Properties, Non-Properties to Properties, Special Words to Properties, Special Words to Non-Properties

Table 2: Features of a śloka.

has-property, property-value), (substance, found-at, location).

When a property is directly attached to a substance, we assume the relationship to be has-property.

We have currently focused our efforts on a single relationship in the bhāvaprakāśa nighaṇṭu, namely, is-synonym-of. In other words, the triplets that we are interested in are of the form (substance-1, is-synonym-of, substance-2). Since the predicate is same for all triplets, we choose to get rid of it and think of the problem as simply *finding pairs of synonyms*.

This task is subdivided into two tasks, (1) finding śloka that contain the synonyms, and (2) given such a śloka, finding pairs of synonyms from it.

5.2 Property Words

The corpus is initially pre-processed in a similar manner as described in Section 3.1. However, a next layer of processing is done to extract more information.

The set of properties is a relatively small set of words. The names and values of these properties together are called *property words*. Since the *property words* recur heavily in every block that describes a substance, they are expected to have much higher frequencies than the names of substances. We test this hypothesis by performing a frequency analysis of the top words and nouns in the entire text.

Table 1 lists the top-10 most frequent words and nouns along with their frequencies. Notice that most frequent words also contain stopwords like च, तद् etc., while the list of nouns indicates that the standard property words such as वात, पित्त, कफ have a high frequency. Following this empirical evidence, we choose the top-50 most frequent nouns as “properties”. The substances are chosen from the rest of the nouns.

5.3 Synonym śloka Identification

Generally, the different synonyms of a substance are listed in a single śloka at the beginning of a block. A set $\{n_1, n_2, \dots, n_k\}$ of nouns is called a *synonym-group* if every n_i is a synonym of every other n_j . Any such (n_i, n_j) pair is called a *synonym-pair*. A śloka that gives information about a synonym-group or synonym-pairs is referred to as a *synonym śloka*. The first task is to identify instances of such synonym śloka.

To identify a synonym śloka automatically, we use various linguistic features of a śloka and then use them in a classifier. We create a 42-dimensional feature vector per śloka. Table 2 enlists all the features used. The features are based on counts and their ratios. Some of the notable features include number of nouns, pronouns and verbs, number of property words present in a śloka, ratios of property words to total number of words, number of words in each case (विभक्ति), and so on. The category “specials” contains adverbs, conjunctions and prepositions.

Once each śloka is converted into a 42-dimensional feature vector, various classifiers and ensemble methods are used to classify into a synonym śloka or otherwise.

5.4 Identifying Synonymous Nouns

Once a synonym śloka is identified, the next task is to identify the synonyms from it. Given a synonym śloka, we first exclude all the property words from it. We next consider the list of all the nouns in the śloka: $\{n_1, n_2, \dots, n_k\}$.

We call a pair of nouns (n_i, n_j) a *synonym pair* if both n_i and n_j have the same case (विभक्ति) as well as the same number (वचन). We do not use the gender (लिङ्ग) information since there are examples of synonymous substance names that belong to different genders. For example, चव्य (neuter), चव्यिका (feminine) and ऊषणा (feminine) form a synonym group.

6 Experiments and Results

In this section, we present our experiments and discuss the results. The code is written in Python3. All experiments are done on Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz system with 16 GB RAM running Ubuntu 16.04.6 OS. RDF is used for storing the knowledge graph, and querying is done using SPARQL querying language. Python library RDFlib is used for working with RDF and SPARQL.

6.1 Datasets

We have worked with texts containing two types of relationships:

1. **Human Relationships:** The two well-known epics of ancient India, rāmāyaṇa and mahābhārata, contain numerous characters and relationships among them. We have, thus, used them as datasets for human relationships.
2. **Synonymous Relationships of Substances:** āyurveda, the traditional Indian system of medicine, has a rich source of information about medicinal plants and substances. We considered bhāvaprakāśa nighaṇṭu, a glossary chapter of the āyurveda text bhāvaprakāśa as the dataset. It enlists numerous medicinal plants and substances along with their properties and inter-relationships. In this work, we only consider the relationship “is-synonym-of”.

Table 3 shows the statistics about the datasets considered.

6.2 Knowledge Graph from rāmāyaṇa and mahābhārata

Table 4 shows the various statistics about the knowledge graphs constructed from the datasets rāmāyaṇa and mahābhārata.

While pre-processing the text requires a large amount of time, the other steps are significantly faster. The querying times are in microseconds.

Dataset	rāmāyaṇa	mahābhārata	bhāvaprakāśa nighaṇṭu
Type	Classical	Classical	Technical
Chapters	7 (kāṇḍa)	18 (parvan)	23 (adhyāya)
Documents	606	2,327	23
śloka	23,934	81,603	4,244
Words (total)	2,69,603	17,49,709	31,532
Words (unique)	16,083	55,366	5,976
Nouns (total)	1,52,878	6,36,781	19,689
Nouns (unique)	9,553	20,545	3,684

Table 3: Statistics of the various datasets used.

		rāmāyaṇa	mahābhārata
Time taken	Preprocessing	~ 3.5 days	~ 13 days
	Triplet Extraction	14.18 sec	57.19 sec
	Triplet Enhancement	0.40 sec	2.05 sec
Before enhancement	Entities (Nodes)	1,711	3,552
	Triplets (Edges)	6,155	18,936
	Type of Relations	24	25
After enhancement	Entities (Nodes)	1,711	3,552
	Triplets (Edges)	16,367	48,395
	Type of Relations	27	27

Table 4: Statistics of the knowledge graphs for the human relationships.

6.2.1 Questions

To evaluate the performance of the question-answering system, we collected 35 questions from rāmāyaṇa and 45 questions from mahābhārata from 12 different users, with each user contributing between 5-10 questions.

6.2.2 Performance

We evaluate the performance of the system for three tasks.

- **QParse** refers to the query parsing task. If the query pattern is correctly formed from the natural language question, we count it as a success; otherwise, it is a failure.
- **QCond** is the conditional question answering task subject to correct query formation. A success is counted only if the answer to the question is completely correct.
- **QAll** is the overall question answering task.

Table 5 demonstrates the performance of our system on the collected questions. The query parsing task is fairly accurate. However, the accuracy of question-answering has a lot of scope for improvement. We next analyze some of the reasons for failure.

6.3 Analysis of Wrong Answers

We analyze the wrong answers in two phases: parsing errors and answering errors.

Text	Task	Total	Found	Correct	Precision	Recall	F1
rāmāyaṇa	QParse	35	33	27	0.82	0.77	0.79
	QCond	27	19	09	0.47	0.33	0.39
	QAll	35	20	10	0.50	0.29	0.37
mahābhārata	QParse	45	45	41	0.91	0.91	0.91
	QCond	41	36	22	0.61	0.54	0.57
	QAll	45	40	23	0.58	0.51	0.54
Combined	QParse	80	78	68	0.87	0.85	0.86
	QCond	60	55	31	0.56	0.46	0.50
	QAll	80	60	33	0.55	0.41	0.47

Table 5: Performance of the question-answering tasks.

6.3.1 Parsing Errors

Following are some examples of queries that got incorrectly parsed.

- गान्धार्याः पुत्राणाम् नामानि कानि → [गान्धारी, पुत्र, किम्]
The question expects all the names of sons of gāndhārī गान्धारी but the parsed query only asks for the name of ‘a son’ of गान्धारी. This error originates from the fact that we have not considered the number (वचन) of the relationship word while parsing the question. Strictly speaking, however, the question is not a simple factoid question. Nevertheless, number (वचन) can be considered, and all triplets that satisfy the criteria can be returned.
- कर्णार्जुनयोः कः सम्बन्धः → [किम्, किम्, सम्बन्ध]
There are patterns in the question set that are not handled by our algorithm. For example, the algorithm did not handle the way of asking the relationship between two people using the word सम्बन्ध and, thus, results in a triplet that does not make sense. If the same question was phrased as कर्णः अर्जुनस्य कः, our algorithm would be able to parse the question to give [अर्जुन, किम्, कर्ण]. Questions like कर्णः अर्जुनस्य कः, अर्जुनस्य कर्णः कः, अर्जुनस्य कः कर्णः and कर्णः कः अर्जुनस्य also get parsed correctly to [अर्जुन, किम्, कर्ण].
- विवाहः अर्जुनस्य अभवत् कया सह → [अर्जुन, किम्, विवाह]
The question parsing algorithm, while tolerant to some extent, is not fully robust to free word order. An occurrence of विवाह word needs to be followed by the instrumental case (तृतीया) word, followed by सह for it to be parsed correctly. Thus, if the question is changed to अर्जुनस्य विवाहः कया सह अभवत्, it will get parsed correctly to yield [अर्जुन, पत्नी, किम्].

6.3.2 Answering Errors

Out of the queries that correctly get parsed, following are the queries which we cannot find the answer due to the inability of performing path queries.

- ऊर्मिला दशरथस्य का → [दशरथ, किम्, ऊर्मिला]
This question would have got answered only if there is a direct edge between दशरथ and ऊर्मिला. If there is no direct edge, but an edge between दशरथ and लक्ष्मण exists along with the edge between लक्ष्मण and ऊर्मिला, then this answer should have been found. Our inability to pose it as a graph path searching query is the cause of this failure.
- हनुमतः पिता कः → [हनुमत, पितृ, किम्]
We correctly parse this question and there exists a triplet [मारुति, पितृ, पवन]. However, as the information that मारुति is another name of हनुमत is not present in the knowledge graph, resulting in the failure to answer this question.

śloka	sandhi-samāsa split
अनिलस्य शिवा भार्या तस्याः पुत्रो मनोजवः । अविज्ञातगतिश्चैव द्वौ पुत्रावनिलस्य तु ॥ २५ ॥ प्रत्यूषस्य विदुः पुत्रमृषिं नाम्नाऽथ देवलम् । द्वौ पुत्रौ देवलस्यापि क्षमावन्तौ मनीषिणौ । बृहस्पतेस्तु भगिनी वरस्त्री ब्रह्मवादिनी ॥ २६ ॥ योगसिद्धा जगत्कृत्स्नमसक्ता विचचार ह । प्रभासस्य तु भार्या सा वसूनामष्टमस्य ह ॥ २७ ॥	अनिलस्य शिवा भार्या तस्याः पुत्रः मनोजवः । अविज्ञात-गतिः-च-एव द्वौ पुत्रौ=अनिलस्य तु ॥ २५ ॥ प्रत्यूषस्य विदुः पुत्रम-ऋषिम् नाम्ना-अथ देवलम् । द्वौ पुत्रौ देवलस्य-अपि क्षमावन्तौ मनीषिणौ । बृहस्पतेः-तु भगिनी वर-स्त्री ब्रह्म-वादिनी ॥ २६ ॥ योग-सिद्धाः जगत्-कृत्स्नम्-असक्ता विचचार ह । प्रभासस्य तु भार्या सा वसूनाम्-अष्टमस्य ह ॥ २७ ॥

Table 6: śloka 25, 26, 27 from adhyāya 67 of ādi parvan in mahābhārata.

- पुरोः कः वंशजः यस्य पुत्रः अर्जुनः → [पुरु, वंशज, किम्], [यद्, पुत्र, अर्जुन]
Again, despite getting correctly parsed, since we cannot follow the “has-son” relationship arbitrary number of times, this query cannot be answered.

6.3.3 Correct Answers despite Wrong Parsing

Interestingly, there are cases when despite the query being parsed incorrectly, the correct answer exists in the result set. The following examples highlight two such cases.

- रावणस्य कनिष्ठतमः भ्राता कः → [रावण, भ्रातृ, किम्]
The triplet is incorrectly formed, since we did not capture the information कनिष्ठतमः (youngest). However, the correct answer, विभीषण, being a brother of रावण, is captured in the result set. The question is, thus, deemed to be answered correctly.
- भीमस्य अग्रजः कः आसीत् → [भीम, भ्रातृ, किम्]
Similar to the previous question, we classify the formed triplet as incorrect, for missing the quality ‘elder’. However, answers found do contain the correct answers युधिष्ठिर and कर्ण.

6.4 Analysis of Errors in KG Triplets

We now take a look at in-depth analysis of some incorrect triplets retrieved by our method and investigate the reasons behind the failure. For this purpose, we consider a small extract from the corpus and follow the entire pipeline of forming the triplets.

Table 6 gives an extract containing three śloka (25, 26 and 27) from adhyāya 67 of the ādi parvan in mahābhārata. Table 7, Table 8 and Table 9 contain the detailed analysis of these śloka as well as a classification of the errors in the analysis.

6.4.1 Types of Errors

We now discuss the possible errors, as exemplified in the analysis tables 7, 8 and 9.

- **AnalysisError:**
This is an error in the analysis obtained from *The Sanskrit Heritage Parser*. For example, the word भार्या in śloka 25 is analysed as a form of भारि instead of a form of भार्या. Thus, the prātipadika identified is wrong. This also results in the other analysis details such as case, gender and number, being wrong. It should be noted that words can be analyzed differently in different contexts. For example, the word भार्या, if analyzed standalone as a word, can get analyzed correctly; however, in the current context, it results in an erroneous analysis.¹⁶
- **OversplitError:**
This is an error in the sandhi and samāsa splitter, where a word that should not have been split is split. For example, in śloka 26, वरस्त्री is wrongly oversplit as वर and स्त्री, and ब्रह्मवादिनी

¹⁶Erroneous analysis of भार्या: <https://sanskrit.inria.fr/cgi-bin/SKT/sktreader.cgi?lex=SH&st=t&us=f&cp=t&text=anilasya+zivaa+bhaaryaa+tasyaa.h+putra.h+manojava.h&t=VH&mode=p>

Word	Root	Analysis	Is-Noun	Is-Verb	Error
अनिलस्य	अनिल	['g.', 'sg.', 'm.']	True	False	
शिवा	शिव	['nom.', 'sg.', 'f.']	True	False	
भार्या	भारि	['i.', 'sg.', 'f.']	True	False	AnalysisError
तस्याः	तद्	['g.', 'sg.', 'f.']	False	False	
पुत्रः	पुत्र	['nom.', 'sg.', 'm.']	True	False	
मनो जवः	मनोजव	['nom.', 'sg.', 'm.']	True	False	Corrected
अविज्ञा	अविज्ञ	['nom.', 'sg.', 'f.']	True	False	OversplitError
आत	अत्	['pft.', 'ac.', 'pl.', '2']	False	True	OversplitError
गतिः	गति	['nom.', 'sg.', 'f.']	True	False	OversplitError
च	च	['conj.']	False	False	
एव	एव	['prep.']	False	False	
द्वौ	द्व	['acc.', 'du.', 'm.']	True	False	
पुत्रौ	पुत्र	['acc.', 'du.', 'm.']	True	False	
अनिलस्य	अनिल	['g.', 'sg.', 'm.']	True	False	
तु	तु	['conj.']	False	False	

Table 7: Analysis of śloka 25.

as ब्रह्म and वादिन्. Sometimes a word is erroneously oversplit by the analyser as well. Again, in śloka 26, for example, वादिन् is erroneously split as वा and आदिन्.

- **SandhiSamaasaError:**

There can be error in analyzing the correct sandhi and samāsa in a word. In other words, when a word is broken, the constituent words can be erroneous. For example, in śloka 27, योगसिद्धा जगत् is split as योग, सिद्धाः and जगत्, where योगसिद्धा, in addition to being oversplit, is also changed into plural form.

6.4.2 Extracting Triplets

After obtaining the analysis, when we proceed to extract triplets as mentioned, we tried using 4 different filters for extracting triplets. In every filter, the case of the subject word must be sixth (षष्ठी) and the gender of the object word must match with the gender of the predicate word. Filters differ in the allowed positions of subject and object words relative to the predicate word as well whether the number (वचन) of the object is matched or not.

Table 10 describe the different filters. Filter 1 is the superset of other filters and Filter 2 is the superset of Filter 3 and Filter 4.

Through empirical evidence, we found that Filter 2, although being stricter than Filter 1, still captures roughly the same number of triplets while reducing the errors. Filter 3 and Filter 4, while exhibiting fewer mistakes, find fewer correct triplets as well. While we acknowledge that such an analysis is required on a larger scale to decide among the filters, for our purposes, we choose Filter 2 based on the empirical evidence, and proceed further.

6.4.3 Analysis of Incorrect Triplets

In this section, we take a look at some wrong triplets that were retrieved and the reasons behind their retrieval.

- (प्रत्यूष, पुत्र, मनीषिन)

śloka 26, listed in Table 6 contains two relationship words, पुत्रम् and पुत्रौ. The first one is used in relation to देवल who is the son of प्रत्यूष, and the triplet (प्रत्यूष, पुत्र, देवल) is found correctly. However, because of the presence of the second word पुत्रौ, which is actually used with देवलस्य, a wrong triplet (प्रत्यूष, पुत्र, मनीषिन) is formed. Due to the same reason, (प्रत्यूष, पुत्र, क्षमावत) is also

Word	Root	Analysis	Is-Noun	Is-Verb	Error
प्रत्यूषस्य	प्रत्यूष	['g.', 'sg.', 'm.']	True	False	
विदुः	विद्	['pft.', 'ac.', 'pl.', '3']	False	True	
पुत्रम्	पुत्र	['acc.', 'sg.', 'm.']	True	False	
ऋषिम्	ऋषि	['acc.', 'sg.', 'm.']	True	False	
नाम्ना	नामन्	['adv.']	False	False	
अथ	अथ	['conj.']	False	False	
देवलम्	देवल	['acc.', 'sg.', 'm.']	True	False	
द्वौ	द्व	['acc.', 'du.', 'm.']	True	False	
पुत्रौ	पुत्र	['acc.', 'du.', 'm.']	True	False	
देवलस्य	देवल	['g.', 'sg.', 'm.']	True	False	
अपि	अपि	['conj.']	False	False	
क्षमावन्तौ	क्षमावत्	['acc.', 'du.', 'm.']	True	False	
मनीषिणौ	मनीषिन्	['acc.', 'du.', 'm.']	True	False	
बृहस्पतेः	बृहस्पति	['g.', 'sg.', 'm.']	True	False	
तु	तु	['conj.']	False	False	
भगिनी	भगिनी	['nom.', 'sg.', 'f.']	True	False	
वर	वर	['voc.', 'sg.', 'm.']	True	False	OversplitError
स्त्री	स्त्री	['nom.', 'sg.', 'f.']	True	False	OversplitError
ब्रह्म	ब्रह्मन्	['acc.', 'sg.', 'n.']	True	False	OversplitError
वा	वा	['conj.']	False	False	OversplitError
आदिनी	आदिन्	['acc.', 'du.', 'n.']	True	False	OversplitError

Table 8: Analysis of śloka 26.

Word	Root	Analysis	Is-Noun	Is-Verb	Error
योग	योग	['voc.', 'sg.', 'm.']	True	False	OversplitError, AnalysisError
सिद्धाः	सिद्ध	['acc.', 'pl.', 'f.']	True	False	OversplitError, SandhiSamaasaError
जगत्	जगत्	['acc.', 'sg.', 'n.']	True	False	
कृत्स्नम्	कृत्स्न	['acc.', 'sg.', 'm.']	True	False	
असक्ता	असक्त	['nom.', 'sg.', 'f.']	True	False	
विचचार	वि-चर्	['pft.', 'ac.', 'sg.', '3']	False	True	
ह	ह	['part.']	False	False	
प्रभासस्य	प्रभास	['g.', 'sg.', 'm.']	True	False	
तु	तु	['conj.']	False	False	
भार्या	भार्य	['nom.', 'sg.', 'f.']	True	False	
सा	तद्	['nom.', 'sg.', 'f.']	False	False	
वसूनाम्	वसु	['g.', 'pl.', 'm.']	True	False	
अष्टमस्य	अष्टम	['g.', 'sg.', 'm.']	True	False	
ह	ह	['part.']	False	False	

Table 9: Analysis of śloka 27.

found. Since the context for finding relationships covers the full śloka, when a single śloka contain multiple relationships, such errors occur. If sentences were instead used, the error could have been reduced. However, there do not exist clear sentence boundaries.

Filter	Position of subject	Position of object	Number (वचन) of object
1	Either side of predicate	Either side of predicate	Does not matter
2	Either side of predicate	Either side of predicate	Must match predicate
3	Before predicate	After predicate	Must match predicate
4	After predicate	Before predicate	Must match predicate

Table 10: Filters for extracting triplets.

Scenario	Training Set	Testing Set
S1	First 20% of adhyāya 1	Rest 80% of adhyāya 1
S2	First 20% of adhyāya 2	Rest 80% of adhyāya 2
S3	adhyāya 1	adhyāya 2
S4	adhyāya 2	adhyāya 1

Table 11: Training and testing scenarios on bhāvaprakāśa nighaṇṭu.

- (बृहस्पति, भगिनी, स्त्री)

As discussed in Section 6.4.1, the word वरस्त्री gets oversplit wrongly into वर and स्त्री, and the split words are analysed separately, resulting in the wrong triplet. Even if this split did not occur, we would have got वरस्त्री as the object in this triplet. This is wrong since this is actually an adjective used for the sister of बृहस्पति. Since we currently do not have any mechanism of distinguishing between nouns and adjectives, it would have resulted in incorrect triplets.

We next examine some triplets that should have been found but were not found and the reasons behind their non-retrieval.

- (अनिल, पत्नी, शिवा)

The relationship word that occurs in śloka 25 in Table 6 is भार्या, which suffers an AnalysisError and is identified as तृतीया of भारि instead of प्रथमा of भार्या. Due to the root word (प्रातिपदिक) itself being misidentified, it is not recognized as a relationship word and thus, does not satisfy the filtering criterion. Consequently, the triplet (अनिल, पत्नी, शिवा) is missed.

- (प्रभास, पत्नी, ब्रह्मवादिनी)

In śloka 27, भार्या of प्रभास is referred to with a pronoun सा, which is connected to a noun in the previous śloka. To correctly identify the triplet (प्रभास, पत्नी, ब्रह्मवादिनी), we would need a mechanism to connect pronouns to their proper subjects. We do not handle this currently.

6.5 Synonym Identification from bhāvaprakāśa nighaṇṭu

Questions for the bhāvaprakāśa are implicit, as we are considering only the synonymous relationship. Therefore, the evaluation is performed on the *synonym groups* and *synonym pairs* identification. We created ground truth for the first two adhyāya of bhāvaprakāśa nighaṇṭu. adhyāya 1 contains 261 śloka, while adhyāya 2 contains 131 śloka. For each of these śloka, we first identified if it is a *synonym śloka*. If it is so, we next extracted the list of synonymous words contained in it.

6.5.1 Classification

Using the feature vectors obtained for each śloka, we used various classifiers to classify each śloka as a synonym śloka or otherwise. We tried four practical scenarios of training and testing set choices as described in Table 11.

Scenario	Train Size	Test Size	P	P'	TP	Accuracy	Precision	Recall	F1
S1	52	209	84	56	42	0.73	0.75	0.50	0.60
S2	26	105	44	43	31	0.76	0.72	0.71	0.71
S3	261	131	54	45	36	0.79	0.80	0.67	0.73
S4	131	261	90	99	66	0.78	0.67	0.73	0.70

Table 12: Performance of classifiers in identifying synonym śloka.

False Positives (9)	False Negatives (18)
कामरूपोद्भवा कृष्णा नैपाली नीलवर्णयुक् काश्मीरी कपिलच्छाया कस्तूरी त्रिविधा स्मृता ॥ ६ ॥	श्रीखण्डं चन्दनं न स्त्री भद्र श्रीस्तैलपर्णिकः गन्धसारो मलजयस्तथा चन्द्र द्युतिश्च सः ॥ ११ ॥
महिषाक्षो महानीलः कुमुदः पद्म इत्यपि हिरण्यः पञ्चमो ज्ञेयो गुग्गुलोः पञ्च जातयः ॥ ३३ ॥	भद्र मुस्तञ्च गुन्द्रा च तथा नागरमुस्तकः मुस्तं कटु हिमं ग्राहि तिक्तं दीपनपाचनम् ॥ १३ ॥

Table 13: Examples of errors in classification (scenario S3).

The size of training sets were chosen to be smaller than those of test sets to resemble the real-world scenario where the ground truth can be created for only a small portion of the text, and predictions are needed to be made on the rest.

Table 12 shows the performance of the best classifier under various scenarios in identifying the śloka containing synonyms.

Table 13 shows some examples of wrongly classified śloka for the best performing scenario S3.

6.5.2 Synonym Identification

We next evaluate the performance of finding synonymous pairs from a synonym śloka. Table 14 shows the performance in identifying groups of synonymous substances. We say that a group of substances is *covered* even if a single pair in the group is identified. The result shows that even this has a scope for improvement.

Table 15 shows an example of a synonym śloka where none of the pairs are extracted correctly. The correct synonyms are चन्द्रिका, चर्महञ्जी, पशुमेहनकारिका, नन्दिनी, कारवी, भद्रा, वासपुष्पा, सुवासरा. We find the pairs (कारिका, हन्तु), (कारिका, भद्र), (कारिका, सपुष्प), (नन्दिन, रवि), (भद्र, हन्तु), (भद्र, सपुष्प), (सपुष्प, हन्तु), none of which are correct. The reasons for the errors are shown in Table 16. Almost all the nouns are analysed incorrectly, resulting in the group being completely missed.

In addition to the errors discussed in Section 6.4.1, an additional error occurs here, that of **TextError**. This refers to an error in the text corpus that we are working with. In particular, the original śloka contains the word चन्द्रिका while the corpus we are working with, has that word split as चन्द्रि and का, which results in this word not being analysed correctly. After correcting this error manually, we now obtain a valid pair (चन्द्रिका, भद्रा), thus covering this group.

We next analyse the finer errors that occur when some members of a synonymous group are identified correctly, but not all. Table 17 shows the performance.

Table 18 shows a synonym śloka from adhyāya 1 (हरीतक्यादिवर्गः).

This śloka contains a total of 11 synonyms. We find pairs of synonyms involving 9 out of

Synonym śloka	Groups present	Groups found	Group coverage
adhyāya 1	90	87	60
adhyāya 2	54	53	39

Table 14: Group coverage in synonym pair identification.

Synonym śloka	sandhi-samāsa split
चन्द्रि का चर्महन्त्री च पशुमेहनकारिका। नन्दिनी कारवी भद्रा वासपुष्पा सुवासरा ॥९६॥	चन्द्रि का चर्महन्त्री च पशुमेहन-कारिका। नन्दिनी कारवी भद्रा वासपुष्पा सु-वासराः ॥९६॥

Table 15: śloka 96 from adhyāya 1 of bhāvaprakāśanighaṇṭu and its sandhi-samāsa split.

Word	Root	Analysis	Is-Noun	Is-Verb	Error
चन्द्रि	चन्द्रि	['?']	False	False	TextError
का	किम्	['nom.', 'sg.', 'f.']	False	False	TextError
चर्म	चर्मन्	['acc.', 'sg.', 'n.']	True	False	OversplitError
हन्त्री	हन्त्	['nom.', 'sg.', 'f.']	True	False	OversplitError
च	च	['conj.']	False	False	
पशुमेहन	पशुमेहन	['voc.', 'sg.', 'n.']	True	False	OversplitError
कारिका	कारिका	['nom.', 'sg.', 'f.']	True	False	OversplitError
नन्दिनी	नन्दिन्	['acc.', 'du.', 'n.']	True	False	AnalysisError
का	किम्	['nom.', 'sg.', 'f.']	False	False	OversplitError
रवी	रवि	['acc.', 'du.', 'm.']	True	False	OversplitError
भद्रा	भद्र	['nom.', 'sg.', 'f.']	True	False	
वा	वा	['conj.']	False	False	OversplitError
सपुष्पा	सपुष्प	['nom.', 'sg.', 'f.']	True	False	OversplitError
सु	सु	['?']	False	False	OversplitError
वासराः	वासर	['voc.', 'pl.', 'm.']	True	False	OversplitError

Table 16: Analysis of śloka 96.

these, synonym pairs involving 8 of which are correct. We show examples of some of the false negatives and false positives among the pairs of synonyms identified.

- **False Positive:** (अमृता, अवी)

The word अव्यथा is split wrongly as अवी and अथा, and are then analysed separately. This results in both अमृता and अवी being in the same case (प्रथमा) and same number (एकवचन), thus getting wrongly marked as a synonymous pair.

- **False Negative:** (अभया, अमृता)

The word अभया gets analysed as instrumental (तृतीया) case of अभा instead of nominative (प्रथमा) case of अभया. This results in a case mismatch with अमृता and the pair is not extracted as a synonymous pair.

7 Conclusions and Future Work

In this paper, we have designed a framework to build a knowledge graph (KG) directly from saṃskṛta texts, and use it for question-answering in saṃskṛta. Our framework has multiple components and is based on rules and heuristics developed using the knowledge of grammar of saṃskṛta language and structure of the text.

However, for almost all the components, the accuracy can be improved. Improvements on any of these components by us or by others will make the system better. In future, we would like to work on improving the modules in a systematic manner. The biggest source of improvement can possibly come from a better word analyser. Usage of dictionaries, thesauri (such as amarakośa) and Sanskrit WordNet will be explored to see if they can help in understanding the structure of a word better. Crowd sourcing tools as well as human experts can also help refine some of

	śloka	Synonym śloka	P	P'	TP	Precision	Recall	F1
adhyāya 1	231	90	534	562	369	0.66	0.69	0.67
adhyāya 2	161	54	300	348	214	0.62	0.71	0.66

Table 17: Performance of finding synonym pairs.

Synonym śloka	sandhi-samāsa split	P	P'	TP
हरीतक्यभया पथ्या कायस्था पूतनाऽमृता हैमवत्यव्यथा चापि चेतकी श्रेयसी शिवाः ॥ ६ ॥	हरीतकी-अभया पथ्या कायस्था पूतना-अमृता हैमवती-अव्यथा च-अपि चेतकी श्रेयसी शिवाः ॥ ६ ॥	11	9	8

Table 18: Example of wrong pairs from adhyāya 1 of bhāvaprakāśa nighaṇṭu.

the steps. We would also like to expand the question-answering framework to work with longer questions that are not necessarily of the type factoid.

To conclude, we hope that this effort serves as a step towards the ultimate aim of automatically building a full-fledged knowledge graph from a saṃskṛta corpus.

Acknowledgements

We thank Dr. Sai Susarla, Dean at Maharshi Veda Vyas MIT School of Vedic Sciences, Pune, India, and his team, for sharing their expertise in āyurveda with us. We thank Shubhangi Agarwal and Rujuta Pimprikar for the help in creating ground truth as well as providing valuable feedback from time to time. We thank Dr. Kripabandhu Ghosh and Garima Gaur for the discussions and valuable feedback. We thank our saṃskṛta teacher Pralay Manna for enabling us in understanding the language better. We also thank the anonymous reviewers for their comments and suggestions.

References

- Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. 2007. DBpedia: A nucleus for a web of open data. In *The Semantic Web*, pages 722–735.
- Mira Bhargava and Joachim Lambek. 1992. A production grammar for Sanskrit kinship terminology. *Theoretical Linguistics*, 18(1):45–60.
- Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: A collaboratively created graph database for structuring human knowledge. In *ACM SIGMOD International Conference on Management of data*, pages 1247–1250.
- Pawan Goyal, Gérard Huet, Amba Kulkarni, Peter Scharf, and Ralph Bunker. 2012. A distributed platform for Sanskrit processing. In *24th International Conference on Computational Linguistics (COLING)*.
- Oliver Hellwig and Sebastian Nehrlich. 2018. Sanskrit word segmentation using character-level recurrent and convolutional neural networks. In *Conference on Empirical Methods in Natural Language Processing*, pages 2754–2763.
- Malhar Kulkarni, Chaitali Dangarikar, Irawati Kulkarni, Abhishek Nanda, and Pushpak Bhattacharyya. 2010. Introducing Sanskrit WordNet. In *5th Global Wordnet Conference (GWC 2010)*, pages 287–294.
- Ora Lassila, Ralph R Swick, et al. 1998. Resource Description Framework (RDF) model and syntax specification.
- Sivaja S Nair and Amba Kulkarni. 2010. The knowledge structure in Amarakośa. In *International Sanskrit Computational Linguistics Symposium*, pages 173–189.
- Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. 2007. YAGO: A core of semantic knowledge. In *16th International Conference on World Wide Web*, pages 697–706.

Introduction to Sanskrit Shabdmitra: An Educational Application of Sanskrit Wordnet

Malhar Kulkarni, Nilesh Joshi, Sayali Khare, Hanumant Redkar, Pushpak Bhattacharyya
Center for Indian Language Technology
Indian Institute of Technology Bombay
malharku@gmail.com, joshinilesh60@gmail.com, sayali.khare92@gmail.com,
hanumantredkar@gmail.com, pushpakbh@gmail.com

Abstract

This paper introduces a digital tool, viz., Sanskrit Shabdmitra, for learning and teaching of Sanskrit in active classroom environment as well as in other formal and informal set-up. It is based on an existing digital resource called Sanskrit Wordnet created at IIT Bombay. Thus, this paper also describes a direct application of Sanskrit Wordnet in particular, and Wordnet in general in the education domain. It describes the structure and various features of Sanskrit Shabdmitra.

1 Introduction

Sanskrit Wordnet¹ (SWN) was created at IIT Bombay as a major and unique lexical resource of Sanskrit (Kulkarni et al., 2010a). Kulkarni (2017) describes this effort in detail and demonstrates the contribution it made to the digital lexical resources of Indian languages. The effort of enriching SWN continues and scholars have tried to study it from the point of view of various natural language processing (NLP) tasks. Bhingardive et. al. (2014) developed well researched method based on SWN to populate one Wordnet using another lexical resource and Redkar et. al. (2016) have developed tool to populate one synset using two synsets with the help of SWN. Wordnet² and IndoWordNet³ has been used at various NLP tasks and applications. One such application is 'IndoWordNet::Similarity' developed by Bhingardive et. al. (2016) which measures semantic similarity and relatedness between two synsets in IndoWordNet. Similarly, IndoWordNet has been used for tasks such as Word Sense Disambiguation (Bhingardive and Bhattacharyya, 2017) for finding the most frequent sense using word and sense embeddings. This justifies the importance of IndoWordNet for word sense disambiguation for Indian languages. Similar to this, Sanskrit Wordnet can be used for the development of such tools, methods and utilities. Further, SWN can be helpful in explaining तत्सम (tatsama) and तद्भव (tadbhava) words which appear in any Indian languages. In this way Wordnet as a resource can be useful in many NLP tasks. Can Wordnet also be used as a base in creating an educational tool to teach and learn language? YES. We found that Wordnet can certainly be used as a base to create a tool to teach and learn Sanskrit. In this paper, in what follows, we elucidate how Sanskrit Wordnet can be used to develop educational application for teaching and learning Sanskrit language. Thus, a digital aid, Sanskrit Shabdmitra, has been introduced in this paper.

The paper is organized as follows - section 2 provides the literature survey; section 3 briefly mentions the related work; section 4 introduces the Sanskrit Shabdmitra, its structure, and its features in detail, explains how Shabdmitra enriches Wordnet, provides some applications; next section concludes the paper; this is followed by the future work.

¹<http://www.cfilt.iitb.ac.in/wordnet/webswn/wn.php>

²<https://wordnet.princeton.edu/>

³<http://www.cfilt.iitb.ac.in/indowordnet/>

2 Literature Survey

Sanskrit, belonging to the Indo-Aryan family of languages, is one of the ancient languages in the world. There is a rich tradition of developing a vast vocabulary in Sanskrit literature (Kulkarni et al., 2010a). Most of the languages in the Indo-European language family can be traced back to Sanskrit (Kulkarni et al., 2010b). There are various grammatical features and properties of Sanskrit which may not be present in other Indian languages (Redkar et al., 2014).

With the increase in the digital presence across the globe, content digitization and digital language learning have been growing enormously. Vocabulary is a crucial part of language learning. Learning Sanskrit vocabulary is one of the challenging tasks for any language learner. There are several applications and platforms available for curriculum based education, but very few are meant for language learning and active classroom. The Indian government is now supporting digital education and has taken several steps in digital language education. Following are government-driven platforms in digital language education:

- NCERT⁴ provides e-textbooks and supplementary books for students. It also provides guidelines for teachers for effective teaching.
- NROER⁵ is a Pan-Indian collaborative platform for teachers, students and professionals from various educational institutes. It allows uploading the digital content such as articles, text, poems, etc. which can be publicly available to the internet users.
- Swayam⁶ is another government designed program, collaborating with several government organisations, such as UGC⁷, AICTE⁸, NCERT, IGNOU⁹, etc. It covers courses from secondary education to post graduation. It teaches subjects like English, Hindi, and Sanskrit through video lectures and provides reading material, self-assessment tests, etc., and has an online discussion forum.

Apart from the above, there are some other non-government platforms engaged in digital language education. They are as follows:

- Openpathshala¹⁰ is an online platform for Sanskrit language teaching using lessons and video tutorials for learning Sanskrit grammar.
- pANini aShTaadhyaayii sUtra paaThaH¹¹ contains the audio pronunciation of the entire treatise on Sanskrit grammar (8 chapters of sūtras), called aṣṭādhyāyī by maharṣi pāṇini.
- shaale¹² provides the traditional methods of teaching Sanskrit using videos, live streaming (webcast), video on demand, audio documentation service, etc.
- Sanskrit Documents¹³ has the vast variety of documents which provides a collection of various links to various repositories useful for Sanskrit language learning.
- Vyoma¹⁴ introduces a guide of Sanskrit to generate a sentence, viz., Sanskrit vocabulary builder, Sanskrit pronunciation, Yogasutraparichaya, Saptāhastotra Saṅgrahaḥ, Sanskrit games, Learn Sanskrit through Hindi and English, etc.

⁴<http://ncert.nic.in/>

⁵<https://nroer.gov.in/>

⁶<https://swayam.gov.in/>

⁷<https://www.ugc.ac.in/>

⁸<https://www.aicte-india.org/>

⁹<http://www.ignou.ac.in/>

¹⁰<https://openpathshala.com/>

¹¹<http://surasa.net/music/samskrta-vani/ashtadhyayi.php>

¹²<https://www.shaale.com/>

¹³http://sanskritdocuments.org/learning_tools/index.php

¹⁴<http://www.sanskritfromhome.in/>

- learnsanskrit.org¹⁵ aims to teach Sanskrit grammar, providing a generative grammar guide of Sanskrit.
- Push to learn¹⁶ is a platform where students learn vocabulary from the school's course-books. However, this platform is not meant for Sanskrit.
- Spoken tutorial¹⁷ offers self-paced, multi-lingual courses. Anybody with a computer and a desire for learning can access this platform.
- Robomate¹⁸ is a curriculum based language learning app which has interactive study material for students like attractive video lessons.
- Byju's¹⁹ is a platform for interactive learning consisting of video lessons for Science, Maths, Economics and Business studies for school education. However, this platform does not have language learning facility.
- Duolingo for Schools²⁰ is a blended learning mate for the classrooms. Duolingo lessons provide personalized feedback to each student and help them to get the most out of classroom instruction. It also provides language specific class tips for teachers; such as phonetic inventory of a language, morphology, syntactic and semantic information. However, this tool does not facilitate Sanskrit language learning.

Other online resources for Sanskrit are bilingual dictionaries and thesauri which provide only the meanings of the words, such as Monier-Williams Dictionary²¹, Apte's Dictionary²², Spoken Sanskrit Dictionary²³, etc. Apart from these, there are some online dictionaries and thesauri in Sanskrit viz., Amarakosha²⁴, Sabda-kalpadruma²⁵, Vacaspatyam²⁶, etc. These online resources have domain-specific ontology, i.e., mythological ontology. Whereas, Wordnet does has been considered an upper ontology (Navigli and Velardi, 2004).

Most of these tools and platforms are in the form of text material, presentations, videos, lesson plan, etc. However, they do not provide relational semantics. Majority of them are not interactive and curriculum specific vocabulary learning is not available. It should be noted that one common thing among all the above resources is that they are more focused on individual learning and do not provide the active classroom learning. This is the desideratum as the knowledge of words or concepts in Sanskrit is not available as per the school curriculum. On the other hand, Sanskrit Shabdmitra, introduced here, is a digital language learning platform designed for Sanskrit vocabulary learning as per the school curriculum and for individual learning as well. This shall be explained in detail in section 4.

3 Related Work

Semantic relations of words helps in better understanding of new vocabulary (Lin, 1997). One such rich lexical resource based on semantic relations is viz., the Princeton WordNet²⁷, i.e., the WordNet(Miller, 1995), has been explored for vocabulary learning and other language learning applications (Hu et al., 1998; Sun et al., 2011; Brumbaugh, 2015; Hiray, 2015). Recently,

¹⁵<http://learnsanskrit.org>

¹⁶<http://pushtolearn.com/features>

¹⁷<https://spoken-tutorial.org>

¹⁸<https://roboestore.com/>

¹⁹<https://byjus.com/>

²⁰<https://schools.duolingo.com>

²¹<http://www.sanskrit-lexicon.uni-koeln.de/monier/>

²²<http://www.aa.tufs.ac.jp/~tjun/sktdic/>

²³<http://spokensanskrit.org/>

²⁴<https://sanskritdocuments.org/sanskrit/amarakosha/>

²⁵<http://www.sanskrit-lexicon.uni-koeln.de/scans/SKDSscan/2013/web/webtc2/index.php>

²⁶<https://archive.org/details/vacaspatyam02tarkuoft>

²⁷<https://wordnet.princeton.edu/>

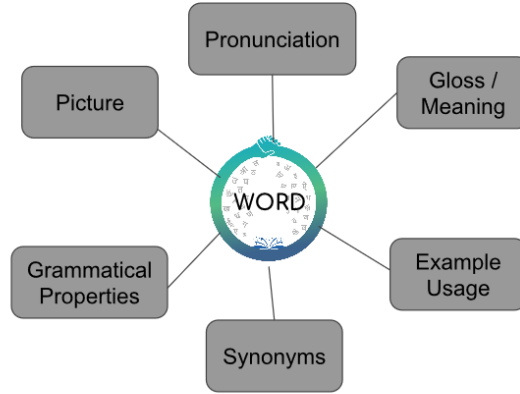


Figure 1: Shabdmitra as a friend of a word by providing word-meaning, example usage, pronunciation, picture, synonyms, other grammatical features, etc.

Hindi Wordnet²⁸ (HWN) has been used to build a teaching and learning digital aid, Hindi Shabdmitra, for Hindi language education in formal (schools) and informal (self-learning) setups (Redkar et al., 2017a). Additionally, the development of Marathi Shabdmitra, using Marathi Wordnet as a resource, is also under process.

A study of current digital resources used by the various educational institutions was also done as part of the background study. The outcome showed that there is a lack of quality resources which can cover all aspects of language learning such as grammar, concepts, usage, and pronunciations in an effective manner.

This motivated us to develop a digital aid, viz., Sanskrit Shabdmitra, that would fill this gap for Sanskrit language teaching and learning in both formal and informal learning environment.

4 Sanskrit Shabdmitra: an educational application using Sanskrit Wordnet

4.1 Shabdmitra

Shabdmitra is an umbrella of multilingual digital aid of language teaching and learning for Indian languages. It is built using IndoWordNet (Bhattacharyya, 2010) as a resource and is related to Hindi Shabdmitra (Redkar et al., 2017b), which is an initiative of IIT Bombay, India²⁹, exploring the applications of wordnet in education domain. The term Shabdmitra and its meanings were originally conceived by Malhar Kulkarni. The term Shabdmitra, शब्दमित्र is coined from two words 'shabda', शब्द, i.e., 'a word' and 'mitra', मित्र, i.e., 'a friend'; also means 'the Sun'. Therefore, Shabdmitra means a friend which helps in understanding a given word/concept. Using the second meaning of the word 'mitra' mentioned above, the word Shabdmitra would mean an illuminator of a word or concept. Thus, the function this tool aims to perform and the goal it wants to achieve is aptly expressed by the word 'Shabdmitra' itself. Thus, this term 'Shabdmitra' can be called self-explanatory. (anvartha-samjñā). This has been visualised in figures 1 and 2.

In Shabdmitra, the IndoWordNet data such as gloss, example sentence(s), synonyms and lexico-semantic relations are used and further augmented in order to cater to language learning needs. It is proposed to develop Shabdmitra for 18 Indian languages viz., Assamese, Bodo, Bengali, Gujarati, Hindi, Kannada, Kashmiri, Konkani, Manipuri, Malayalam, Marathi, Nepali, Odia, Punjabi, Tamil, Telugu, Urdu and Sanskrit, which are present in the IndoWordNet³⁰.

²⁸<http://www.cfilt.iitb.ac.in/wordnet/webhwn/wn.php>

²⁹<http://www.iitb.ac.in/>

³⁰<http://www.cfilt.iitb.ac.in/indowordnet/>



Figure 2: Shabdmitra as an illuminator for a word where it provides multiple senses, lexico-semantic and ontological relations, etc. of the same word

Figure 3 illustrates the IndoShabdmitra for IndoWordNet languages.

Shabdmitra is a multifaceted model which acts as a platform, as a resource and as a brand for the multilingual Indian scenario.

- As a Platform, various Indian languages which are present in IndoWordNet are made available at a single place.
- As a Resource, the multilingual Shabdmitra can be easily developed using the shared and not-shared data available in all the wordnets in the IndoWordNet database.
- As a Brand, all the wordnets can be branded under the umbrella of Shabdmitra which can be seen in figure 3.

Synset Category
Common
Uncommon
Common in Indian languages
Region and Language Specific

Table 1: Classification of Synsets by Bhattacharyya (2010)

4.2 Sanskrit Wordnet

Wordnet is a lexical resource composed of synsets, lexico-semantic relations and ontological information. Synset is the basic building block of a wordnet and it contains a gloss, an example sentence and synonyms. Wordnet is linked by semantic relations like hypernymy-hyponymy (is-a), meronymy-holonymy (part-of), troponymy (manner-of), etc. and by lexical relations like antonymy, gradation, etc. (Bhattacharyya, 2017). IndoWordNet is a linked structure of wordnets of 18 Indian languages from Indo-Aryan, Dravidian and Sino-Tibetan language families (Bhattacharyya, 2010).

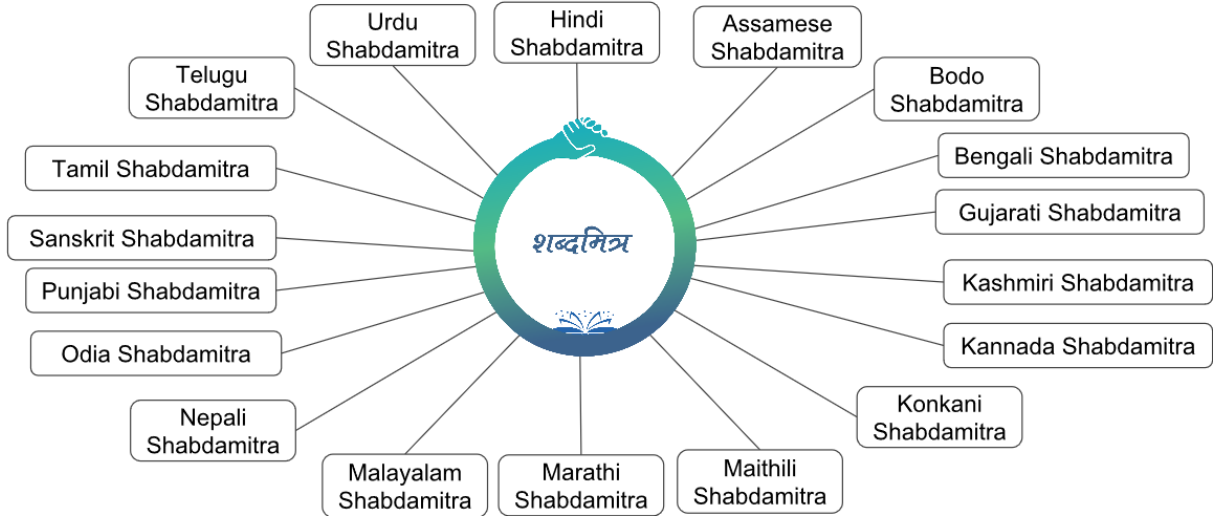


Figure 3: IndoShabdmitra: Shabdmitra of all IndoWordNet Languages

Sanskrit Wordnet is a part of IndoWordNet and is constructed using an expansion approach in which Hindi wordnet is used as a source (Kulkarni et al., 2010a). As Sanskrit has both the vedic as well as the modern literature, it has a greater scope of vocabulary than that of Hindi. Therefore, though the Sanskrit wordnet was built using an expansion approach from HWN, all Sanskrit synsets could not be developed. Hence, Sanskrit was developed in versions. And this development is an ongoing process. In this approach, the following part-of-speech wise method has been adapted for creating synsets (Kulkarni et al., 2010a):

Nouns

In the case of nouns in Sanskrit, a gender information is included in the word itself. In SWN, all nouns are stored in nominative singular form, however other Indian WordNets store nouns in their root forms. For example, देवः (devah), मतिः (matih), etc. are stored in nominative singular form in SWN; while in HWN, देव (deva), मति (mati), etc. will be stored in root form.

Adjectives

Adjectives in general have no gender of their own. However, in Sanskrit, they take the gender of the nouns which they qualify. Hence, in the synsets of adjectives, only the root forms are included. For example, भद्र (bhadra), निर्मल (nirmala) are stored in root form.

Adverbs

In SWN, adverbs are in their root form, however, it is observed that some of the adverbs have case-ending suffixes. These suffixes indicate the closed form of the word in that particular case-ending. Hence, these adverbs are regarded as frozen adverbs. In such cases, adverbs are stored with their case-ending suffixes. For example, व्यतिरेकेण (vyatirekeṇa) is stored as instrumental singular form; रहसि (rahasi) is stored as locative singular form. While, साधु (sādhu) is stored in root form.

Verbs

In SWN, verbs are also stored in their root form. For example, भू (bhū); कृ (kṛ); are stored in root form.

Apart from the above parts-of-speech information, other information such as gloss, examples are stored in the SWN database. Synsets in wordnet are interlinked by means of conceptual semantic and lexical relations. A combination of the glosses given in traditional dictionaries like Shabdakalpadruma³¹, vācyaspatyam³² and the translation of the gloss of the HWN synset is used to create SWN gloss for nouns, adjectives and adverbs. In the case of verbs, though these traditional Sanskrit dictionaries contain etymology based glosses, they are not appropriate for verbs which has ontology based wordnet structure. Hence, Navyanyāya terminology has been adapted for verbal glosses to construct synsets (Kulkarni et al., 2010b).

All these data, features and properties of SWN can be effectively used and utilised for teaching and learning Sanskrit. This is the base for building Sanskrit Shabdmitra which uses SWN data for language education purpose. This has been explored in detail in the next section.

4.3 Sanskrit Shabdmitra: Structure, Features and Applications

Sanskrit Shabdmitra (संस्कृत शब्दमित्र) is a digital language teaching and learning tool for Sanskrit language education. It uses Sanskrit Wordnet (SWN) as a resource. SWN was originally developed for the research purpose in the area of natural language processing. Soon, it was realized that this rich resource can be applied and used in developing educational applications. Sanskrit Shabdmitra is one such application of SWN.

Shabdmitra has been devised by taking into consideration the various stakeholders of this application. The major stakeholders of Sanskrit Shabdmitra are: Teachers, Students and Parents. Teachers' concern is that he/she should be able to convey the entire content to students in all the possible nuances and make them competent in language learning, and prepare them for examinations. Students' concern is that he/she should learn and understand the content as exhaustively as possible in all nuances and grow in terms of competence and be prepared for examinations. Parents' concern is that their child should get quality education and obtain competitive results. All these stakeholders and their concerns were considered while designing this digital aid.

4.3.1 Structure

In this tool, SWN data, features and properties are further augmented, simplified and presented in the form of educational application in order to cater to language teaching and learning requirements of Sanskrit language education.

SWN data such as gloss, example(s), synonyms, ontological information, lexico-semantic relations, etc. forms the content of Sanskrit Shabdmitra. Some of this information is customized and modified as per the language learning requirement and the learning levels of the individuals. Apart from this, Sanskrit Shabdmitra has various other features which are stored in the Shabdmitra database. The details of these features are presented in the next section.

Broadly, Sanskrit Shabdmitra has two types of interfaces, viz., Class-Wise and Level-Wise. Following sections elaborate on the same:

Class-wise interface

Class-Wise interface is designed specifically for classroom or formal setup wherein Sanskrit teacher uses this digital aid. Here, the data is presented in the interface, lesson by lesson. In this interface, the teacher chooses a school curriculum board (CBSE, ICSE, State Board, etc.); followed by a class to which he/she wants to teach; followed by a lesson/chapter. Once he/she clicks on a chapter, all the words from that chapter appear in the order in which they appeared in the textbook. While teaching, teacher can simply click on any of the word from the list and the word-specific information with the same sense is displayed accordingly in the interface. In most schools in India, Sanskrit is considered as second or third language. Hence, students have Sanskrit as a subject in the secondary. Therefore, the provision is made to include Sanskrit as

³¹<https://www.sanskrit-lexicon.uni-koeln.de/scans/SKDSscan/2013/web/webtc2/index.php>

³²<https://www.sanskrit-lexicon.uni-koeln.de/scans/VCPScan/2013/web/webtc2/index.php>

a 2nd or 3rd language in the school setup. Figure 5 shows the class-wise interface of Sanskrit Shabdmitra.

Level-wise interface

Level-Wise interface is designed for non-formal setup where any individual can learn Sanskrit depending upon his/her prior knowledge and language acquisition capabilities. In this scenario, Sanskrit Shabdmitra is focused on self-learning, which is as per the convenience of an individual. However, we should take into the account the nature of mother tongue (L1) and second language (L2) acquisition. Figure 6 shows an interface of Sanskrit Shabdmitra.

The level-wise interface is a big challenge as very few people have Sanskrit as their mother tongue. The majority of people study Sanskrit as their second or even third language. Hence, the levels are determined according to the knowledge of an individual. In order to get a better idea of L1 acquisition, researchers have tried to explain how children progress from “no language” or “blank slate” to their mother tongue. Whereas, for L2 acquisition, the process is more complicated as learners already have the knowledge of their mother tongue (Ipek, 2009).

Hence, the level-wise interface is different for first and second language learners. Taking the above scenario into the consideration and taking help from the National Curriculum Framework (NCF)³³ devised by NCERT - Government of India, and Common European Framework of Reference (CEFR)³⁴ by the Council of Europe the following levels for Sanskrit Shabdmitra are determined:

- Novice प्रारम्भिकः (prārambhikah) -
Novice is considered as a basic user where he/she is provided with the basics/fundamentals of language, like, varṇamālā (i.e., Sanskrit alphabet), word formation, etc.
- Intermediate माध्यमिकः (mādhyamikah) -
Intermediate is an independent user who has mastered the basics of Sanskrit and can communicate simple and basic needs. Here, most frequent words are provided.
- Advanced प्रवीणः (pravīṇah) -
Advanced is a proficient user. Here concept meaning with grammatical information is provided.
- Superior विशेषज्ञः (viśeṣajñah) -
Superior is a well versed language user. Here, multiple senses along with their grammatical and lexico-semantic features are provided.

Figure 4 depicts the levels of Sanskrit Shabdmitra.

4.3.2 Features

Sanskrit Shabdmitra has numerous features. Keeping standardization and language education need as a focus, features of Sanskrit Shabdmitra have been designed. In Sanskrit Shabdmitra, there are tool specific features and lexico-semantic features. Tool specific features are designed considering the usability and accessibility of the tool while teaching and learning Sanskrit. Lexico-semantic features are features which are specific to the word in picture. Lexico-semantic features are given in tables 2 and 3, there are two wide sections of features, viz., ‘Derived features’, which are derived from Sanskrit Wordnet and ‘Advanced features’, which are additional features specially designed considering the properties of Sanskrit language along with the interest of various stakeholders of this digital aid. Sanskrit Wordnet does not provide

³³<http://www.ncert.nic.in/rightside/links/pdf/framework/english/nf2005.pdf>

³⁴<https://www.babbel.com/en/magazine/how-and-why-to-determine-language-proficiency/>

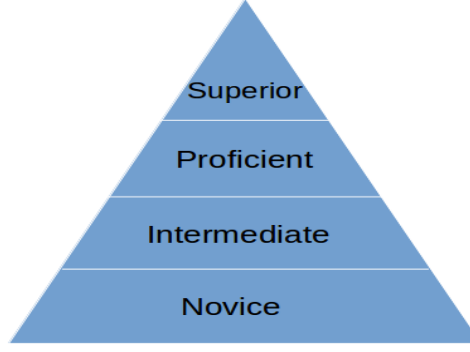


Figure 4: User levels of Sanskrit Shabdmitra

संस्कृत शब्दमित्र

CBSE

Class 6

Lesson 2

अपरः

दृष्टया

खादित्यामि

अन्यः

अकथयत्

गृहं

नीत्वा

भक्षयिष्यामि

इति

बादल

गृहम्



नाम

प्रातिपदिकम् - गृह

परिभाषा - मनुष्यैः इष्टिकादिभिः विनिर्मितं वासस्थानम् ।

प्रयोगः उद्धरणं वा - गृहिण्या एव गृहं शोभते ।

लिङ्गम् - नपुंसकलिङ्गम्

समानार्थी शब्दः - गेहम्, सदनम्, भवनम्, आलयः

विभक्तिः - द्वितीया

वचनम् - एकवचनम्

Figure 5: Class-Wise Interface for Sanskrit Shabdmitra

morphological features, however, Sanskrit Shabdmitra provides them. Table 2 shows the Derived features and Table 3 shows the Advanced features of Sanskrit Shabdmitra. These features rendered along with input word (search word) in interface of the Sanskrit Shabdmitra. Following are the details of these features.

Tool Specific Features

- **Standardization:** Standardization is a unique feature of Shabdmitra wherein all Shabdmitra of all Indian languages are interlinked. This inter-linkage is established using a unique identifier of a synset, called as a synset id. This feature has been inherited from IndoWordNet in which different wordnets are interlinked on the basis of synset id. Hence someone who is learning Hindi can see Sanskrit word for the same concept. Similarly, common Sanskrit words in Hindi for e.g., animals, numbers, flowers, body parts, etc. are unique across all the languages. This way we can attain standardization. Under standardization, we can separate synsets as per the classification of synsets as shown in Table 1; Similarly, illustrations can be shared across all the Indian languages.
- **Varṇamālā:** Sanskrit varṇamālā (alphabets) in Devanagari form is made available in the interface. Here, each of the letter of varṇamālā is displayed in animated form. This can help a learner in understanding the pattern of alphabet writing. Also, pronunciation of the

संस्कृत शब्दमित्र

घर

Levels

प्रारम्भिक: (Novice)

> माध्यमिक: (Intermediate)

प्रवीण: (Advanced)

विशेषज्ञ: (Superior)

अक्ष:



नाम

प्रातिपदिकम् - अक्ष

परिभाषा - आयताकृतिघनः येन द्यूतकाराः दीव्यन्ति।

प्रयोगः उद्धरणं वा - मोहनः अक्षैर्द्वने पटुः ।

लिङ्गम् - पुल्लिङ्गम्

समानार्थी शब्दः - पाशः , पाशकः

Figure 6: Level-Wise Interface for Sanskrit Shabdmitra

same is provided separately in the interface.

- Picture depiction: In Sanskrit Wordnet, there are several concepts which are difficult to explain using the gloss itself. For example, the concept of चषकः (caṣakaḥ, a glass) in Sanskrit is explained as - कषायादिपानार्थम् उपयुक्तं मृद्धात्वादिभिः विनिर्मितं पात्रम्। (kaṣāyādipānārtham upayuktaṁ mṛd-dhātṛvādibhiḥ vinirmitaṁ pātram, a container for holding liquids while drinking).



Figure 7: Picture depicting the concept चषकः (caṣakaḥ, a glass)

This gloss seems to be difficult for lower level learners to understand the concept due to the presence of some difficult words. However, as shown in figure 7, this can be easily understood with the help of a picture. Hence, pictures and illustrations help in differentiating the fine-grained senses found in Wordnet.

- Audio pronunciation: Shabdmitra interface has two types of audio pronunciation viz., मन्दम् (mandam, slow) and सामान्यम् (sāmānyam, normal). The slow-paced pronunciation provides the syllable-based output wherein each syllable is pronounced slowly, one at a time. This helps in understanding the sound structure of a syllable. Whereas for the normal

DERIVED FEATURES	
1.	Word (in a synset form):
2.	Original Gloss (परिभाषा)
3.	Original Example (वाक्ये प्रयोगः उद्धरणं वा):
4.	Gender (लिङ्गम्)
5.	Synonyms (समानार्थी शब्दः)
6.	Antonyms (विरुद्धार्थी शब्दः)
7.	Holonymy (अवयवी)
8.	Meronymy (अवयवः)
9.	Hypernymy (पराजातिः)
10.	Hyponymy (अपराजातिः)

Table 2: Derived Features of Sanskrit Shabdmitra

paced pronunciation, the words are pronounced at a normal pace. These audio features provided with Shabdmitra help in understanding the pronunciation and getting audio clarity of a word.

Derived Features

- Word (in a synset form) - The word which is stored and available in Sanskrit Wordnet synset is shown in this field.
- Original Gloss (परिभाषा) - If gloss is simple enough to understand then the original Sanskrit Wordnet gloss having same sense and synset id is kept as it is and rendered in this field, else a simplified gloss is rendered (This has been explained in the section 'Simplified Gloss' below).
- Original Example (वाक्ये प्रयोगः उद्धरणं वा) - Similarly, by default the original example sentence is retained.
- Gender (लिङ्गम्) - A gender of the word is directly taken from the Sanskrit Wordnet database.
- Synonyms (समानार्थी शब्दः) - Most frequent synonymous words of input word are displayed here. Right now the tool allows to display maximum of 5 words.
- Antonyms (विरुद्धार्थी शब्दः) - Antonyms of input word are displayed in this field.
- Holonymy (अवयवी) - A semantic relation that holds between a whole and its parts.
- Meronymy (अवयवः) - Relation between lexical units where the objects, etc., denoted by one are parts of those denoted by other.
- Hypernymy (पराजातिः) - A semantic relation between two synsets to capture super-set hood.
- Hyponymy (अपराजातिः) - A semantic relation between two synsets to capture sub-set hood.

Advanced Features

- Word (inflected form) - This particular feature is specific to a class-wise interface wherein an input word (i.e., word appeared in the textbook) which is having an inflected form is displayed.
- Word [in root form] (प्रातिपदिकम्) - This is applicable only to nouns which are in nominative singular form. Here, root word of the noun is displayed.

ADVANCED FEATURES	
1.	Word (inflected form)
2.	Word [in root form] (प्रातिपदिकम्)
3.	Simplified Gloss (परिभाषा)
4.	Simplified Example (वाक्ये प्रयोगः उद्धरणं वा)
5.	Type of Noun (संज्ञायाः प्रकारः)
6.	Type of Adjective (विशेषणस्य प्रकारः)
7.	Type of Verb (क्रियायाः प्रकारः)
8.	Type of Adverb (क्रियाविशेषणस्य प्रकारः)
9.	Case (विभक्तिः)
10.	Lakāra (लकारः)
11.	Person (पुरुषः)
12.	Number (वचनम्)
13.	Affix, Suffix (प्रत्ययः)
14.	Preposition, Prefix (उपसर्गः)
15.	Accent (स्वरः)
16.	Dhātuprakāraḥ (धातुप्रकारः)
17.	Gaṇaḥ (गणः)
18.	Padam (पदम्)
19.	With Augment 'इ'
20.	Transitivity (कर्मकत्वम्)

Table 3: Advanced Features of Sanskrit Shabdmitra (feature numbers 1, 2, and 9 to 19 are morphological features)

- Simplified Gloss (परिभाषा) - Concepts which are difficult to understand are simplified.
For example, in SWN for a word 'अक्षः' (akṣaḥ) the original gloss is 'काष्ठस्य वा अस्थिनः आयताकृतिघनः येन द्यूतकाराः दीव्यन्ति' (kāṣṭhasya vā asthinaḥ āyataakṛtighanaḥ yena dyūtakārah dīvyanti, a cubical shaped piece made of wood or bone used by gamblers for playing). Such a gloss, being too elaborate and difficult to follow at the beginner's level, has been simplified to: आयताकृतिघनः येन द्यूतकाराः दीव्यन्ति (āyataakṛtighanaḥ yena dyūtakārah dīvyanti, a cubical shaped piece used by gamblers for playing).
- Simplified Example (वाक्ये प्रयोगः उद्धरणं वा) - Similarly, examples are simplified.
- Type of Noun (संज्ञायाः प्रकारः) - If the input word is a noun then it is assigned with the prescribed types of nouns. This information is usually taken from ontological database of IndoWordNet.
- Type of Adjective (विशेषणस्य प्रकारः) - If the input word is an adjective then it is assigned with the prescribed types of adjectives. This information is usually taken from ontological database of IndoWordNet.
- Type of Verb (क्रियायाः प्रकारः) - If the input word is a verb then it is assigned with the prescribed types of verbs. This information is usually taken from ontological database of IndoWordNet.
- Type of Adverb (क्रियाविशेषणस्य प्रकारः) - If the input word is an adverb then it is assigned with the prescribed types of adverbs. This information is usually taken from ontological database of IndoWordNet.

- Countability (गणनीयता) - Nouns can be either countable or uncountable. Accordingly, the countability is assigned to the nouns. Countable nouns are those that refer to something that can be counted. On the other hand nouns which do not typically refer to things that can be counted, are Uncountable nouns³⁵.
- Case (विभक्तिः) - The input word can belong to any of the eight cases. They are listed as below:
 - Nominative - प्रथमा (prathamā)
 - Accusative - द्वितीया (dvitīyā)
 - Instrumental - तृतीया (tṛtīyā)
 - Dative - चतुर्थी (caturthī)
 - Ablative - पञ्चमी (pañcamī)
 - Genitive - षष्ठी (ṣaṣṭhī)
 - Locative - सप्तमी (saptamī)
 - Vocative - संबोधन (sambodhana)
- Lakāra (लकारः) - This is verb specific property of a word which helps in identifying the tense, aspect and modality of a word. The input word can belong to any of the 10 types of lakāra. They are listed as below:
 - laṭ - लट् (laṭ)
 - lañ - लङ् (lañ)
 - loṭ - लोट् (loṭ)
 - vidhiliñ - विधिलिङ् (vidhiliñ)
 - āśīrliñ - आशीर्लिङ् (āśīrliñ)
 - liṭ - लिट् (liṭ)
 - luṭ - लुट् (luṭ)
 - luñ - लुङ् (luñ)
 - lṛṭ - लृट् (lṛṭ)
 - lṛñ - लृङ् (lṛñ)
- Person (पुरुषः) - This is a verb specific property of a word wherein the verb can appear in the sense of person viz. the first (उत्तमः), second (मध्यमः) and third (प्रथमः) (Pāṇini and Vasu, 1962) [1.4.101]
- Number (वचनम्) - Inflectional category basically distinguishing reference to one individual from reference to more than one.
- Affix, Suffix (प्रत्ययः) - There are six main kinds of affixes given in Sanskrit grammar viz., सुप्, तिङ्, कृत्, तद्धित, धातुप्रत्ययः [(i.e. सन्, क्यप्, etc.)] and स्त्रीप्रत्ययः. Right now, in Sanskrit Shabdmitra only first 3 types of affixes i.e. सुप्, तिङ्, कृत् are shown.
- Preposition, Prefix (उपसर्गः) - The word उपसर्ग originally meant only 'a prefixed word'. These prefixes are always used along with a verb (Abhyāṅkara and Shukla, 1977) [pg 88]
- Accent (स्वरः) - This property is possessed only by vowels and not by consonants (Abhyāṅkara and Shukla, 1977) [pg 438]. Accents are basically found in vedic texts. Except traditional schools, vedic texts are not part of the school syllabus viz., CBSE, ICSE, etc. Hence, accents are not introduced in primary level of Sanskrit Shabdmitra. However, words with accents shall be introduced in advanced levels. Following are the types of accents:

³⁵<https://www.lexico.com>

- उदात्तः the acute accent defined by Panini (Pāṇini and Vasu, 1962) [1.2.29]. The acute is the prominent accent in a word (Abhyaṅkara and Shukla, 1977) [pg 81]. According to the position in the word, the acute accent has following sub-types:
 - * आद्युदात्तः a word beginning with an acute accent i.e. which has got the first vowel accented acute.
 - * मध्योदात्तः the acute accent to the middle vowel which is neither the initial nor the final.
 - * अन्तोदात्तः a word with its last vowel accented acute.
- अनुदात्तः the grave accent defined by Panini (Pāṇini and Vasu, 1962) [1.2.30].
- स्वरितः the circumflex accent defined by Panini (Pāṇini and Vasu, 1962) [1.2.31].
- Dhātuprakārah (धातुप्रकारः) - There are different types of root verb as follows:
 - औपदेशिकधातुः (पाणिनीयधातुपाठे उपदिष्टः) Panini has given a long list of roots under ten groups named as औपदेशिकधातुः or primary roots.
 - आदेशिकधातुः (सनाद्यन्तादिधातवः). There are two types of them, they are as follows:
 - * roots derived from roots. These are classified into three types:
 - causative (णिजन्त)
 - desiderative (सन्नन्त)
 - intensive (यङन्त)
 - * roots derived from nouns.
 - वैदिकधातुः roots found in vedic literature.
 - सौत्रधातुः roots mentioned specifically in paninian rule only.
- Gaṇah (गणः) - There is a long list of roots under the following ten groups. They are as follows:
 - भ्वादिगणः (bhvādigaṇah)
 - अदादिगणः (adādigaṇah)
 - जुहोत्यादिगणः (juhotyādigaṇah)
 - दिवादिगणः (divādigaṇah)
 - स्वादिगणः (svādigaṇah)
 - तुदादिगणः (tudādigaṇah)
 - रुधादिगणः (rudhādigaṇah)
 - तनादिगणः (tanādigaṇah)
 - क्र्यादिगणः (kryādigaṇah)
 - चुरादिगणः (curādigaṇah)
- Padam (पदम्) - A technical term for the affixes. There are three types of padam:
 - parasmaipadam परस्मैपदम् term used in grammar with reference to the personal affixes ति (ti), तः (ta), etc.
 - ātmanepadam आत्मनेपदम् a technical term for the affixes त (ta), आताम् (ātām), etc.
 - ubhayapadam उभयपदम् a technical term in which a specific group of verbs are from both parasmaipada and aatmanepada (Abhyaṅkara and Shukla, 1977) [pg 92]
- With Augment 'इ' - Here इ (i) is prefixed in the case of root.
 - अनिट् (aniT) roots अनिट् does not allow the augment इ to be prefixed.
 - सेट् roots सेट् always allows the augment इट् to be prefixed.
 - वेट् roots optionally admit the application of the augment इ.

CBSE Syllabus	All words	Unique words
Class VI	1499	813
Class VII	2655	1604
Class VIII	3072	1987
Class IX	2701	1814
Class X	2902	1989
Total	12829	8207
All class unique words		6784

Table 4: Sanskrit word-collection statistics

- Transitivity (कर्मकत्वम्) - karmakatvam can be one among the two as follows:
 - सकर्मकः sakarmakah, transitive
 - अकर्मकः akarmakah, intransitive

4.3.3 Shabdmitra Enriches Wordnet

It is noticed that the development of Sanskrit Shabdmitra leads to the enrichment of SWN. It is a two-way process in which SWN helps Sanskrit Shabdmitra by providing the resource, while Sanskrit Shabdmitra helps SWN by providing additional words and properties, hence enriching the same. Table 4 depicts the count of words which are collected and unique words from classes VI to X under the CBSE board.

4.3.4 Applications

There are various applications of Sanskrit Shabdmitra. Some of them are listed as below:

- Sanskrit Shabdmitra is an educational tool for teaching and learning Sanskrit vocabulary.
- It also acts as a teaching and learning aid for teachers in school setup.
- It can also be used for testing the Sanskrit language knowledge of an individual.
- This tool can be of great help for conducting and preparing Sanskrit competitive exams.
- It can be used to explain tatsama and tadbhava words in other languages.

5 Conclusion

In this paper, how Sanskrit Wordnet can be used for developing educational application has been explained. It is also demonstrated how a semantically rich lexical resource like Wordnet, originally developed for research purpose can be remodeled for practical usage in education domain.

Sanskrit Shabdmitra is one such comprehensive e-learning aid which helps in learning Sanskrit language, pronunciation, grammar and understanding the concepts through images, definition and examples. It caters to a wider range of audience ranging from school children to individual learners at different levels, i.e., from novice to the superior. The tool, Sanskrit Shabdmitra presented here is a multi-modal, multi-layered Sanskrit language teaching and learning aid which can be used for formal and informal learning environments. Further, Shabdmitra acts as a platform, as a resource as well as a brand. It helps in enriching the Sanskrit Wordnet and vice versa.

6 Future Work

In Future, we plan to incorporate question answering system which can help in understanding the knowledge of the user, also which can help in understanding the level at which he can start learning Sanskrit. Also, the tool will be improved with the inclusion of gamification, bilingual as well as multilingual learning and teaching under Shabdmitra platform.

Acknowledgements

We would like to thank the entire Hindi Shabdmitra Team; Center for Indian Language Technology (CFILT), and Indian Institute of Technology Bombay (IITB) for providing us necessary resource and support. We further thank TCTD, IIT Bombay³⁶ for providing the necessary support. Finally, we thank reviewers for their comments, positive remarks and encouragements.

References

- K.V. Abhyankara and J.M. Shukla. 1977. A Dictionary of Sanskrit Grammar. Number no. 134 in A dictionary of Sanskrit grammar. Oriental Institute.
- Pushpak Bhattacharyya. 2010. Indowordnet. In Proceedings of Lexical Resources Engineering Conference (LREC), Malta.
- Pushpak Bhattacharyya. 2017. Indowordnet. In *The WordNet in Indian Languages*, pages 1–18. Springer.
- Sudha Bhingardive and Pushpak Bhattacharyya. 2017. Word sense disambiguation using indowordnet. In *The WordNet in Indian Languages*, pages 243–260. Springer.
- Sudha Bhingardive, Tanuja Ajothkar, Irawati Kulkarni, Malhar Kulkarni, and Pushpak Bhattacharyya. 2014. Semi-automatic extension of sanskrit wordnet using bilingual dictionary. In Proceedings of the Seventh Global Wordnet Conference, pages 324–329.
- Sudha Bhingardive, Hanumant Redkar, Prateek Sappadla, Dharendra Singh, and Pushpak Bhattacharyya. 2016. Indowordnet:: Similarity computing semantic similarity and relatedness using indowordnet. In *Global WordNet Conference*, page 39.
- Heidi Brumbaugh. 2015. Self-assigned ranking of L2 vocabulary: using the Bricklayer computer game to assess depth of word knowledge. Ph.D. thesis, Arts & Social Sciences:.
- Amit C. Hiray. 2015. Teaching and Learning of EAP Vocabulary: A Web-based Integrative Approach at the Tertiary Level in India. Ph.D. thesis, Dept. of HSS, IIT Bombay.
- X Hu, AC Graesser, Tutoring Research Group, et al. 1998. Using wordnet and latent semantic analysis to evaluate the conversational contributions of learners in the tutorial dialog. In Proceedings of the international conference on computers in education, volume 2, pages 337–341.
- Hulya Ipek. 2009. Comparing and contrasting first and second language acquisition: Implications for language teachers. *English Language Teaching*, 2(2):155–163.
- Malhar Kulkarni, Chaitali Dangarikar, Irawati Kulkarni, Abhishek Nanda, and Pushpak Bhattacharyya. 2010a. Introducing sanskrit wordnet. In *Principles, Construction and Application of Multilingual WordNets*, Proceedings of the 5th GWC, edited by Pushpak Bhattacharyya, Christiane Fellbaum and Piek Vossen, Narosa, page 257–294. Narosa Publishing House, New Delhi.
- Malhar Kulkarni, Irawati Kulkarni, Chaitali Dangarikar, and Pushpak Bhattacharyya. 2010b. Gloss in sanskrit wordnet. In Proceedings of Sanskrit Computational Linguistics, pages 190–197. Berlin: Springer-Verlag / Heidelberg.
- Malhar Kulkarni. 2017. Sanskrit wordnet at indian institute of technology (iitb) mumbai. In *The WordNet in Indian Languages*, pages 231–241. Springer.

³⁶<http://www.tatacentre.iitb.ac.in/>

- Chih-Cheng Lin. 1997. Semantic network for vocabulary teaching. *Journal of National Taiwan Normal University*, (42):43–54.
- George A Miller. 1995. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41.
- Roberto Navigli and Paola Velardi. 2004. Learning domain ontologies from document warehouses and dedicated web sites. *Computational Linguistics*, 30(2):151–179.
- Pāṇini and Srisa Chandra Vasu. 1962. *The Aṣṭādhyāyī of Pāṇini*. Motilal Banarsidass.
- Hanumant Redkar, Jai Paranjape, Nilesh Joshi, Irawati Kulkarni, Malhar Kulkarni, and Pushpak Bhattacharyya. 2014. Introduction to synskarta: An online interface for synset creation with special reference to sanskrit. In 11th International Conference on Natural Language Processing (ICON-2014), Goa, India, page 229.
- Hanumant Redkar, Nilesh Joshi, Sandhya Singh, Irawati Kulkarni, Malhar Kulkarni, and Pushpak Bhattacharyya. 2016. Samāsa-kartā: An online tool for producing compound words using indowordnet. In 8th Global WordNet Conference.
- Hanumant Redkar, Sandhya Singh, Meenakshi Somasundaram, Dhara Gorasia, Malhar Kulkarni, and Pushpak Bhattacharyya. 2017a. Hindi shabdmitra: A wordnet based e-learning tool for language learning and teaching. In Proceedings of the 4th Workshop on Natural Language Processing Techniques for Educational Applications (NLPTEA 2017), pages 23–28, Taipei, Taiwan, December. Asian Federation of Natural Language Processing.
- Hanumant Redkar, Nilesh Joshi, Sayali Khare, Lata Popale, Malhar Kulkarni, and Pushpak Bhattacharyya. 2017b. Hindi shabdmitra: A wordnet based tool for enhancing teaching-learning process. In Proceedings of the 14th International Conference on Natural Language Processing (ICON 2017), Jadavpur University, Kolkata, India, December.
- Koun-Tem Sun, Huang Yueh-Min, and Liu Ming-Chi. 2011. A wordnet-based near-synonyms and similar-looking word learning system. *Journal of Educational Technology & Society*, 14(1):121.

Vaijayantīkośa Knowledge-Net

Aruna Vayuvegula*, Satish Kanugovi†, Sivaja S Nair‡, Shivani V§ and Mahalakshmi¶

Karnataka Samskrit University

Bangalore, India

33aruna@gmail.com, satishk.rao@gmail.com,

sivaja.s.nair@gmail.com, shivani.ksu@gmail.com and prasmx@gmail.com

Abstract

A kośa (lexicon) is a literary work that provides a comprehensive understanding of words by arranging them along with their synonyms and other words that are semantically related. Its format has been designed to include not just ontological classification, but to give a holistic idea of a concept represented by the word. This allows a thorough understanding of the words, and also the knowledge they embody. Vaijayantīkośa is a popular Sanskrit lexicon that contains words from spoken language as well those used in Vedic literature. To facilitate dissemination of this knowledge, a web-based tool, Vaijayantīkośa Knowledge Net, is created for easy access and analysis of the words in the kośa. The objective of the tool is to provide information to researchers from different fields of study to explore the knowledge contained in the kośa with the help of synsets and ontological structure.

Key words: Vaijayantīkośa, Synset, Ontology, KnowledgeNet, Semantic relations

1 Introduction

Sanskrit is rich with domain-specific and subject-specific kośa literature. They are written in verse format enabling them to be memorized easily by students. Generally kośa, in the Indian tradition of knowledge representation, is a grouping of words with semantic relations to provide comprehensive understanding of the word and its ontological classification. The ontological classification and knowledge structure in Sanskrit kośas have been described in detail by Kulkarni, A (2010).

Patkar (1981) in his book “History of Sanskrit Lexicography”, lists at least 81 lexicons that were written in Sanskrit between 400 BC and 1800 AD. Vogel (1975) in his work, ‘Indian Lexicography’ details the characterization of Indian lexica and lists down over forty unique dictionaries, many special, bilingual and multilingual dictionaries. Unfortunately, many of these works have been lost and we are left with very few of these treasures. Hence, there is a need to ensure that the existing lexicons are well-preserved for posterity and technology can be a great asset to achieve this goal.

Amarakośa is the most authoritative and ancient thesaurus of Sanskrit. There have been several commentaries and translations of the lexicon (Patkar, 1981, pp. 19-21) both in Indian as well as foreign languages. In recent times, it has also captured the interest of computational linguists. Nair, in her PhD thesis (Nair, 2011) has detailed the knowledge structure of Amarakośa and developed the tool, Amarakośa Knowledge Net (AKN), that systematically represents the links between words based on a structured table in a dynamic manner. She has suggested in her thesis that AKN can serve as a model for developing tools for other kośas.

As part of the Post-graduate diploma in Sanskrit computational linguistics program, we take this suggestion forward and develop the Vaijayantīkośa Knowledge Net (VKN) tool to capture

the knowledge structure of Vaijayantīkośa. The paper introduces Vaijayantīkośa and details the different aspects of the VKN development in the following sections.

2 Vaijayantīkośa (VK)

VK is written by Yādavaprakāśa between 10th and 11th century (Bühler, 1887). He lived in the southern part of India, near the present day Kanchipuram in Tamilnadu (Oppert, 1893, p. 2). VK not only has a rich vocabulary of words for common usage, it also has a large number of terms from the Vedas.

Though there are many manuscripts on VK, in different Indian languages, none of them is complete, except one manuscript in Malayalam language. (Oppert, 1893, pp. 3-4).

For the purpose of this work, we have referred to the following two texts of VK.

1. The Vaijayantī of Yādavaprakāśa compiled by Gustav Oppert
This version has introduction by Gustav in English and an elaborate section of vocabulary with meanings in English. Gustav has painstakingly referred to 11 manuscripts and consolidated all the kāṇḍas as one entity (Oppert, 1893).
2. Vaijayantīkośa compiled by Sri. Pandit Haragovindashastry.
This version has introduction by Pandit Haragovindashastry in Hindi and appears, to a large extent, based on Gustav Oppert's work itself. He gives a brief commentary on the uniqueness of the lexicon and adds glossary of words at the end with references to the ślokas where the words appear (Haragovindashastry, 1971).

Bühler (1887) gives an overview of VK, its structure and information about its author. Kulkarni refers to VK while giving an overview of lexicographic traditions in India and Sanskrit (Kulkarni, 2010). Kaur also touched upon VK through a taxonomical analysis of early Sanskrit literature (Kaur & Singh, 2018). Vogel touches upon VK while chronicling Indian Lexicography and gives brief details about the style and classification adopted by the author (Vogel, 1975). Some regional scholars have also referred to VK in their works. For example, Mallinatha, in *Amarapadapārijāta* (commentary on Amarakośa) provides close to 212 citations from VK (Nair, 2011).

For this project, the compilation of VK by Gustav Oppert has been taken because of the comprehensiveness of his work as well as the detailed vocabulary of words with meanings in English.

2.1 Structure of VK

The author, Yādavaprakāśa has arranged the words into kāṇḍas and adhyāyas based on a clear ontological structure. The kāṇḍas are named according to the major topic covered. For example, the antarīkṣakāṇḍa consists of all the words related to the sky, universe, astronomy, astrology etc.

Each kāṇḍa is further divided into adhyāyas with semantically related words, arranged together according to context, in the form of ślokas. The classification is detailed in the Figure 1.

1. VK consists of nearly 20000 entries of words listed in verse form.
2. It begins with a maṅgalaśloka followed by nine and a half verses of paribhāṣaślokas which provide pointers to decode the gender information of the words.

स्त्रीपुंनपुंसकं लिङ्गं सङ्कीर्णं तच्च पञ्चधा।

नृस्त्री नृषण्डषण्डस्त्री त्रिलिङ्गं वाच्यलिङ्गकम्॥१.१.३॥¹

समासे स्युः पृथक् सर्वे शब्दा बहुवचोऽन्तके॥१.१.५॥

More rules for interpreting the liṅga (gender) of the words are described in 58 śloka of Liṅgaśaṅgrahādhyāya (of Śeṣakāṇḍa).

3. There are two major divisions of the kośa - Paryāyabhāga (synonymous words) and Nānārthabhāga (polysemous words).
4. There are five kāṇḍas under Paryāyabhāga and three under Nānārthabhāga.
5. The kāṇḍas are further divided into adhyāyas; they are 43 in total.
6. The structure of VK is represented below (Figure 1).

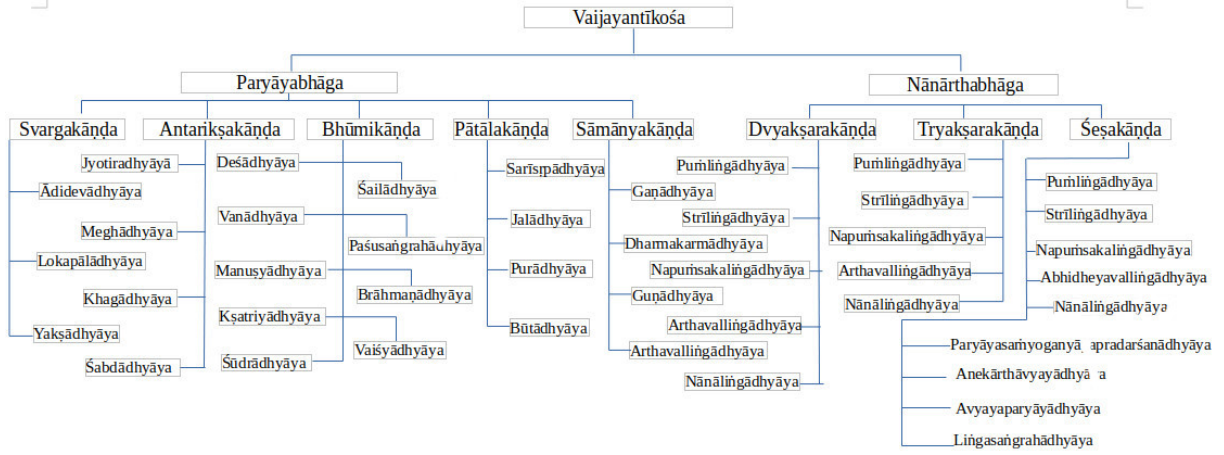


Figure 1: Classification of VK

7. Śloka in VK contain words, their synonyms and meanings. In some cases, probably where the author found it necessary, information pertaining to gender, brief description of the term may also be included.
8. VK emphasizes understanding a concept at greater depth and precision.

2.1.1 Semantic Arrangement of Words in VK

In VK, the kāṇḍas are arranged based on a particular theme. Kāṇḍas are further divided into adhyāyas which are based on sub-themes. Adhyāyas contain śloka that mostly follow semantic order with occasional violations. Śloka contain words that are related to a concept. A given word is typically followed with its synonyms and subsequently other relations, like पति-पत्नीभावः (husband – wife relation), जन्य-जनकभावः (child – parent relation), स्व-स्वामिभावः (owner – property relation), सेव्य-सेवकभावः (lord - servant relation), धर्म-धर्मिभावः (property - locus relation), गुण-गुणिभावः (quality - qualifier relation) etc. For example, in concept Viṣṇu, first 53 words form a synset. Subsequently, the author lists words that refer to powers of Viṣṇu. They are followed by possessions of Viṣṇu and so on. Nevertheless, there is a pattern that perhaps reflects the logic of the times it was written.

Given below is the example of the word how Viṣṇu is dealt in VK.

¹Śloka reference: The position of a śloka in the VK is represented numerically as x.y.z, where x=adhyāya number, y = kāṇḍa number in the adhyāya, and z = the śloka number in the kāṇḍa. For example in this śloka 1.1.3.

Example 1: Concept of विष्णु:

The śloka 1.1.10 to 1.1.38 from ādidevādhyāya of svargakāṇḍa describe the concept of Viṣṇu with different relations. See Figure 2.

field 1 : word in sanskrit, field 2 : English equivalent in (), field 3 : number in synset in (), field 4 : kāṇḍa.adhyāya.śloka in ()

- विष्णुः (epithet of Viṣṇu)(53)(1.1.10 - 1.1.15)
 वैष्णवी (power of Viṣṇu)(9)(1.1.16)
 कौस्तुभः (jewel of Viṣṇu)(1)(1.1.17)
 श्रीवत्सः (mark on Viṣṇu)(1)(1.1.17)
 नन्दकः (sword of Viṣṇu)(1)(1.1.17)
 शार्ङ्गः (bow of Viṣṇu)(1)(1.1.17)
 पाञ्चजन्यम् (conch of Viṣṇu)(1)(1.1.17)
 सुदर्शनम् (discus of Viṣṇu)(1)(1.1.17)
 कौमोदकी (mace of Viṣṇu)(1)(1.1.18)
 नरसिंहः (incarnation of Viṣṇu)(1)(1.1.18)
 वामनः (incarnation of Viṣṇu)(10)(1.1.19 - 1.1.20)
 परशुरामः (incarnation of Viṣṇu)(2)(1.1.20)
 श्रीरामः (incarnation of Viṣṇu)(15)(1.1.20 - 1.1.24)
 बलभद्रः (incarnation of Viṣṇu)(20)(1.1.22 - 1.1.24)
 संवर्तकम् (Plough of Balabhadra)(1)(1.1.24)
 सौनन्दनम् (pestle of Balabhadra)(1)(1.1.25)
 कृष्णः (incarnation of Viṣṇu)(10)(1.1.25 - 1.1.26)
 दारुकः (Charioteer of Kriṣṇa)(1)(1.1.26)
 वसुदेवः (father of Kriṣṇa)(3)(1.1.26)
 मन्मथः (god of love, son of Viṣṇu)(25)(1.1.27 - 1.1.29)
 अनिरुद्धः (son of Manmatha)(3)(1.1.29)
 नरनारायणः (incarnation of Viṣṇu)(2)(1.1.30)
 हयग्रीवः (incarnation of Viṣṇu)(2)(1.1.30)
 आदिशेषः (incarnation of Viṣṇu)(1)(1.1.30)
 व्यासः (incarnation of Viṣṇu)(6)(1.1.30)
 दत्तात्रेयः (incarnation of Viṣṇu)(1)(1.1.31)
 कल्किः (incarnation of Viṣṇu)(1)(1.1.31)
 कपिलः (incarnation of Viṣṇu)(3)(1.1.31)
 व्यासः (incarnation of Viṣṇu)(6)(1.1.31- 1.1.32)
 बुद्धः (incarnation of Viṣṇu)(32)(1.1.32 - 1.1.35)
 लक्ष्मीः (wife of Viṣṇu)(10)(1.1.36)
 गरुडः (vehicle of Viṣṇu)(12)(1.1.37 - 1.1.38)

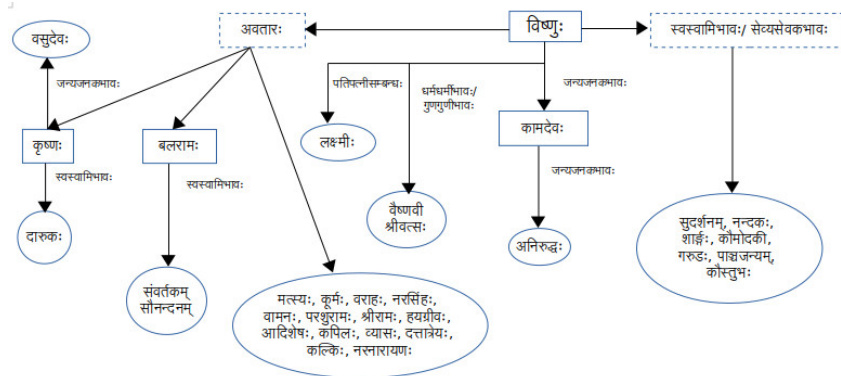


Figure 2: Relations of Viṣṇu

Example 2: Concept of कालः

In VK, the reference to kāla is from śloka 2.1.52 to 2.1.54, which is a total of 43 words in the jyotiradyāya of antarikṣakāṇḍa. The concept of kāla starts with the smallest unit of time which is referred to as तुटिः (moment). Subsequently, higher units of time are mentioned as depicted below:

कालः (time)(3)(2.1.52)

तुटिः (moment)(2)(2.1.52)

लघ्वक्षरकः (space of two moments)(1)(2.1.52)

अक्षरपातकः (space of two laghvakṣarakas)(1)(2.1.52)

निमेषः (space of two akṣarapātakas)(1)(2.1.53)

लिप्तिका (space of two nimeṣas)(1)(2.1.53)

काष्ठा (space of nine liptikās)(1)(2.1.53)

लवः (space of two kāṣṭhas)(1)(2.1.53)

कला (space of five lavas)(1)(2.1.53)

लेशः (space of twelve kalās)(1)(2.1.54)

क्षणः (space of 16 leśas)(1)(2.1.54)

नाडी (space of six kṣaṇas)(1)(2.1.54)

मुहूर्तः (space of twelve nādis)(1)(2.1.54)

घटिका (space equal to one muhūrta)(2)(2.1.54)

Here we can see the hierarchical order of the words which is connected through the relation अवयव-अवयविसम्बन्धः. Subsequent ślokas i.e. 2.1.55 to 2.1.73 also deal with the concept of kāla but has not been depicted here due to lack of space. A few observations on examining the concept of the word kāla are as follows:

- A very logical and precise structure of division of time has been adopted starting from the lowest measure of time.
- A very systematic division of time until it spans 24 hours or one day is seen. Then, there is the first violation of nesting where day is followed by night and the author goes on to describe night, different kinds of night. Within the nesting of night too, after describing different kinds of night, he suddenly introduces darkness and then goes on to describe different types of darkness.
- After this, there is the third violation of nesting when he goes back to day and then defines different parts of the day followed by different parts of night. Next, he picks terms that talk about space of three hours (which is relevant to both day and night), lucky portion of the day, dawn and twilight. He then ends by addressing a lunar day and different days in a lunar month.
- The list is followed by months, seasons, years, yugas etc.

The author often describes the qualities of a particular term. For example, under the main word 'sun', the term sunray is given. The author lists down 22 words under the concept of sunrays. These words do not appear to be synonymous but indicate a more complex idea that needs further research.

तासां शतानि चत्वारि रश्मीनां वृष्टिसर्जने।
शतत्रयं हिमोत्सर्गे तावत्घर्मस्य सर्जने॥ २.१.१७²

3 Vaijayantikośa Knowledge Net (VKN)

VKN is a web-based tool to access knowledge embodied in VK by providing comprehensive information related to the word including meanings, synonyms and relations with other words.

²This śloka is only a small extract of the group of verses that are referred under sunrays.

3.1 Scope of the present project

VK is a voluminous lexicon with approximately 20,000 entries of words. However, for developing this version of the web-tool, the first two kāṇḍas mainly the svargakāṇḍa and antarīkṣakāṇḍa have been taken, which contain 3,000 entries. The output of the web-tool is the synset and the set of related words of a given input - padam (word in its first person singular form) or prātipadikam (stem). The tool consciously confined to the first 3,000 entries as new fields and features kept evolving through the research. For example, including English meanings was not part of the initial plan but was included as it would help users. Once the web-tool is fine-tuned in all respects, it is easier to scale it up to include the entire database.

An Android Application version of the tool is also currently under development. An initial version is available for volunteer testing to get feedback and suggestions on usability. The Android App is briefly described in Section 3.8.

3.2 Data Structure

The first step towards the creation of the web-tool is to digitise the entire kośa. The following categories of information are extracted from the ślokaś.

प्रातिपदिकम् (stem), पदम् (nominative form), सन्दर्भसूची (reference), लिङ्गम् (gender), अध्यायः (chapter) and काण्डः (section)

For example, the śloka number 47 in the lokapālādhyāya reads as follows:

वातो वायुर्जगत्प्राणश्शुषिलश्श्वसनोऽनिलः।
गन्धवाहो गन्धवहो मातरिक्षा समीरणः॥१.२.४७॥

The words are extracted and categorized as in Table 1.

प्रातिपदिकम्	पदम्	सन्दर्भसूची	लिङ्गम्	अध्यायः	काण्डः	आङ्ग्लार्थः	अर्थः	मुख्यपदम्
वात	वातः	1.2.47.1.1	पुं.	लोकपालाध्यायः	स्वर्गकाण्डः	epithet of vāyu	स्पर्शगुणकः पञ्चभूतभेदः	वायुः
जगत्प्राण	जगत्प्राणः	1.2.47.1.2	पुं.	लोकपालाध्यायः	स्वर्गकाण्डः	epithet of vāyu	स्पर्शगुणकः पञ्चभूतभेदः	वायुः
शुषिल	शुषिलः	1.2.47.1.3	पुं.	लोकपालाध्यायः	स्वर्गकाण्डः	epithet of vāyu	स्पर्शगुणकः पञ्चभूतभेदः	वायुः
श्वसन	श्वसनः	1.2.47.1.4	पुं.	लोकपालाध्यायः	स्वर्गकाण्डः	epithet of vāyu	स्पर्शगुणकः पञ्चभूतभेदः	वायुः
अनिल	अनिलः	1.2.47.1.5	पुं.	लोकपालाध्यायः	स्वर्गकाण्डः	epithet of vāyu	स्पर्शगुणकः पञ्चभूतभेदः	वायुः
गन्धवाह	गन्धवाहः	1.2.47.1.5	पुं.	लोकपालाध्यायः	स्वर्गकाण्डः	epithet of vāyu	स्पर्शगुणकः पञ्चभूतभेदः	वायुः
गन्धवह	गन्धवहः	1.2.47.2.1	पुं.	लोकपालाध्यायः	स्वर्गकाण्डः	epithet of vāyu	स्पर्शगुणकः पञ्चभूतभेदः	वायुः
मातरिक्षन्	मातरिक्षा	1.2.47.2.2	पुं.	लोकपालाध्यायः	स्वर्गकाण्डः	epithet of vāyu	स्पर्शगुणकः पञ्चभूतभेदः	वायुः
समीरण	समीरणः	1.2.47.2.3	पुं.	लोकपालाध्यायः	स्वर्गकाण्डः	epithet of vāyu	स्पर्शगुणकः पञ्चभूतभेदः	वायुः

Table 1: Information extraction of the synset वायुः

It is to be noted that words from वातः till समीरणः are synonyms, i.e. words with the same meaning.

1. प्रातिपदिकम् is the stem of the tokens from śloka and has been used so that it is compatible with other computational resources such as morphological generator and analyser, various e-lexicons etc.; many of them use प्रातिपदिकम् as input and not the पदम्.
2. पदम् field contains the nominative singular form of the प्रातिपदिकम्, generated using the morphological generator. In the case of नित्यबहुवचनान्त words, the nominative plural form will be taken. If a प्रातिपदिकम् has more than one gender, and masculine form is one of them, the masculine singular form is taken. In case the word has feminine and neuter forms the neuter form of the प्रातिपदिकम्, will be used. These guidelines are based on the

rules of the dictionaries such as Śabdakalpadruma, Vācaspatya etc. This option does not appear in AKN but has been introduced in VKN web-tool to allow users to search for a word using पदम् option in case they are unsure about प्रातिपदिकम् of a particular word. It is hoped that this feature will make the tool user-friendly.

3. सन्दर्भसूची is the reference indicating the precise position of the word in VK using a 5-tuple number as kāṇḍa, adhyāya, śloka, pāda and word number in the pāda. The pāda number and word number in the pāda are entered manually into the database, whereas the other fields are derived automatically.
4. लिङ्गम् - gender information of the word. The gender of a word is decided by the meta-information mentioned by Yādavaprakāśa. Cross reference to लिङ्गसङ्ग्रहाध्याय as well as Oppert's vocabulary is also consulted.
5. अध्यायः refers to the chapter or the adhyāya name to which the entry belongs. The adhyāyas are named based on the topic or subject that the word is categorized under. Thus, this field gives an ontological idea about the word.
6. काण्डः refers to the specific section of VK or kāṇḍa to which the entry belongs.
7. आङ्ग्लार्थः or the meaning in English is an additional field that has been included to document from the translation that Oppert compiles under the vocabulary section of the book. This has been included to ensure VKN is accessible to those who may not be Sanskrit scholars.
8. अर्थः refers to the meaning in Sanskrit given by Yādavaprakāśa in VK. Where ever the meaning is not found in VK, other dictionaries have been referred.
9. मुख्यपदम् or headword represents the synset with synonymous words. Headword is chosen as follows - if the headword used in AK appears in VK synset, that word is chosen as the headword. In case, there is no equivalent word in AK, Oppert's vocabulary at the end of the kośa is referred to choose the headword. There are some challenges in choosing the headword because there are no commentaries on VK that a researcher can refer to in case of doubt. However, effort has been made to ensure that words are chosen as far as possible based on the available resources - Compatibility with AKN and Oppert's vocabulary being a primary guiding forces.

As compared to AKN, three categories, namely - पदम्, meaning in English and meaning in Sanskrit are additional fields incorporated into VKN. The decision to incorporate these additional fields was taken mid-way through the research as it was found to be a useful improvisation over the AKN.

3.3 Relations in VKN

The various relations amongst different headwords are marked in the database. Twelve hierarchical or associative relations are marked in different fields - two kinds of ontological categories, class and attribute are marked in the last two fields. Except ontological categories, all other relations are marked using headwords.

3.3.1 पर्यायवाची (Synset)

The set of words that have similar meaning is defined as a synset. See the example of वातः in table 2. The output synset is displayed in the Figure 6 in the appendix.

3.3.2 अवयव-अवयविभावः (Part-whole Relation)

The अवयव-अवयवि relation is marked to indicate part and whole relation. For example - the synset पक्षः is a part of the synset पक्षी. Each member of the synset पक्षः is related to the members of पक्षी through this relation³.

³See Figure 7. in appendix

3.3.3 परा-अपरासम्बन्धः (Superset-subset Relation)

This field marks परा-अपरासम्बन्धः. For example - the synset मृदुवातः is a kind of वायुः. So the synset मृदुवातः is related to the synset वायुः with परा-अपरा relation. Each member of the synset मृदुवातः is marked to the synset वायुः⁴.

3.3.4 जन्य-जनकभावः (Child-parent Relation)

This field marks जन्य-जनकभावः of two concepts. For example - the synset of पार्वती is related to the synset हिमवान् through जन्य-जनक relation. पार्वती is daughter of हिमवान् and हिमवान् is father of पार्वती.⁵

3.3.5 पति-पत्नीभावः (Husband-wife Relation)

This field is meant for marking पति-पत्नी relation. For example - the synset of शची is related to the synset इन्द्रः with Husband-wife relation. Here इन्द्रः is the husband of शची and शची is the wife of इन्द्रः.

3.3.6 स्व-स्वामिभावः (Owner-property Relation)

स्व-स्वामि relation is marked to indicate owner-property relation. For example - the synsets of वैजयन्तः - the house of इन्द्रः and अमरावती - the city of इन्द्रः are related to the synset इन्द्रः with owner-property relation. इन्द्रः is the स्वामी of वैजयन्तः and अमरावती.

3.3.7 सेव्य-सेवकभावः (Lord-servant Relation)

सेव्य-सेवक relation is marked to indicate lord-servant relation. For example - the synset of गरुडः, the vehicle of विष्णुः is related to the synset विष्णुः with lord-servant relation. विष्णुः is the सेव्यः of गरुडः and गरुडः is the सेवकः of विष्णुः.

3.3.8 धर्म-धर्मिभावः (Property-locus Relation)

धर्म-धर्मि relation is marked in this field. For example - the synsets of वैष्णवी, the power of विष्णुः is related to the synset विष्णुः with property-locus relation. विष्णुः is the धर्मि of वैष्णवी and वैष्णवी is the धर्मः of विष्णुः.

3.3.9 गुण-गुणिभावः (Quality-qualificand Relation)

गुण-गुणि relation is marked in this field. For example - the synsets of श्रीवत्सः, the mark of विष्णुः is related to the synset विष्णुः with quality-qualificand relation. विष्णुः is the गुणि of श्रीवत्सः and श्रीवत्सः is the गुणः of विष्णुः.

3.3.10 उपजीव्य-उपजीवकभावः (Life-livelihood Relation)

उपजीव्य-उपजीवक relation is marked to indicate livelihood. For example - the synset of मत्स्यः is related to the synset धीवरः with life-livelihood relation. मत्स्यः is the उपजीव्यम् of धीवरः and धीवरः is the उपजीवकः of मत्स्यः.

3.3.11 अवतारः (Incarnation)

In this field the incarnation or अवतार relation is marked. For example - the synsets of वामनः, श्रीरामः and श्रीकृष्णः are related to the synset विष्णुः with अवतार relation.

3.3.12 अन्यसम्बन्धाः (Associated With)

This field is meant for other relations which are not defined. For example - the synset देवः may be related to the synset स्वर्गः with a relation आवाससम्बन्धः, is not taken care of. The other relations such as बन्धुता, सौभ्रात्रम्, भ्रातृत्वम् etc. are also not considered here. All such relations are marked as अन्यसम्बन्धाः. These relations will be categorised later.

⁴See Figure 8. in appendix

⁵See Figure 9. in appendix

3.4 Ontological Categories

The ontological categories are handled based on the corresponding ontological charts as described in the जाति: and उपाधि: sections below.

3.4.1 जाति: (Ontological Class)

The universal property of a word is considered as jāti. The ontological categories are marked according to the ontological chart proposed by Nair S S. et. al. (2013). The जाति chart is given in the appendix in Figure 12. Each and every entry has ontological class mentioned in the field⁶.

3.4.2 उपाधि: (Attribute)

Any property ie. qualified to be the universal as per the conditions mentioned in the article of Nair S S. et. al. (2013) is considered as upādhi. The उपाधि classes are marked according to the उपाधि chart proposed. The उपाधि chart is given in the Figure 13 in the appendix.

3.5 Frequency Analysis

For frequency analysis set of 3,000 words are considered. Among them 2876 words are found unique. 2719 words have single sense, 191 words are having two senses, 81 words are having 3 senses and nine words are having four senses. Out of 3000 words 659 Synsets are created. For each word, one or more relations are marked using headwords. Hierarchical relations such as परा-अपरासम्बन्ध: and अवयव-अवयवीसम्बन्ध: are the highly frequent relations. The frequency of high frequent occurrences is detailed in the Table 2.

Relation	Total Words	Total Synsets
परा-अपरासम्बन्ध:	1631	356
अवयव-अवयविभाव:	391	117
जन्य-जनकभाव:	286	15
अन्यसम्बन्धा:	275	83
पति-पत्नीभाव:	175	21
स्व-स्वामिभाव:	149	68
अवतार:	106	15

Table 2: Relational statistics

3.6 Data Implementation

Once the lexicon table was ready with the data, three databases were created using dbm engines of Unix using hashing techniques. Three hash tables were created to represent a data structure to map a given key to value.

- i. Hash table for मुख्यपदम्_headword (key = पदम्_word and value = मुख्यपदम्_headword)
- ii. Hash table for synset (key = मुख्यपदम्_headword and value = synset)
- iii. Hash table for पदम्_word info (key = word and value = निगम:_Reference & लिङ्गम्_Gender)

With the help of this data structure, a user can key in a desired word and get output in the form of synonyms, meaning and related information about the word.

The ontological structure adopted for creating the web-tool for Amarakośa has been replicated here with modifications for two reasons. Firstly, the division of various kāṇḍas and categorization of words in both the lexicons are very similar and therefore what has been

⁶See Figure 6. in appendix

created earlier can be easily adapted to VK as well. Secondly, this also enables for technical integration of the two tools in future that will facilitate easy cross-reference.

As this is an ongoing project, the other relations will be supplied in due course as appropriate. It was felt that the first step to get the synsets in order will provide vital wealth of information to researchers and students on this lexicon and the emphasis was thus on categorizing the headword first.

3.6.1 Processing Flow

The input word, the type of requested information (meaning and relation with other words) along with parameters like input encoding for the identification of input and output encoding for formatting of the result on the webpage is processed by a series of scripts in the server. The scripts identify the word and the relation for which the information is requested. They access the databases corresponding to the relations that have been created a-priori, and extract information from the database(s) corresponding the selected relation and format the output into HTML file.

When “All Relations” is chosen as the input, a pictographic representation of all relations is created and embedded in the resultant HTML file. Refer Figure 11. for the output in the appendix.

This HTML file is returned as response to the requesting browser for display to the user. The following flowchart describes the steps in the processing in Figure 3.

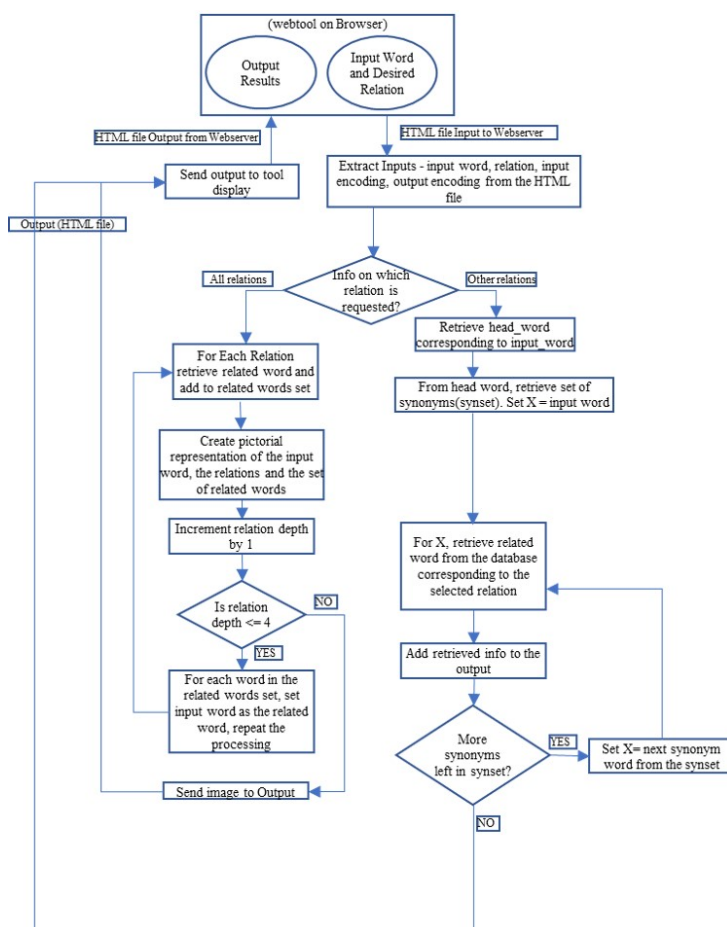


Figure 3: Flowchart for processing in the web tool

3.7 Architecture of VKN

Figure 4 illustrates the architecture of the Vaijayantīkośa KnowledgeNet (VKN) tool. It consists of the following functional components.

- Web user interface
- Webserver
- VK datasets

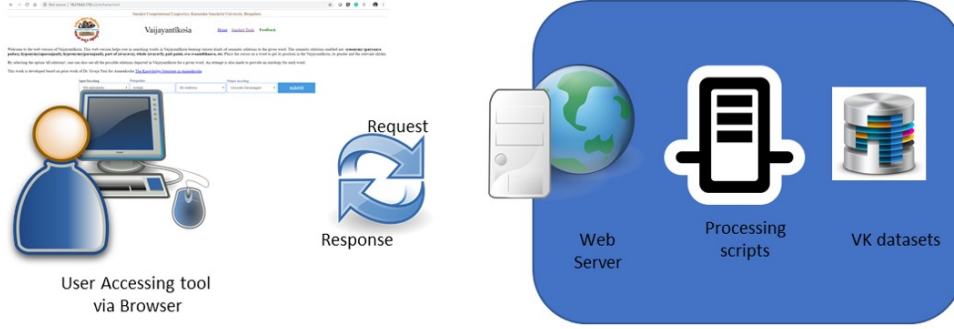


Figure 4: Architecture of the VKN tool

3.7.1 Web User Interface

The user interface for the tool is a HTML web page (currently, first version of the tool is available at <http://13.235.131.68/CompLing/vk/>) It provides a means to input the word from the VK lexicon that needs to be analysed for the specific set of relations. See Figure 5.

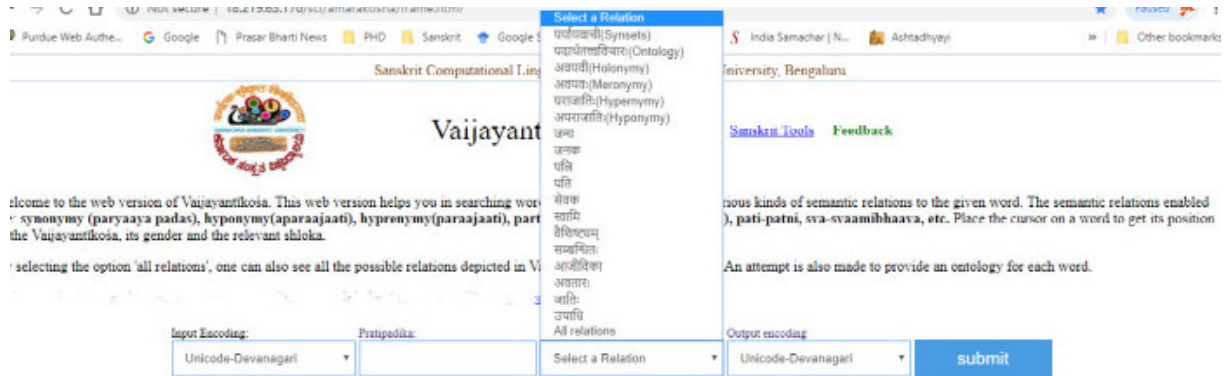


Figure 5: VKN tool

Multiple input encoding forms are provided including Devanagari and WX encoding. The input word can either be प्रातिपदिकम् or प्रथमा एकवचनरूपम्. The desired semantic relation can be extracted from the lexicon from the drop-down list. The tool supports analysis of the relations mentioned in the section 3.3.

3.7.2 Webserver

An Apache webserver hosted on an ubuntu instance running on AWS, is used to interact with the Web user interface. It captures the inputs from the HTML webpage and passes onto the CGI script in the backend for processing. The result of the processing is sent as HTML response to the requesting webpage for display to the user.

3.7.3 VK Datasets

WX encoded original VK śloka and database (see section 3.2 and 3.3) that contain manually created and verified metadata for each word are the input files. The databases are created as per the data implementation described in section "Data Implementation". The processing scripts analyse the inputs for requested information/relations associated with the words in VK, retrieve desired information from these datasets and display the results in the tool as results.

3.8 VKN Android Application

VKN Android App provides a convenient interface to Android smartphone users to access and analyse information in the Vyjayantīkośa. It uses the same input data set, words and relation information used for the web-tool. The Android App collects inputs from the user i.e - the word and its relation. It then communicates the input parameters to a webserver hosted in the cloud, where python scripts are used to search and formulate the response using the input data set. The response is conveyed back to the App on the smartphone for display.

The VKN android App is available for download from the VKN tool webpage. The tool is under development and has been released for volunteer testing and collecting feedback on usability. It currently allows input in Devanagari format and supports the synset relation analysis. The App is being enhanced to support relations and features supported by the web tool as discussed in the previous sections.(See Figure 10)

4 Conclusions

VK has a rich repository of words from the Sanskrit language and literature. The VKN web-tool enables convenient access to this knowledge. It is also designed to enable analysis in specific areas of research by providing a list of words related to that area, which can be used to trace information related to that area in Sanskrit literature. For example, in a paper published in the Indian Journal of History of Science, the use of the term hemaghna (destroyer of Gold), for lead metal was examined in detail (Dube, 2010), and this uncovered, unique properties of the metal lead, when interacting with Gold. There is scope for deeper research for experts from different fields - geology, geography, ornithology, metallurgy, sociology, biology and more. In this context, this tool becomes significant as it provides preliminary information to researchers in their respective fields with synsets and ontological structure and could become a starting point for a more comprehensive research. The inclusion of meaning in English, bridges the language divide, connecting this knowledge base with the large number of English speaking researchers.

5 Future Research

Few suggested future work is as follows:

- Continue updating the kośa with all the remaining entries.
- The child-parent, master-possession, husband-wife relations and other such relations (see section 3.2) were captured at this stage. There are possibilities of including other relations such as siblings, dwellings etc.
- Linking each synset to Amarakośa Knowledge Net.
- Linking it with various other computational linguistic tools.
- Using for Word sense disambiguation.
- Currently only four layers of nesting depth is represented in "all relations". This can be expanded to more layers in future.

Acknowledgement

The Authors would like to thank Prof. Amba Kulkarni, Department of Sanskrit Studies, University of Hyderabad, Hyderabad for providing technical support at various levels. The project team also acknowledges the valuable guidance of Prof. Shrinivasa Varakhedi, Vice-chancellor, Kavikulaguru Kalidasa Samskrita University, Nagpur. The team of Post Graduate Diploma in Sanskrit Computational Linguistics at Karnataka Samskrit University, Bangalore is also acknowledged.

References

- Bühler, G. (1887). Gleanings from Yâdavaprakâsa's Vaijayantî. Wiener Zeitschrift für die Kunde des Morgenlandes, 1, 1-7. Retrieved April 1, 2019, from <https://www.jstor.org/stable/23858800>
- Dube, R. K. (2010). An Assessment of the Sanskrit Word Hemaghna used for Lead Metal. Indian Journal of History of Science, 395-401.
- Haragovindashastri, P. (1971). Vaijayantikośa. Varanasi: Chowkhamba Sanskrit Series Office.
- Kaur, S., & Singh, L. (2018). Indian Arthropods in Early Sanskrit Literature: A Taxonomical Analysis. Indian Journal of History of Science, 59-64.
- Kulkarni Amba. & Nair S Sivaja. (2010) Knowledge Structure in Sanskrit Kosas. Proceedings of 8th ICON, Indo-wordnet Workshop, IIT Khragpur.
- Nair, S Sivaja. (2011). The Knowledge Structure in Amarakośa. Hyderabad: University of Hyderabad.
- Nair, S Sivaja, Varakhedi Shrinivasa & Shivani V (2013). Extended Nyaya-Vaisheshika Ontology as Applied to Amarakośa KnowledgeNet, Proceedings of 5th ISCLS, IIT Bombay, DK Print world, New Delhi
- Oppert, Gustav. (1893). The Vaijayanti of Yadavaprakasa. Madras: Madras Sanskrit and Vernacular Text Publication Society.
- Patkar, M. M. (1981). History of Sanskrit Lexicography. New Delhi: Munshiram Manoharlal.
- Popescu, F. (2019). A Paradigm of Comparative Lexicology. UKNewcastle upon Tyne, UK: Cambridge Scholars Publishing.
- Sun, M. H., Safwanah, N. L., & Tan, D. (2017). Lexicology: The Importance of Words in Society. Universiti Sains Malaysia. Retrieved April 19, 2019, from https://www.researchgate.net/profile/Ernest_Mah/publication/320839664_Lexicology_The_Importance_of_Words_in_Society/links/59fcb88baca272347a22773b/Lexicology-The-Importance-of-Words-in-Society.pdf
- Varakhedi, Shrinivasa., Jaddipal, Viroopaksha. & Sheeba, V. (2007). An effort to develop a tagged lexical resource for Sanskrit. Sanskrit Computational Linguistics, 339-345. Retrieved April 19, 2019, from <https://hal.inria.fr/inria-00207962/d>

A Appendix - 1

VKN Sample Outputs

अर्थः :: पक्षी | वर्गः :: खगाध्यायः | , पक्षिन्, पत्ररथ, पत्रिन्, पित्सत्, पिपतिषत्, पतत्, पतङ्ग, पतग, प्लाविन्, पतत्रिन्, अङ्गु, पतत्रि, विहङ्गम, विहग, विहङ्ग, नभसङ्गम, नीडोद्भव, शुक्र, नीडिन्, मलूक, विप्रुष, भसत्, वशाकु, मदन, पीतु, मशाक, मदुर, द्विज, ऊक, शकुन्ति, शकुनि, शकुन्त, शकुन, खग, शलक, विकिर, तुण्डिन्, नीडज, वातगामिन्

जाति
=> पक्षी
=> जन्तुः
=> मनुष्येतरः
=> चलसजीवः
=> पृथ्वी
=> द्रव्यम्
=> पदार्थः

Figure 6: Example of Ontology

अर्थः :: अग्निः | वर्गः :: लोकपालाध्यायः | , वह्नि, वैश्वानर, घासि, कृष्णवर्त्मन्, समन्तभुज, जातवेदस्, बृहद्भानु, वीतिहोत्र, तनूनपात्, दहन, ज्वलन, शुष्मन्, रोहिताश्व, उषर्बुध, शोचिष्केश, त्रिधामन्, अग्नि, उदर्विस्, पावक, अनल, हिरण्यरेतस्, सप्तार्चिस्, वसुरेतस्, हुताशन, कृपीटयोनि, अर्चिष्मत्, धूमकेतु, दुरासन्, मन्त्रजिह्व, सप्तजिह्व, सुग्निह्व, हव्यवाहन, आश्रयाश, वातसख, कृशानु, वातसारथि, वभि, भुजि, पचि, साधि, चिदि, वञ्चति, अञ्चति, जागृवि, सहुदि, साद्भि, दनुमन्, हवन, हव, सुवाकु, भरथ, पीथ, जुहुराण, ईषिर, आशिर, तेजस्, अप्पित, अपांपित्त

अवयवः (Meronym)

अर्थः :: धूमः | वर्गः :: लोकपालाध्यायः | , धूम, तरि, मेघवाहिन्

अर्थः :: जिह्वा | वर्गः :: लोकपालाध्यायः | , शिरसा, जिह्वा, कील, ज्वाल

अर्थः :: स्फुलिङ्गः | वर्गः :: लोकपालाध्यायः | , स्फुलिङ्ग, अपुञ्ज, खड्गाङ्ग

अर्थः :: सन्तापः | वर्गः :: लोकपालाध्यायः | , सञ्चर, सन्ताप

अर्थः :: उल्का | वर्गः :: लोकपालाध्यायः | , उल्का, क्रमुक, अलात, उत्सुक

अर्थः :: भस्मन् | वर्गः :: लोकपालाध्यायः | , भसित, भस्मन्, भूति

Figure 7: Example of अवयवः

अर्थः :: वायुः | वर्गः :: लोकपालाध्यायः | , वात, वायु, जगत्प्राण, शुषिल, श्वसन, अनिल, गन्धवाह, गन्धवह, मातरिक्षन्, समीरण, युजिन्, लघुग, वाति, ध्वजप्रहरण, जगत्, दैत्यदेव, वह, प्राण, नभःप्राण, अङ्कति, सर, आशुग, पवन, स्पर्श, पवमान, प्रभञ्जन्, मरुत्, धूलिध्वज, मर्क, समीर, अग्निसख, चल, शुष्मि, महाबल, वेगिन्, नभोजात, सदागति, नभस्वत्, मारुत्, शीघ्र, पृषदश्च, प्रकम्पन्

अपराजाति (Hyponym)

अर्थः :: सङ्क्रा | वर्गः :: लोकपालाध्यायः | , सङ्क्रा

अर्थः :: वात्या | वर्गः :: लोकपालाध्यायः | , वात्या

अर्थः :: मृदुवातः | वर्गः :: लोकपालाध्यायः | , मृदुवात, चिञ्चलिक, लेढ, लिह, सौरत

अर्थः :: हारितः | वर्गः :: लोकपालाध्यायः | , हारित, नेरिन्, यौनिक, स्वेदचूषक

अर्थः :: झञ्झानिलः | वर्गः :: लोकपालाध्यायः | , स्वरिङ्गण, झञ्झानिल

Figure 8: Example of अपराजातिः

अर्थः :: कार्तिकेयः | वर्गः :: आदिदेवाध्यायः | , कुमार, शरज, स्कन्द, तारकारि, उमासुत, महासेन, महातेजस, शक्तिपाणि, गुह, अग्निभू, वैजयन्त, सिद्धसेन, सेनानी, शिखिवाहन, स्वामिन्, गाङ्गेय, गौरेय, ब्रह्मचारिन्, महौजस, षाण्मातुर, कार्तिकेय, ब्रह्मगर्भ, षडानन, सुब्रह्मण्य, नीलवट्ट, क्रौञ्चारि, कुक्कुटध्वज

जनक

अर्थः :: शिवः | वर्गः :: आदिदेवाध्यायः | , महेश्वर, पशुपति, श्रीकण्ठ, पांसुचन्दन, शङ्कर, गिरिश, रुद्र, गिरीश, शशिभूषण, भद्रेश्वर, चन्द्रमौलि, पिनाकिन्, शशिशेखर, कपर्दिन्, धूर्जटि, शर्व, कपालिन्, नीललोहित, ईश, ईश्वर, ईशान, भर्ग, मृत्युञ्जय, मूड, व्योमकेश, महादेव, प्रमथाधिपति, शिव, शूलिन्, दक्षाध्वराराति, कामारि, परमेश्वर, कृत्तिवासस, अहिर्बुध्न्य, नीलग्रीव, त्रिलोचन, गङ्गाधर, विरूपाक्ष, वामदेव, वृषध्वज, भूतेश, खण्डपरशु, स्थाणु, अन्धकसूदन, भगनेत्रान्तक, भीम, त्रिपुरारि, दृगायुध, कटाटङ्क, जटाटीर, जटाझाट, महानट, झिण्टीकान्त, रेरिहाण, सर्वज्ञ, नन्दिवर्धन, स्थाल, डिण्डीश, उड्डीश, कण्ठकाल, महाव्रत, कृशानुरेतस, जोटिङ्ग, कङ्कटीक, उमापति, खड्गङ्गिन्, मन्दरमणि, उलन्द, वृषवाहन, उग्र, कटप्र, दिवासस, झण्ड, षण्ड, अकृतश्चन, बहुरूप, अर्धमुकुट, दशबाहू, दशाव्यय

Figure 9: Example of जनकः

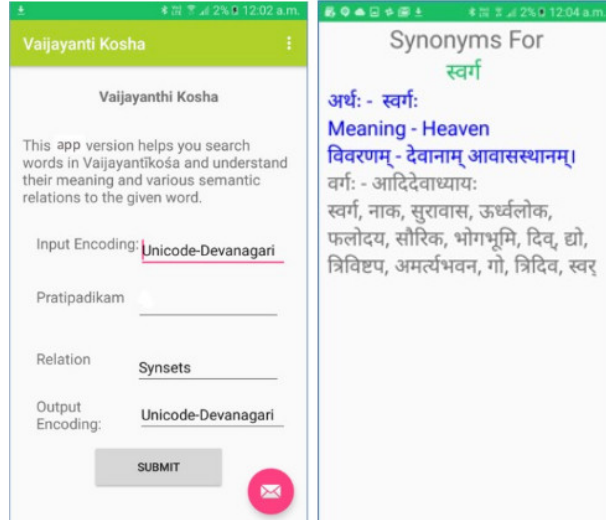


Figure 10: VKN Android Application

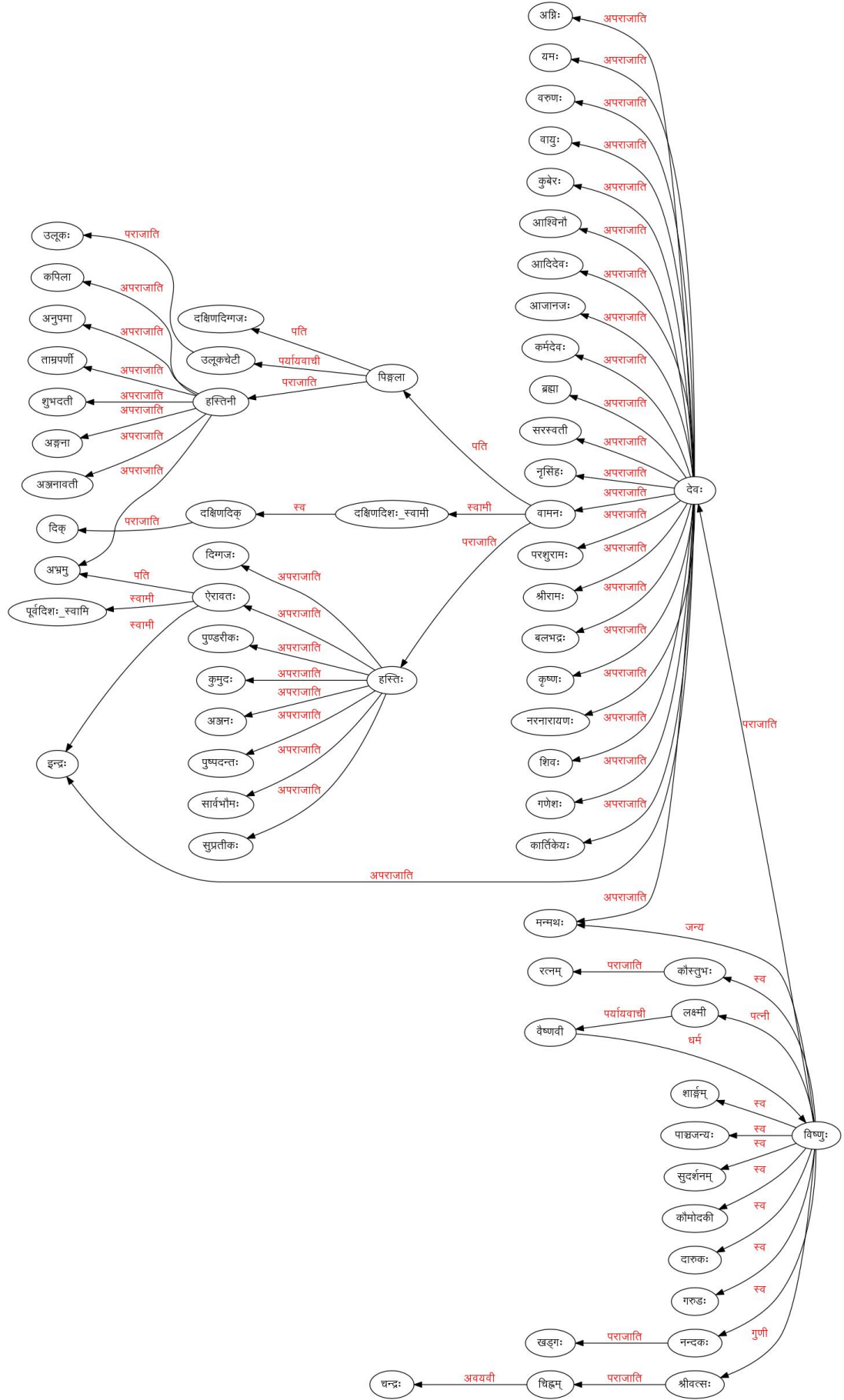


Figure 11: Example of All-relations of Viṣṇu

उपाधिवृक्षः

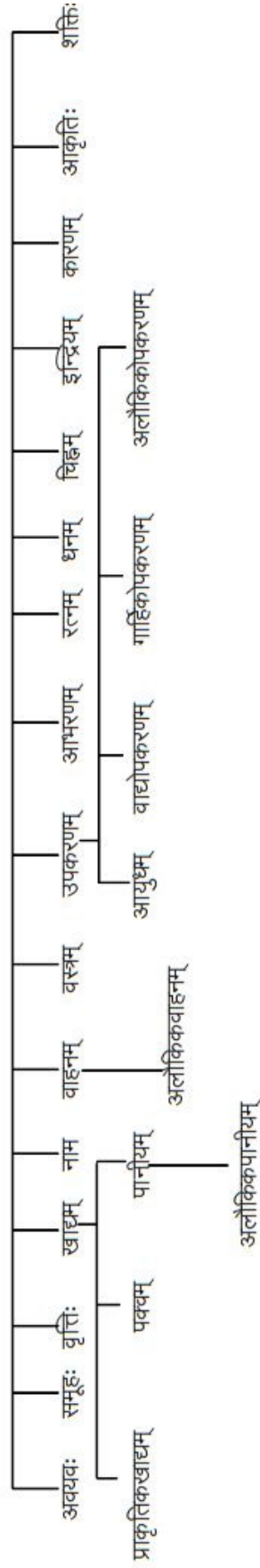


Figure 13: Upādhi Chart

Utilizing Word Embeddings based Features for Phylogenetic Tree Generation of Sanskrit Texts

Diptesh Kanojia^{†♣*}, Abhijeet Dubey[†], Malhar Kulkarni[†], Pushpak Bhattacharyya[†], Reza Haffari^{*}

[†]IIT Bombay

[♣]IITB-Monash Research Academy

^{*}Monash University

[†]{diptesh,abhijeet,pb}@cse.iitb.ac.in, [†]malhar@iitb.ac.in,

^{*}reza.haffari@monash.edu

Abstract

Tracing the root of a text i.e., the original version of the text, by inferring phylogenetic trees has been a topic of interest in philological studies. However, existing methods face meaning conflation deficiency due to the usage of lexical similarity based measures which feed the distance matrix to clustering algorithms. In this paper, we utilize word embeddings as features to compute the distances among manuscripts. We conduct this pilot study on using word embeddings to compute inter-manuscript distances and provide an effective distance matrix to infer phylogenetic trees. We conduct experiments on the historical Sanskrit text known as Kāśikāvṛtti (KV) and infer phylogenetic trees using this approach. For comparison, we also develop baseline methods using lexical distance-based measures to infer phylogenetic trees for KV. We show that our methodology produces better trees which club closely related manuscripts together compared to the baseline methods.

1 Introduction

Phylogenetics is defined as the task of creating a Phylogenetic Tree which represents a hypothesis about the evolutionary ancestry of a set of genes, species or any other taxa. It is the study of evolutionary history and relationships among various taxa. A Taxon represents a group of one or more manuscripts written in Sanskrit in our case, where we analyze how the manuscripts are related to each other. These relationships are discovered through phylogenetic methods that compute observed heritable traits in a manuscript, such as spelling errors, variations in text, text deletion, the morphology of the text etc. under a model of the evolution of these traits. The result of these analyses is a phylogeny (also known as a phylogenetic tree) – a diagrammatic hypothesis about the history of the evolutionary relationships of a group of manuscripts (usually belonging to the same text).

The computational purview of our problem deals with developing new methodologies for the estimation of the said trees. Computational historical linguistics, which involves the development of methods for estimating evolutionary histories of languages and, of models of language evolution, is another research problem based on phylogenetics. Phylogenetic methods are designed to recover the “true” evolutionary tree as often as possible. They do not guarantee to do so with high probability under reasonable conditions. Some which offer this guarantee vary considerably in their requirements (Warnow et al., 2001). To rigorously establish the validity of such a phylogenetic approach, a fundamental question that must be addressed is whether the models in use are identifiable. Parameters for simple models include the topology of the evolutionary tree, edge lengths on the tree, and rates of various types of substitution, though more complicated models have additional parameters as well. If a model is non-identifiable, one cannot show that performing inference with it will be statistically consistent. Informally, even with large amounts of data produced by an evolutionary process that was accurately described by the model, we might make erroneous inferences if we use a non-identifiable model. Under other models, many methods will be able to recover the tree if given long enough sequences.

The latter techniques are said to be statistically consistent under the model of evolution. Under some models of evolution, no method can be guaranteed to recover the true tree with high probability, due to unidentifiability.

Using the currently available models, finding optimal phylogenetic trees using compatibility criteria is, in its general case, NP-Complete (Warnow, 1993). Also, finding a maximum compatible tree is NP-Hard (Roch, 2006). As a consequence, this will mean that efficient algorithms to solve the problem, probably, can not exist. On the other hand, by restricting the kinds of input to the problem, we may be able to solve it efficiently. Our work restricts the input of data to a distance matrix which consists of distances between various manuscripts. We hypothesize inter-manuscript distance by using two methodologies and are able to construct phylogenetic trees based on both of them. Phylogenetic reconstruction and analysis is based on a data matrix where the rows represent the manuscripts to be studied, and the columns represent a linguistic feature or character (Nichols and Warnow, 2008). Moreover, the methods inspired from glottochronology take a boolean matrix as input, which denotes the change in the state of the ‘characters’ (the ‘characters’ can be lexical, morphological or phonological) to infer the phylogenetic trees. In our case, the distance matrix consists of manuscripts to be studied in both rows and columns, but the distances calculated are based on either character-based features (which is our baseline methodology) or word embeddings based distances which is our novel contribution to the area.

Our work is based on an earlier published sample edition of the KV on A 2.2.6 (Kulkarni, 2009). This edition was prepared using seventy manuscripts written in several scripts and collected from various parts of the world. This earlier work did not utilize the computational method to establish inter-relations between manuscripts. Kulkarni and Kahrs (2018) also published a manually drawn tree based on the edition mentioned above. In this work, we apply the computational methods on the same data mentioned above and automatically infer phylogenetic trees that show the inter-relations between manuscripts.

1.1 Motivation

Texts are important sources of intellectual history. In the Indian context, texts have travelled in the course of time both orally and written. Establishing a particular text using extant available resources enables us to have an authenticated base for the reconstruction of intellectual history. In order to create an authenticated base, we need to apply technological tools and methods. These will ensure objectivity and scientific explanation in the establishment of the text. Previous work on creating phylogenetic trees have not explored the usage of word embeddings which foray in the semantic space of linguistics. Word embeddings can provide a highly accurate representation of the context for a given word (Rong, 2014)

Rama and Singh (2009) use corpus-based measures to compute the distance matrix containing inter-language distances and construct phylogenetic trees for a linguistic area¹. Corpus-based measures can calculate the inter-language distance, but they use feature n-grams and cognate identification methods which loosely take into account the semantics of a word. It is well known that word meaning can be represented with a range of senses/concepts. The methods above do not take into account the ‘semantics’ in a language and measure the inter-language distance only based on associated words pairs. Recently, an increasing boom on large-scale pre-trained word embedding models e.g., FastText (Bojanowski et al., 2017), ELMo (Peters et al., 2018), BERT (Devlin et al., 2018) have attracted considerable attention in the field of NLP. Inspired by the above works, this paper proposes to use word embeddings (Mikolov et al., 2013) created using fasttext approach (Conneau et al., 2017) to find the inter-manuscript distance based on functional units in a text.

¹The term linguistic area or Sprachbund (Emeneau, 1956) refers to a group of languages that have become similar in some way as a result of proximity and language contact, even if they belong to different families. The best-known example is the Indian (or South Asian) linguistic area.

The question that we try to answer in this paper is:

“Can word embeddings with sub-word information help build more accurate phylogenetic trees from multiple versions of a manuscript ?”

2 Related Work

Computational phylogenetics has, in recent years, developed various methodologies under the purview of computational biology (Felsenstein and Felsenstein, 2004; Huelsenbeck et al., 2001; Saitou and Nei, 1987; Swofford et al., 1996). The growth of phylogenetics as an area with significance to statistical methods is captured by Felsenstein (2001) in an article where he explains the developments of numerical methods for the creation of phylogenies. These methods have been widely adopted in computational linguistics for the construction of phylogenetic trees. A major disadvantage of using these character-based or lexical distance-based methods is the need for manually curated word lists. Csernel and Patte (2007) discuss the LCS algorithm for preparing a critical edition of Sanskrit texts and provide a method for comparison of Sanskrit manuscripts. Among the many available methods (Huelsenbeck, 1995) to construct phylogenetic trees, UPGMA (Gronau and Moran, 2007) is widely used in historical linguistics. It assumes a constant rate of evolution and is not a well-regarded method for inferring relationships unless this assumption has been tested and justified for the data set being used. The UPGMA method constructs phylogenetic trees based on a distance matrix which can be computed in various ways. Saitou and Nei (1987) proposed neighbour joining method to construct phylogenies based on sequence analysis, which uses genetic distance as a clustering metric. Moret et al. (2002) study the sequence lengths required by neighbour-joining, greedy parsimony, and a phylogenetic reconstruction method based on disk-covering and the maximum parsimony criterion and show improvements in large scale phylogenetic reconstruction. Symmetric cross-entropy is one of the methods which is purely a letter n-gram based measure similar to the one used by Singh (2006b) for language and encoding identification. Singh and Surana (2007) used corpus-based measures to show that corpus can be used for a comparative study of languages. They used both character n-gram distances and surface similarity (Singh, 2006a) to identify the potential cognates, which in turn are being used to estimate the inter-language distance. Rama and Singh (2009) also used measures based on cognate identification, and feature n-grams to infer this matrix. Ellison and Kirby (2006) discussed establishing a probability distribution for every language through intra-lexical comparison using confusion probabilities and estimate distances using KL divergence and Rao’s distance (Atkinson and Mitchell, 1981). Automatic Cognate Detection (ACD) is an important task which can help phylogenetic reconstruction and complement current research on language phylogenies (Rama et al., 2018). Rama (2016) come up with siamese architectures that jointly learn phoneme level feature representations and language relatedness from raw words for cognate identification. Rama et al. (2017) explore the use of unsupervised methods for detecting cognates in multilingual word lists. They use online EM to train sound segment similarity weights for computing similarity between two words. Kanojia et al. (2019) utilize wordnets and identify cognates among Indian languages for improvement in the construction of the phylogenetic trees. They used lexical similarity based measures to find the similarity among Indian language word lists and induced the cognates in clustering methods to generate phylogenies. Kulkarni (2012) builds a phylogenetic tree for Malayalam manuscripts of the Kāśikāvṛtti, and show that M is the archetype source and Ma, Mb and Mc are its hyperarche child nodes. M is decided as a source based on the analysis made on the manual reading of the manuscripts. In this process, manuscripts are grouped together and named as M1, M2, M3 ..., M11. Kulkarni (2003) and Kulkarni (2008) build a similar tree for the Sharada manuscripts of the KV.

To the best of our knowledge, no one has utilized word embeddings to construct the distance matrix for inter-manuscript distances. We deploy lexical similarity-based methods as a baseline for inter-manuscript distance and compare the tree with the trees generated via our approach i.e., using word-embeddings to construct the distance matrix for the clustering methods (UPGMA and Neighbour Joining).

We contribute the following through this work:

- We hypothesize inter-manuscript distance and create efficient distance matrices for phylogenetic tree construction.
- We build baseline methodology using lexical similarity based measures for comparison with our approach and generate phylogenetic trees.
- We construct a distance matrix through a word embeddings based approach as a novel contribution and show that the trees generated are better than the baseline method.

3 Dataset and Experiment Setup

3.1 Dataset

We collect the following data for performing our experiments and tree construction.

3.1.1 KV Dataset

For distance matrix generation, we focus on specific portions of the KV. We collect seventy different versions of the KV on AST 2.2.6. We perform cleaning and manual analysis with the help of philologists. These versions were available in different parts of the country from where we accumulated them in a single repository. We observe different kinds of changes in these versions and describe them in Section 6.

3.1.2 Raw Corpus for obtaining Word embeddings

We obtain raw monolingual Sanskrit corpus from various sources. We download the Sanskrit Wikimedia dump and collate all the articles as a single corpus. We, also, add Glosses and Example sentences from the Sanskrit Wordnet to this corpus. We obtain raw corpus from other sources available online². We perform cleaning for this corpus by removing any other ASCII characters apart from the Devanagari script. The final cleaned corpus used for creating embeddings contains 5,38,323 lines. Eventually, We use binarized vectors to compute the distance between two words.

3.2 Experimental Setup

The Neighbor Joining method and the UPGMA method are both distance-based methods as described in Section 4. They require a distance matrix which specifies the distance between the Taxa being used to populate the phylogeny. We also describe the methodologies used to obtain these matrices in Section 4. For our experiments, we divide the KV data into different functional units. The functional unit division in KV depends on the type of sutra. The sutra that we use for our experiments, namely AST 2.2.6, is of the type vidhi.

The functional unit division of this type is as follows:

- vidhi: This type of sutra prescribes either a verbal element or an operation. The KV on this sutra contains the following functional parts (Sutra AST 2.2.6):
 1. The sentence explaining the meaning of the words in the sutra.
 2. Examples

²Available on the School of Sanskrit and Indic Studies, J.N.U. and NLP for Sanskrit from GitHub

These functional units help us understand the text in a better manner, and for computational purposes, they create separate divisions in the text so that the versions are compared to each other in an efficient manner. We compare each functional unit only with its counterpart from the versions. For e.g., In AST 2.2.6 dataset, we compare the examples from one version only with the examples of the other version.

For training the word embeddings based model, we use Gensim³. We choose FastText (Bojanowski et al., 2017) for training the word embeddings and obtaining vectors as it utilizes subword-level information within the text. Sanskrit is an agglutinative language which is also highly morphological. To capture the morphology and semantics within each word, we also need to take into account the sub-word level information. We train the models with the following hyperparameters. We create these models based on 100 and 50 dimensions due to a limited amount of the corpus collected⁴. The rest of the parameters were the same for both the models. We restrict the context window to 5 and use 0.1 as the learning rate. The maximum length of word n-gram we use is one word. We retain the sampling threshold at a default 0.0001. We use softmax as the loss function and train the models for five epochs⁵.

4 Methodology

In this section, we describe the various methodologies used for calculating the inter-manuscript distances and tree construction.

4.1 Computing the Inter-Manuscript Distances

We use two approaches for constructing the inter-manuscript distances. The baseline approach utilizes various lexical similarity based measures and later, we also provide weights to them, using empirical approaches, to increase their efficiency. In our approach, we use word-embedding based models and compute distances using vectors obtained from them. Since angular cosine distance distinguishes nearly parallel vectors better (Cer et al., 2018), we also include this in our approach, apart from cosine distance to generate more trees and discuss the outcome in Section 5.

4.1.1 Lexical Distance based measures: A Baseline Approach

We use the following lexical similarity based measures to compute the distances among manuscripts:

Normalized Edit Distance Method (NED)

The Normalized Edit Distance (also known as Levenshtein Distance) approach computes the edit distance (Nerbonne and Heeringa, 1997) for all word pairs in a functional unit of the text and then provides as output the average distance between all word pairs (we term it as 'Unit Distance'). In each of the operations has unit cost (except that substitution of a character by itself has zero cost), so NED is equal to the minimum number of operations required to transform 'word a' to 'word b'. A more general definition associates non-negative weight functions (insertions, deletions, and substitutions) with the operations.

Cosine Distance (CoD)

The cosine similarity measure (Salton and Buckley, 1988) is another similarity metric that depends on envisioning preferences as points in space. It measures the cosine of the angle between two vectors projected in a multi-dimensional space. The cosine similarity is particularly used in positive space, where the outcome is neatly bounded in $[0,1]$. The name derives from the term

³Gensim Source

⁴The standard number of dimensions for word embeddings, given a big corpus, is 300

⁵More epochs usually lead to a better learned/trained model; we retain the best epoch output with a minimum loss to be utilized for our work

“direction cosine”: in this case, unit vectors are maximally “similar” if they’re parallel and maximally “dissimilar” if they’re orthogonal (perpendicular). This is analogous to the cosine, which is 1 (maximum value) when the segments subtend a zero angle and 0 (uncorrelated) when the segments are perpendicular. In this context, the two vectors are the arrays of character counts of two words. We calculate the cosine distance as (1 - Cosine Similarity).

Jaro-Winkler Distance (JWD)

Jaro-Winkler distance (Winkler, 1990) is a string metric measuring similar to the normalized edit distance deriving itself from Jaro Distance (Jaro, 1989). Here, the edit distance between two sequences is calculated using a prefix scale P which gives more favourable ratings to strings that match from the beginning, for a set prefix length L . The lower the Jaro–Winkler distance for two strings is, the more similar the strings are. The score is normalized such that 1 equates to no similarity and 0 is an exact match.

Distance Matrix Computation

The above similarity metrics use different ways to compute the distance between each word pair and hence, produce varying distance matrices. We compute the distance between a sutra by averaging over each ‘Unit Distance’ present in a sutra. We compute these distances between all the manuscript pairs. Thus, we generate three inter-manuscript distance matrices based on the methods described above.

Since all the matrices above use different ways to compute distances, we performed another set of experiments for coming up with a more homogenous approach. For computational purposes, we provide all the metrics equal weightages initially, and compute a single the distance matrix using the average score of all three methods. So, for manuscripts p and q , the average inter-manuscript distance is defined as:

$$LD_{pq} = \frac{(NED_{pq} + CoD_{pq} + JWD_{pq})}{3} \quad (1)$$

We, also, experiment over weightages and later provide different weightages to each method. Empirically, we find best results by setting the weight of NED to 50%, CoD to 25%, and JWD to 25%. For manuscripts p and q , the weighted average inter-manuscript distance is defined as:

$$LD_{pq} = (NED_{pq} * 0.5) + (CoD_{pq} * 0.25) + (JWD_{pq} * 0.25) \quad (2)$$

Using the baseline methodology, we create a total of 5 matrices for the text in the AST 2.2.6 dataset.

4.1.2 Word embeddings based distance measures: Our Approach

We calculate the cosine distance between all word pairs belonging to the same functional unit from the embedding space. Thus, the average over the word pair distances gives us ‘Unit Distance’. Similar to the baseline method, we average over all unit distances to find out the inter-manuscript distance for each manuscript pair and compute the distance matrix. Since angular cosine distance distinguishes nearly parallel vectors better (Cer et al., 2018), we also use angular cosine distance and calculate the inter-manuscript distance for each manuscript pair, in a similar fashion. We perform this experiment using two different models described in the experimental setup.

Thus, for each dataset, our approach generates four matrices i.e., a matrix which utilizes Cosine Distance from the model with 100 dimensions, another which utilizes Cosine Distance

from the model with 50 dimensions and another pair of matrices with Angular Cosine Distance from the models with 100 and 50 dimensions each. Using this approach, we create a total of four matrices.

Using all of the methodologies described above (both baseline and our approach), we create a total of 9 matrices for the text in AST 2.2.6 dataset.

4.2 Tree generation using distance-based clustering methods

We choose two distance-based methods for our work, namely, the Neighbor Joining method and the UPGMA method. We further describe these methods below, along with the reasons for choosing these methods.

4.2.1 Distance-based Methods

Distance analysis compares two aligned manuscripts at a time and builds a matrix of all possible sequence pairs. During each comparison, the number of changes (base substitutions and insertion/deletion events) is counted and presented as a proportion of the overall sequence length. These final estimates of the difference between all possible pairs of manuscripts are known as pairwise distances. A variety of distance algorithms are available to calculate the pairwise distance (between versions), for example, Proportional (p) distances. We use the baseline approach and our approach to compute these pairwise distances. Once the pairwise distances are calculated, they must be arranged into a tree. There are many ways to “arrange” the Taxa according to their distances. One way to cluster or optimize the distances is to join Taxa together according to their increasing differences, as embodied by their distances. Other ways use various coefficients to measure how well the branch lengths of the tree reflects the original pairwise distances.

Distance-matrix methods of phylogenetic analysis explicitly rely on a measure of “genetic distance” between the manuscripts being classified, and therefore they require an MSA (multiple sequence alignment) as an input. Distance is often defined as the fraction of mismatches at aligned positions, with gaps either ignored or counted as mismatches (David, 2001). The main disadvantage of distance-matrix methods is their inability to efficiently use information about local high-variation regions that appear across multiple subtrees (Felsenstein and Felsenstein, 2004). Distance methods attempt to construct an all-to-all matrix from the sequence query set describing the distance between each sequence pair. From this is constructed a phylogenetic tree that places closely related manuscripts under the same interior node and whose branch lengths closely reproduce the observed distances between manuscripts. Distance-matrix methods may produce either rooted or unrooted trees, depending on the algorithm used to calculate them. They are frequently used as the basis for progressive and iterative types of multiple sequence alignment. The distance-based methods which we use are:

UPGMA Method

The Unweighted Pair Group Method with Arithmetic mean (UPGMA) method (Sokal and Rohlf, 1962) produces rooted trees and requires a constant-rate assumption, i.e. they assume an ultrametric tree in which the distances from the root to every branch tip are equal. At each step, the nearest two clusters are combined into a higher-level cluster. The distance between any two clusters A and B, each of size (i.e., cardinality) $|A|$ and $|B|$, is taken to be the average of all distances $D(x,y)$ between pairs of objects x in A and y in B, that is, the mean distance between elements of each cluster. In other words, at each clustering step, the updated distance between the joined clusters and a new cluster X is given by the proportional averaging of the distance between A given X and the distance between B given X.

We use the UPGMA method to construct phylogenetic trees for all the manuscript pairs. The input to the UPGMA method is the distance matrix created via the methodologies described above. We use the implementation of UPGMA provided by PHYLIP (Felsenstein, 1993) and

generate baseline trees for NED, CoD, JWD, Average, and Weighted Average distance matrices. We also generate trees for distance matrices obtained using our approach of cosine distances and angular cosine distances from word embeddings space.

Neighbor Joining Method

Neighbour-Joining (Saitou and Nei, 1987) is a bottom-up (agglomerative) clustering method for the creation of phylogenetic trees. It applies general data clustering techniques to sequence analysis and uses genetic distance as a clustering metric. The simple version of the neighbour-joining method produces unrooted trees, but it does not assume a constant rate of evolution (i.e., a constant timeline) across lineages. Neighbour-joining may be viewed as a greedy algorithm for optimizing according to the ‘balanced minimum evolution’ (BME) criterion. For each topology, the tree length (sum of branch lengths) is a particular weighted sum of the distances in the distance matrix, with the weights depending on the topology. The optimal topology (as per BME) is the one which minimizes this length. At each step, it greedily joins the pair of taxa which provides the greatest decrease in the estimated tree length. This procedure is not guaranteed to find the topology which is optimal by the BME criterion, although it often does and is usually quite close.

Similarly, we use the neighbour-joining method to construct phylogenetic trees for all the manuscript pairs. The input to this method is also the distance matrix created via the methodologies described above. We use the implementation of neighbour-joining provided by PHYLIP (Felsenstein, 1993) and generate all the trees from the matrices described above.

5 Results

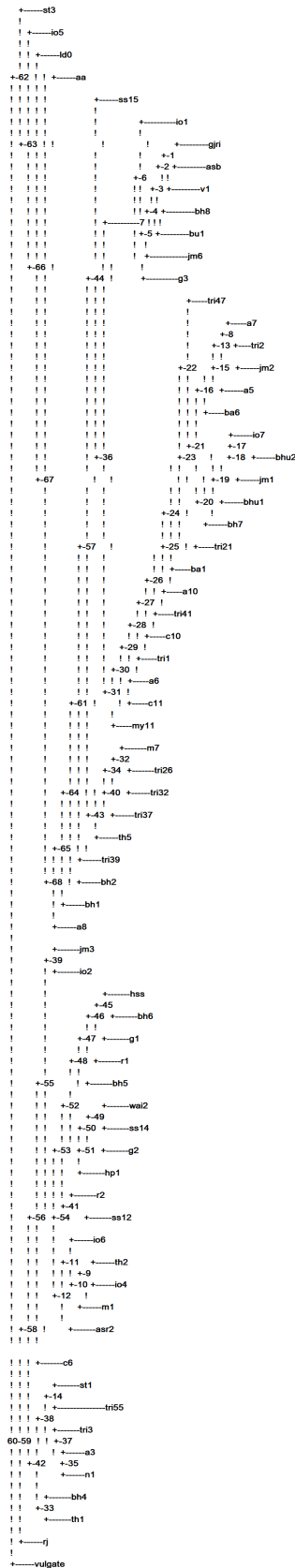
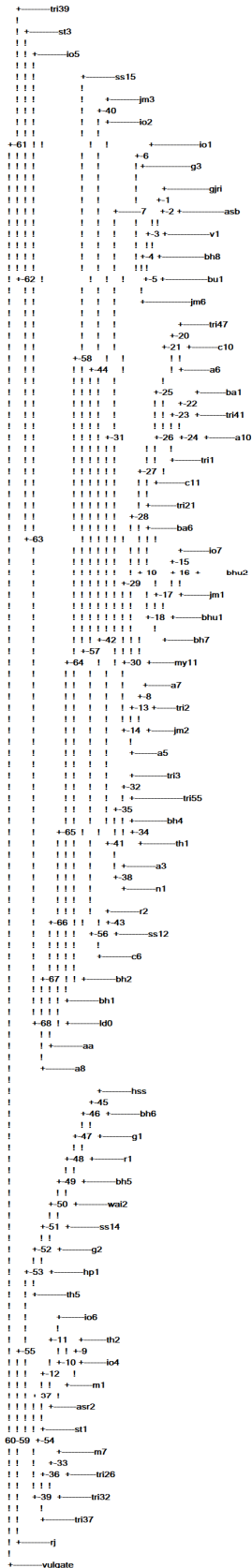
We generate trees using both the neighbour-joining and the UPGMA methods for all the matrices described above and compare them with the trees manually created by our philologists. The basis of this evaluation was the expert knowledge of our philologists who have studied the KV and are aware of the origin, groupings, and a vague timeline of all these manuscript versions. Their findings indicate that the trees generated via our approach of using word embeddings were closest to the manually created trees and required a few corrections among the subgroupings to be accurate. Although, among the baseline approaches, the weighted average methodology also reached the closer to the manually created phylogenetic tree, but it was still a few corrections behind. We can not present the complete set of 18 trees (9 x UPGMA and 9 x Neighbour Joining) here hence show the best tree generated by the baseline method in Figure 1a for the text in 2.2.6 dataset. We obtain this tree using our novel approach of using word-embeddings based model and using Neighbour-joining as the tree generation methodology. In Figure 1b for the text in 2.2.6 dataset, we also show the tree obtained by the weighted average lexical similarity measure, which was also generated using the Neighbour-joining method.

Among the word embeddings based approach, the trees generated via cosine distance are reported to be more accurate than the trees generated via angular cosine distance, as per our philologists.

We compared the matrices generated by both cosine distance and angular cosine distance and found out that the distance values did not have a lot of difference. This is probably due to the lack of a large raw monolingual corpus for creation of word embeddings for Sanskrit. Despite being one of the most ancient languages, the availability of the resource for Sanskrit is scarce, which motivates us further to keep exploring this area. We discuss the results of our work and the merits of our methodology in the next section. We also provide justifications of our philologists’ view in the forthcoming section.

6 Discussion

We discuss the functional units of the AST 2.2.6 dataset in the section above in brief and generate results based on the comparison of each unit. The division of KV data for the AST 2.2.6 text is



(a) Tree Generated using Neighbour-joining method. Distance matrix computed using the word-embeddings based method (b) Tree Generated using Neighbour-joining method. Distance matrix computed using the lexical similarity-based method (See Equation 2)

shown in Table 1.

2.2.6.	नञ्
2.2.6.1	नञ् समर्थेन सुबन्तेन सह समस्यते तत्पुरुषश्च समासो भवति ।
2.2.6.2	न ब्राह्मणो अब्राह्मणः । अवृषलः ।।

Table 1: Example of Functional Unit based Division for sūtra AST 2.2.6

As can be seen in Figure 1a above, the sub-groupings for manuscripts has been done more accurately. Manuscripts io1, g3, gjri, asb, v1, bh8, bu1 and jm6 have been grouped together since they do not contain a common functional unit. The same can be said about the tree in Figure 1b but it does not group bh1 and ld0 in the same sub-group which should not have been the case.

Differences among the manuscript variants in this edition (Kulkarni, 2009) are mainly divided into four categories. The apparatus of this edition contains the mention of the following types of variants:

Omission (Om.): absence of a word.

Addition (Add.): presence of an additional word

Change of word (CW): lexical changes in the word due to morphological inflection, or due to the opinion of the scribe who created the manuscript variant.

Change in the place of a word (CPW): change in the positioning of a word among the functional unit in a text.

We develop both the baseline approach and word embeddings based approach keeping these variants in mind. Our approaches handle these variants in the following manner:

Omission (Om.)

Omission reflects the omitted portion of the text derived after comparing the critical edition with the manuscripts of the text. Our approaches calculate the distances between all word pairs of each functional unit, on both sides. When we perform the comparison between an omitted word on one side and do not find its counterpart on the other side, it results in a higher penalty and a greater distance like it should for an omitted word.

Addition (Add.)

Addition refers to the added portion of the text as available in the manuscripts. It can be one or more words depending on the variant. When we average of all the distances between all word pairs, and in the comparisons made, do not find the added words; it results in a high penalty a greater overall distance like it should for an added portion.

Change of word (CW)

CW refers to a change of word, in the manuscript, in comparison with the critical edition i.e., a word may undergo some morphological inflection or takes some other form but retains a semantic notion. In such a case, the baseline approach measures the lexical changes in a word but penalise this change relatively lower in magnitude. In our approach, since the semantic notion is maintained, the embeddings would provide with nearby vectors and thus also penalise relatively lower in magnitude, which is what should be done for such a variance.

Change in the place of a word (CPW)

CPW refers to the change in the place of the word in the manuscript in comparison with the critical edition. CPW implies that the words in question exist in the manuscript but changes its place. This is not the case with the previous three types of changes. Our methodology counters this variance when we average over all the word pairs. Since the word is indeed present in the functional unit of the text, we should be able to find its occurrence on the other side, and thus this would result in a penalty of lower magnitude in terms of distance. We discuss these approaches with our philologists and their views are in accordance with what our methodology does in penalising computing distances.

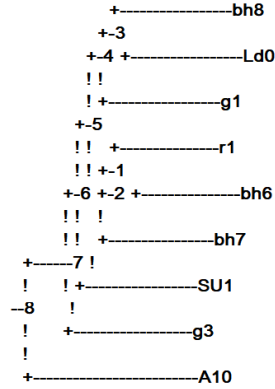


Figure 2: Phylogenetic Tree for the dated manuscripts generated using our method

Availability of the timeline

Ancient Sanskrit text and its manuscripts are scarcely found dated. The unavailability of a timeline (or a temporal reference of versions) of how these texts evolved is a primary reason phylogenetic methods are needed to derive the root version (or the critical edition). We also note that some manuscripts among all the versions are dated, which do help identify the accuracy of a generated tree. Among the seventy versions of KV, we currently have the temporal references for eleven versions. We also generate phylogenetic trees for these versions using the neighbour-joining method based on the distance matrix computed using the word embeddings based approach they provided us with the best trees for AST 2.2.6. We depict this tree in Figure 2. In this tree, we have not yet implemented a method to refer to the timeline which is available. We plan to refine and generate such sub-trees based on the temporal references available to implement more accurate sub-trees of this type.

7 Conclusion and Future Work

In this paper, we presented a novel word embeddings based approach to create inter-manuscript distances and hypothesize functional units as a part of the text. We devised a baseline approach for drawing a comparison from our approach, which is based on lexical distance-based measures. We collect manuscript versions from different sources and accumulate them in a single repository and compute the inter-manuscript distance between all manuscript pairs, thus formulating a distance matrix for each approach. We collect raw Sanskrit corpus from various sources and create a word embeddings model using the state-of-the-art library. We release this word embeddings model publicly for the use of other researchers looking to explore this area. Also, we compute inter-manuscript distances using this model and generate trees for both using both the baseline and this approach. We compare the trees manually, evaluate them with the help of expert philologists where we go on to show that the trees generated via word embeddings based models were better in subgrouping and required the least number of corrections to reach the state of manually drawn trees. We discuss the merits of our approach

with examples and provide justifications of our results. Our approach clearly outperforms the baseline method and thus should help the researchers in this area to create better, more accurate phylogenetic trees in the near future.

In future, we would like to extend our dataset of the KV text to complete all the containing sutras and perform the same experiments for all such portions of the KV text. We plan to divide each of such portions of text into functional units and perform the same experiment for the text. We aim to include the other material like text commentaries and earlier texts as a part of the experiment in the future, as they provide important references to the text.

References

- [Atkinson and Mitchell1981] Colin Atkinson and Ann FS Mitchell. 1981. Rao’s distance measure. *Sankhyā: The Indian Journal of Statistics, Series A*, pages 345–365.
- [Bojanowski et al.2017] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.
- [Cer et al.2018] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, et al. 2018. Universal sentence encoder. arXiv preprint arXiv:1803.11175.
- [Conneau et al.2017] Alexis Conneau, Guillaume Lample, Marc’Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. 2017. Word translation without parallel data. arXiv preprint arXiv:1710.04087.
- [Csernel and Patte2007] Marc Csernel and François Patte. 2007. Critical edition of sanskrit texts. In *Sanskrit Computational Linguistics*, pages 358–379. Springer.
- [David2001] W Mount David. 2001. Bioinformatics: sequence and genome analysis. *Bioinformatics*, 28.
- [Devlin et al.2018] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.
- [Ellison and Kirby2006] T Mark Ellison and Simon Kirby. 2006. Measuring language divergence by intra-lexical comparison. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 273–280. Association for Computational Linguistics.
- [Emeneau1956] Murray B Emeneau. 1956. India as a linguistic area. *Language*, 32(1):3–16.
- [Felsenstein and Felsenstein2004] Joseph Felsenstein and Joseph Felsenstein. 2004. *Inferring phylogenies*, volume 2. Sinauer associates Sunderland, MA.
- [Felsenstein1993] Joseph Felsenstein. 1993. PHYLIP (phylogeny inference package), version 3.5 c. Joseph Felsenstein.
- [Felsenstein2001] Joseph Felsenstein. 2001. The troubled growth of statistical phylogenetics. *Systematic Biology*, pages 465–467.
- [Gronau and Moran2007] Ilan Gronau and Shlomo Moran. 2007. Optimal implementations of upgma and other common clustering algorithms. *Information Processing Letters*, 104(6):205–210.
- [Huelsenbeck et al.2001] John P Huelsenbeck, Fredrik Ronquist, Rasmus Nielsen, and Jonathan P Bollback. 2001. Bayesian inference of phylogeny and its impact on evolutionary biology. *science*, 294(5550):2310–2314.
- [Huelsenbeck1995] John P Huelsenbeck. 1995. Performance of phylogenetic methods in simulation. *Systematic biology*, 44(1):17–48.
- [Jaro1989] Matthew A Jaro. 1989. Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida. *Journal of the American Statistical Association*, 84(406):414–420.

- [Kanojia et al.2019] Diptesh Kanojia, Malhar Kulkarni, Pushpak Bhattacharyya, and Gholemreza Haffari. 2019. Cognate identification to improve phylogenetic trees for indian languages. In Proceedings of the ACM India Joint International Conference on Data Science and Management of Data, pages 297–300. ACM.
- [Kulkarni and Kahrs2018] Malhar Kulkarni and Eivind Kahrs. 2018. Materials for the critical edition of *kāśikāvṛtti* 1.1.
- [Kulkarni2003] Malhar Kulkarni. 2003. The sharada manuscripts of the *kāśikāvṛtti*. pages 353–364.
- [Kulkarni2008] Malhar Kulkarni. 2008. The sharada manuscripts of the *kāśikāvṛtti*: Part ii. pages 419–428.
- [Kulkarni2009] Malhar Kulkarni. 2009. A sample of the new edition of the *kāśikāvṛtti*: 2.2.6. LXV:116–127.
- [Kulkarni2012] Malhar Kulkarni. 2012. The malayalam manuscripts of the *kāśikāvṛtti*: A study. 6:103–112.
- [Mikolov et al.2013] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In Advances in neural information processing systems, pages 3111–3119.
- [Moret et al.2002] Bernard ME Moret, Usman Roshan, and Tandy Warnow. 2002. Sequence-length requirements for phylogenetic methods. In International Workshop on Algorithms in Bioinformatics, pages 343–356. Springer.
- [Nerbonne and Heeringa1997] John Nerbonne and Wilbert Heeringa. 1997. Measuring dialect distance phonetically. In Computational Phonology: Third Meeting of the ACL Special Interest Group in Computational Phonology.
- [Nichols and Warnow2008] Johanna Nichols and Tandy Warnow. 2008. Tutorial on computational linguistic phylogeny. *Language and Linguistics Compass*, 2(5):760–820.
- [Peters et al.2018] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. arXiv preprint arXiv:1802.05365.
- [Rama and Singh2009] Taraka Rama and Anil Kumar Singh. 2009. From bag of languages to family trees from noisy corpus. In Proceedings of the International Conference RANLP-2009, pages 355–359.
- [Rama et al.2017] Taraka Rama, Johannes Wahle, Pavel Sofroniev, and Gerhard Jäger. 2017. Fast and unsupervised methods for multilingual cognate clustering. arXiv preprint arXiv:1702.04938.
- [Rama et al.2018] Taraka Rama, Johann-Mattis List, Johannes Wahle, and Gerhard Jäger. 2018. Are automatic methods for cognate detection good enough for phylogenetic reconstruction in historical linguistics? arXiv preprint arXiv:1804.05416.
- [Rama2016] Taraka Rama. 2016. Siamese convolutional networks for cognate identification. In Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers, pages 1018–1027.
- [Roch2006] Sebastien Roch. 2006. A short proof that phylogenetic tree reconstruction by maximum likelihood is hard. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3(1):92–94.
- [Rong2014] Xin Rong. 2014. word2vec parameter learning explained. arXiv preprint arXiv:1411.2738.
- [Saitou and Nei1987] Naruya Saitou and Masatoshi Nei. 1987. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular biology and evolution*, 4(4):406–425.
- [Salton and Buckley1988] Gerard Salton and Christopher Buckley. 1988. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523.
- [Singh and Surana2007] Anil Kumar Singh and Harshit Surana. 2007. Can corpus based measures be used for comparative study of languages? In Proceedings of ninth meeting of the ACL special interest group in computational morphology and phonology, pages 40–47. Association for Computational Linguistics.

- [Singh2006a] Anil Kumar Singh. 2006a. A computational phonetic model for indian language scripts. In Constraints on Spelling Changes: Fifth International Workshop on Writing Systems. Nijmegen, The Netherlands.
- [Singh2006b] Anil Kumar Singh. 2006b. Study of some distance measures for language and encoding identification. In Proceedings of the Workshop on Linguistic Distances, pages 63–72. Association for Computational Linguistics.
- [Sokal and Rohlf1962] Robert R Sokal and F James Rohlf. 1962. The comparison of dendrograms by objective methods. *Taxon*, pages 33–40.
- [Swofford et al.1996] DL Swofford, GJ Olsen, PJ Waddell, and DM Hillis. 1996. Phylogenetic inference, p. 407–514. *Molecular Systematics* (second edition). Sinauer Associates, Sunderland, Massachusetts.
- [Warnow et al.2001] Tandy Warnow, Bernard ME Moret, and Katherine St John. 2001. Absolute convergence: true trees from short sequences. In Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms, pages 186–195. Society for Industrial and Applied Mathematics.
- [Warnow1993] Tandy J Warnow. 1993. Constructing phylogenetic trees efficiently using compatibility criteria. *New Zealand Journal of Botany*, 31(3):239–247.
- [Winkler1990] William E Winkler. 1990. String comparator metrics and enhanced decision rules in the fellegi-sunter model of record linkage.

An Introduction to the Textual History Tool

Diptesh Kanojia^{†,♣,*}, Malhar Kulkarni[†], Pushpak Bhattacharyya[†], Sayali Ghodekar[†],
Irawati Kulkarni[†], Nilesh Joshi[†], Eivind Kahrs[♠]

[†]IIT Bombay

[♣]IITB-Monash Research Academy

^{*}Monash University

[♠]University of Cambridge

[†]{diptesh,pb,malhar}@iitb.ac.in, [†]sayalighodekar26@gmail.com,
[†]irawatikulkarni@gmail.com, [†]joshinilesh60@gmail.com, [♠]egk1000@cam.ac.uk

Abstract

This paper describes a digital tool called the Textual History Tool in detail. This tool captures the historical evolution of a text through various temporal stages, and inter-related data culled from various types of related texts. This tool also provides a historical view of the transmission of a text through the manuscript tradition. This tool provides an online interface which allows philologists to enter manuscript data for a text. It also provides an online interface which helps philologists compare the variants in a separate mode. It allows the user to generate phylogenetic trees, for the text, based on distance methods using the data entered in the tool. It also contains the facility to generate critical edition using a semi-supervised approach. This tool also divides the text into meaningful functional units and helps achieve a better comparison among the manuscripts. The text of the KV and its textual history is mentioned as a specific example to demonstrate the features of this tool.

1 Introduction

In the twentieth century, before computers came to be used in the effort of preparing the critical edition of a text, philologists used paper-based methods for various purposes viz. collation, description of manuscripts, inter-relation of manuscripts, apparatus creation etc. With the advent of computers and development in technology, we can now have tools with us, that can facilitate the data entry, storage and display of the aforementioned functions, all on the same interface. The tool described in this paper is of the same kind.

A text is, generally a structured verbal expression of intellectual processes. This definition is derived from:

बुद्धिस्तद्व्यक्तिर्निबद्धा ग्रन्थ इत्यभिधीयते । स द्विधा मौखिक आद्यो लिखितोऽन्यश्च कीर्त्यते ॥

It can exist and get transmitted in both forms, oral as well as written.

वायुरूपो मुखे तिष्ठन् ग्रन्थो मौखिक उच्यते । पत्रे मश्यादिभिश्चिह्नैः ग्रन्थो लिखित उच्यते ॥

मौखिकः कर्णनिर्ग्राह्यो लिखितश्चक्षुदर्शनः । मुखाद्मुखं मौखिको हि प्रसरत्यथ कालतः ॥

लिखितश्च पुनर्लैखैः स वै सङ्गमः उच्यते । एवं ग्रन्थे पुनर्दृष्टमैतिह्यं सङ्गमस्य वै ॥

विकासस्य पुनर्बुद्धेः कालतो देशतोऽपि वा । ग्रन्थैतिह्यमहायन्त्रे योजितं स परिश्रमम् ॥¹

Oral transmission led to the development of various vikṛtis i.e., methodologies used to memorize Vedic lore, based on cognitive features. Written transmission is carried out through copies of the text, also known as manuscripts. Historically, manuscripts were written or copied by one or more scribes. Transmission of the text from one source to another generates variants

¹This definition comes from an Unpublished Sanskrit Work ग्रन्थैतिहासोद्योगः by Malhar Kulkarni.

which differ significantly when compared to each other. In terms of expression, the text undergoes various changes in terms of spellings, word replacements etc. Texts are used as the primary sources by scholars in reconstructing the History. The texts assume more significance as a source when it comes to reconstructing the history of an intellectual tradition. These texts represent important stages in the development of thought that contributes to the continuation of the intellectual tradition. What makes the process of reconstruction of intellectual history more complex and therefore, perhaps, more interesting as well as challenging, is the fact that these texts, themselves, are part of a historical process, also known as transmission, and have evolved in certain typical manners and ways in the course of time. It becomes necessary, therefore, in order to study the history of intellectual tradition, the history of the text used as a primary resource.

In the Indian context, we know that the transmission of texts happened in two major ways: oral and written. Texts like Vedas were transmitted from one generation to another, primarily, orally and were written down eventually. So is the case of Epic poems like Ramayana and Mahabharata². In the case of Vedas, though, there is no scope of evolution of the text as such, as it was orally transmitted in a regulated manner with components of the texts noted down in great details up to the level of single letters and accent marks. In the case of Epics, however, the evolution of the text was observed by scholars and traditionally as well, it is believed that Mahabharata, for example, originally consisted of merely 10000 verses which grew in the course of time and has now become a text of one hundred thousand verses (Satasahari Samhita).

When we study the texts in the Indian grammatical tradition, that too, the paninian one, traditional commentators like Madhava and Bhattoji Dikshita etc. (Kulkarni, 2002b; Kulkarni and Kahrs, 2015), and modern scholars like Kielhorn (1887) and Kulkarni (2012a) observe that the text of the Aṣṭādhyāyī (AST) has evolved in the course of time. The text of the sutras that Patanjali had in front of him is not the same as we have it today. As shown by Kulkarni (2015b) and Kulkarni (2016), the traditional commentators quoted above, consider the text of the KV as an important stage of evolution of the text of the AST because the KV brought about numerous modifications in the text of the AST, by sometimes adding a word or two in the sutra, splitting one sutra into two, converting a later vārttika into a sutra etc. Joshi et al. (1995) also state that the KV also preserved a tradition of interpretation of the AST, independent of Patanjali. Bronkhorst (2009) showed that the KV also has an interface with other, non-paninian, Sanskrit grammatical traditions. Therefore it becomes important for scholars interested in the development of an intellectual tradition of linguistic thought in India to study the evolution of the text of the KV seriously through various sources like commentaries and manuscripts³. In order to study this stage of evolution further, when we turn to the printed text of the KV as available to us through more than 10 editions, as of now, we notice that the printed editions do not present to us a picture of a uniform text and rather suggest that this text of the KV that we have with us today, must have evolved in a particular manner historically. Kulkarni (2012c) studied the 'ganapathas' and after analyzing the data from manuscripts showed how the number of words in a 'gana' increased in the course of time and also formulated the stages of this historical development⁴.

²When Malhar Kulkarni delivered his lecture on 'Text and Transmission with special reference to Classical Sanskrit Texts' in Almaty, Kazakhstan on 25th August 2015, some members of the audience remarked that there exist texts even in Kazakhstan, which were committed to memory and were handed down from one generation to the next orally. For oral traditions of India, see (Falk, 1993) and for more recent discussions, see (Kulkarni, 2015a).

³There is no evidence that the KV was ever handed down orally. So oral transmission cannot be used as a resource in the reconstruction of the evolution of the KV. A modern counterexample will also make this point more clear: The text of the VajyakaranaSiddhantaKaumudi was handed down orally, and even Malhar Kulkarni memorised it as part of his traditional education. In fact, it can be said that the primary focus of the structure of the text of the VSK is oral transmission.

⁴Also, Kulkarni presented another paper at the WSC 2018 studying in detail the printed editions of the KV on various Ganas (accepted for publication).

A Brief History of the Critical Edition of the KV in the Post-1990 era

It is this state of affairs with reference to the printed editions of the KV that led to Johannes Bronkhorst and Saroja Bhate to undertake the project of critically editing the text of the KV. Malhar Kulkarni joined this project in 1994 and collected manuscripts from various parts of India and successfully defended his dissertation submitted to the University of Pune in 2000 in which he prepared a critical edition of the KV on A 2.2. Following suit, Deo (2001) submitted her dissertation on the critical edition of the KV on A 3.1 and Dash (2004) on the KV on A 4.1. Malhar Kulkarni also published a sample edition of the KV on A 2.2.6 in a 2005 volume of a journal published by Bharatiya Vidya Bhavan, Mumbai. He also published his studies about the interrelation of groups of manuscripts of the KV (manuscripts written in Sharada script in 2003 (Kulkarni, 2003) and 2008 (Kulkarni, 2008) and manuscripts written in Malayalam script in 2012). In 2010, Eivind Kahrs and Malhar Kulkarni jointly got awarded by British Academy for their proposal to restart editing of the text of the KV critically. Kahrs and Kulkarni worked on preparing the critical edition of the KV on A 1.1 and also collected manuscripts for the same. This effort was further supported by the University of Cambridge through its funds and also by IIT Bombay. Through these funds, they paid their assistants⁵ and assigned various tasks to prepare data for the purpose of critically editing the text of the KV. Through these funds, they could also get the entire manuscript collection earlier stationed at the University of Lausanne, Switzerland shipped to IIT Bombay. The outcome of this support was in the form of a book entitled “Material for the critical edition of the KV” published in April 2018. In 2018, Malhar Kulkarni was awarded another grant by Rashtriya Sanskrit Sansthan, India to critically edit the text of the KV on A 1.1. These grants are the base of our work for the purpose of critically editing the text of the KV. Textual history tool is part of our work to edit the text of the KV critically.

1.1 Functional Divisions of the text of KV

The text of KV, as mentioned above, can be, generally, divided into its functional parts. There are two basic divisions in the text of KV, one that of the sūtra and other of the KV. Within the KV, the text can further be divided according to its functional properties based on the type of sūtra it is commenting upon. We present below the functional divisions in the KV on the samjñā sūtra. Functional parts of the KV on vidhi sutra is described in (Kulkarni, 2012b).

- samjñā: this type of sūtra introduces a technical term, and hence the KV on this sūtra contains the following functional parts:
 1. Introduction of the words in the sūtra and meaning of the sūtra.
 2. Examples.
 3. Mention of other sūtras in which this technical term appears.

An example of the functional division of a sūtra is presented in Table 1.

1.1.1.	Sutra	वृद्धिरादैच्। (१॥१॥१)
1.1.1.1	Introduction & Meaning	वृद्धिशब्दः संज्ञात्वेन विधीयते प्रत्येकमादैचां वर्णानां सामान्येन तद्भाषितानामतद्भाषितानां च। तपरकरणमैजर्थं तादपि परस्तपर इति खड्वैरकादिषु त्रिमात्रचतुर्मात्रप्रसङ्गनिवृत्तये।
1.1.1.2	Examples	आश्वलायनः। ऐतिकायनः। औपगतः। औपमन्यवः। शालीयः। मालीयः।
1.1.1.3	Other Occurences of the term	वृद्धिप्रदेशः। सिचि वृद्धिः परस्मैपदेषु इत्येवमादयः।।

Table 1: Example of Functional Unit based Division of the KV on AST 1.1.1

1.2 Motivation

The Textual History Tool is required because at one go it can present to a reader, the entire history of a text. A text in the Indian context can have a predecessor text as well as a successor

⁵Mukta Tilak, Prajakta Deodhar, Anuja Ajotikar, Trupti Kulkarni, Tanuja Ajotikar and Samhita Joshi.

text. It is an outcome of the intellectual activity based on one or more predecessor texts as well as textual traditions. It becomes a part of intellectual discourse and is commented upon by critical scholars within the same tradition. It gets quoted in the successor texts of the same tradition as well as other traditions and disciplines. It gets copied down in written form for various generations across different geographical regions and in different scripts. In this process, the text itself undergoes various stages of evolution, which can be marked as historical landmarks in the development of thought. Capturing the history of this intellectual world, at a glance, is the aim of this tool.

Currently, available tools do not present the historical information in a form which is coherent, and they do not provide an efficient data-entry interface which can help computational phylogenetics. There are multiple toolkits available which perform computational phylogenetics given the data is formatted in their required input format; none of them takes raw manuscript data to automate the complete pipeline which is the eventual aim of this tool. We allow users to enter raw manuscript data and create functional divisions to ease the task of phylogenetics which is a novel contribution of our work.

The key contribution of our work is:

‘Building a comprehensive tool for visualizing the transmission and history of a text - a tool which can,

- (i) Visualize the multiple versions of the same text which also allows data entry for manuscript versions and thus, helping one compare these versions with each other and aids one in adapting them to a graphical model viz. a phylogenetic tree.
- (ii) Visualize the data from earlier texts.
- (iii) Visualize the data from testimonia.
- (iv) Visualize the data from commentaries.’

2 Related Work

Currently, a lot of texts written in Sanskrit are available in the electronic format available at SARIT⁶, GRETIL⁷, DCS⁸ etc. Many of them are in searchable format. DCS presents texts with various other applied tools like Morphological Analyzer, POS tagger etc. However, no tool presents historical information the way it is needed i.e., with manuscript versions which can be compared/edited at the same time. KWIC is an acronym for Key Word In Context (KWIC) and is the most common format for concordance lines. DCS employs KWIC to be used in the concordance functionality it provides on its interface. Some tools for visualization of data are available online. Csernel and Patte (2007) discuss the LCS algorithm for preparing a critical edition of Sanskrit texts and provide a method for comparison of Sanskrit manuscripts using XML and HTML formats. BabelNet (Navigli and Ponzetto, 2010) is an important lexical resource as far as computational aspects are concerned. Navigli and Ponzetto (2012) design an explorer to visualize its database. It uses the tree layout for visualization which, in the convention, is similar to the phylogenetic visualization of texts. Visuwords⁹ is an online graphical dictionary designed for accessing Princeton WordNet and uses a force-directed graph layout for visualizing the synset structure. Nodebox visualizer¹⁰, on the other hand, provides a very static layout. WordTies (Pedersen et al., 2013) is a WordNet visualizer designed for Nordic and

⁶<http://sarit.indology.info/>

⁷<http://grettil.sub.uni-goettingen.de/>

⁸<http://www.sanskrit-linguistics.org/dcs/>

⁹<https://visuwords.com/>

¹⁰<https://www.nodebox.net/code/index.php/WordNetwo>

Baltic wordnets. Chaplot et al. (2014) present such a visualizer for IndoWordNet- which is a lexical resource for Indian language WordNets.

Overlapping textual structures can be accurately modelled either as a minimally redundant directed graph, or, more practically, as an ordered list of pairs, each containing a set of versions and a fragment of text or data (Schmidt and Colomb, 2009). On a similar note, Hanneder (2010) writes about text genealogy and textual criticism. Maas (2009) discusses the textual versions of Carakasamhitā Vimānasthāna and uses computer stemmatics to aid them in the construction of a Phylogenetic tree later (Maas, 2010). Sathaye (2017) present an analyses of Vetāla-pañcaviṃśati, in the context of 'fluid' textual dynamics and discuss the differences in oral folklore when compared to written text. Phillips-Rodriguez et al. (2009) discuss the transmission of the Mahābhārata and the bifurcations within the diagrams about its written transmission. Kulkarni (2002a) discuss the transmission of KV and conclude that there seems to be no Vt (version) on 2.2.6 in the KV. Kulkarni (2015a) discuss the perspectives on how memory acts as an important device in the tradition of oral transmission of texts.

The TEI Critical Edition¹¹ Toolbox is a tool for preparing a digital TEI critical edition which allows you to check for the encoding of the text. It also facilitates the parallel look-up of the manuscript version by visualizing them on a web-based GUI. Although the software is not available for download and offline use, yet. In the current state, it accepts only TEI format XML files but does not allow one to generate versions. A technique for textual criticism is also provided by West (1973). Classical Text Editor¹² allows one to build a critical edition and critical apparatus manually. It also allows one to prepare the phylogenetic trees but does not provide a visualization interface. It allows one to collate the textual versions and edit them on an offline interface. Our work is significantly different from CTE as our online interface allows multiple users to collaborate and enter data for the same text. It allows the users to create functional divisions in the sutra text being entered and thus helps our novel phylogenetic methodology. In philosophy, our tool is focussed on the entire textual history of which manuscripts are an important part. Our tool preserves testimonia, printed editions, commentaries etc. which the CTE does not. PAUP is a tool for Phylogenetic Analysis based on Maximum Parsimony (Fitch, 1971) and other related methodologies, has been created by Swofford (1999) and is available online¹³. To the best of our knowledge, there is no tool which presents a comprehensive picture of the history of a text by presenting various resources useful for the reconstruction of the history of a text like testimonia, commentaries, earlier texts, printed editions etc.

3 Tool Architecture and Description

The Textual History Tool¹⁴ allows users/philologists¹⁵ to register and the registration to be approved by the tool administrator, which is authenticated based on a username/password based login interface. It also provides philologists with a data entry interface which allows the creation of a text with multiple manuscript versions in the tool database, which is a novel contribution of this work. It also encompasses a view mode, a compare mode, and a tree visualization mode (Kanojia et al., incorporated in Kulkarni and Kahrs, 2018). We describe the tool interface in the form of these modes, in the following subsections.

¹¹<http://ciham-digital.huma-num.fr/teitoolbox/>

¹²<http://cte.oeaw.ac.at/>

¹³<http://paup.sc.fsu.edu/>

¹⁴The idea of developing such a tool was originally conceived by Malhar Kulkarni. He called it ग्रन्थेतिहास-यन्त्रम् in his Sanskrit work mentioned in Footnote 1. He thanks the other authors of this paper for the successful implementation. He wishes to dedicate this tool to the community of Indologists past, present and future. An earlier version of this tool was presented in the demo session at World Sanskrit Conference (2018), Vancouver, Canada.

¹⁵Further, we shall use users and philologists interchangeably depending on the usage of the tool.

3.1 Data Entry

The Data entry interface, based on the user login, allows the user to start with the creation of a new manuscript, or takes them back directly to the last entry they made in a previous manuscript they were working on. At any point, a user can choose to start a new manuscript creation. In such a case, the tool requests the entry of the manuscript label. Upon the entry of the manuscript label, the tool presents the user with an option to enter the manuscript data in a functional unit division or directly in a text box.

We provide this option because manuscripts are different in nature and may not contain that text or may contain the text in a different form. More importantly, the user can choose to enter text directly if they do not feel the need to divide the text into logical units. In such a case, the tool presents the users with text boxes with next and previous buttons, which allows the user to enter the text and move on the next text entry from the manuscript. In the case where the user chooses to enter the text in a functional unit division, they are presented with a text ID along with a text entry field for data. Such fields can be added or removed by the user as per the manuscript text. The user is allowed to create multiple logical divisions, and even leave a functional unit entry empty if the manuscript data requires them to do so. The tool requests the user to enter vulgate data which can be a basic building block for manuscript data for phylogenetic analysis, if the vulgate data is not present the user can ignore the request, and the phylogenetic analysis can then be carried out without it; although they can enter vulgate data at any point later in time. The data entry interface also allows a user to enter commentaries and quotations into the database. These optional entries can allow a philologist to evaluate the phylogenetic tree constructed, and can also aid the tree construction.

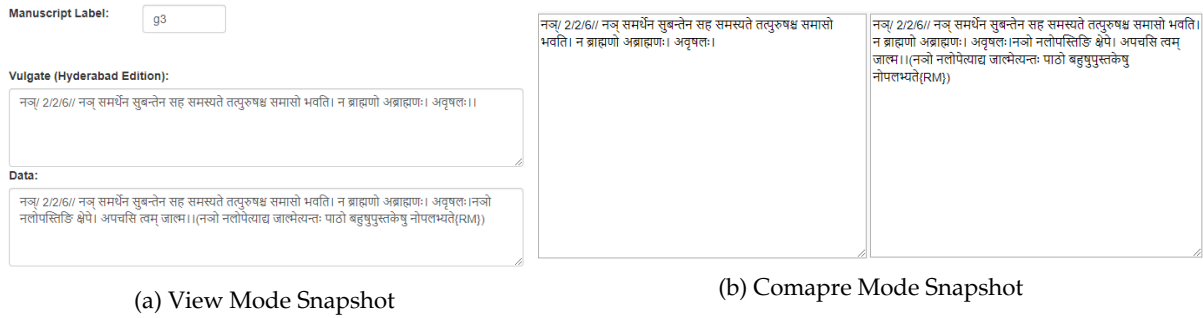


Figure 1: Screenshot from the Textual History Tool

3.2 View Mode

In this mode, the user can view the manuscript version on the interface based on the label. They can select a label from the list labels in the database or search for a label and view the sūtra entries, one at a time; this mode also provides the option to correct an entry based on user privileges. We have added the functionality of viewing the sūtras in the form of functional unit division if they were created with one. This can also be used to instantaneously compare the current version with the Vulgate text, which appears on the top in view mode for each manuscript (if present in the database). A snapshot of the said mode is shown in Figure 1a.

3.3 Compare Mode

It allows a user to view different manuscript versions on the interface based on user selection. The data from Vulgate, if present in the database, is always shown on top for a base comparison. This mode does not facilitate editing of the manuscript versions but allows one to compare versions, the outcome of which can be utilized during a manual analysis later. It allows the user to select one to four versions for comparison. A snapshot of this mode is shown in Figure 1b.

3.4 Phylogenetic Tree Mode

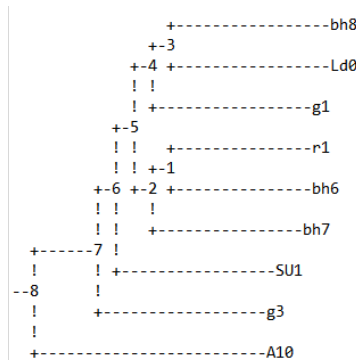


Figure 2: A sample tree produced in the Phylogenetic Tree Mode

This mode is a novel contribution of our work, where based on functional unit distances, a distance matrix can be created. These functional units are part of a text, and thus the user has a choice for selecting one or more texts wherein the functional unit division has been created in the Data Entry mode described in a subsection above. We use two different approaches to create a distance matrix. The baseline approach, which uses the notion of lexical similarity, uses Cosine Distance, Jaro-Winkler Distance, and Normalized Edit Distance to compute these distances. The second approach utilizes word-embeddings learned from Sanskrit corpora, which are stored in a model. These approaches are further detailed in Section 3.5.2.

Eventually, the distance matrix is used to cluster similar manuscripts in the same sub-group, and then the tree can be created using one of the distance based methods viz. Neighbor Joining or UPGMA. These methodologies are also explained in detail in Section 3.5.3. The tree visualization is shown on the interface in the form of manuscript labels being shown as leaf nodes, which can be seen in Figure 2. The user is allowed to view the tree on the interface as well as download it in PDF format for further analysis.

3.5 Technical Development Details

This section provides a detailed technical description of the tool interface frontend and backend. Along with the interface description, it also entails the methodologies used to create the distance matrix which is used for tree generation in the Phylogenetic Tree mode (Section 3.4). The tool architecture is shown as a diagram in Figure 3.

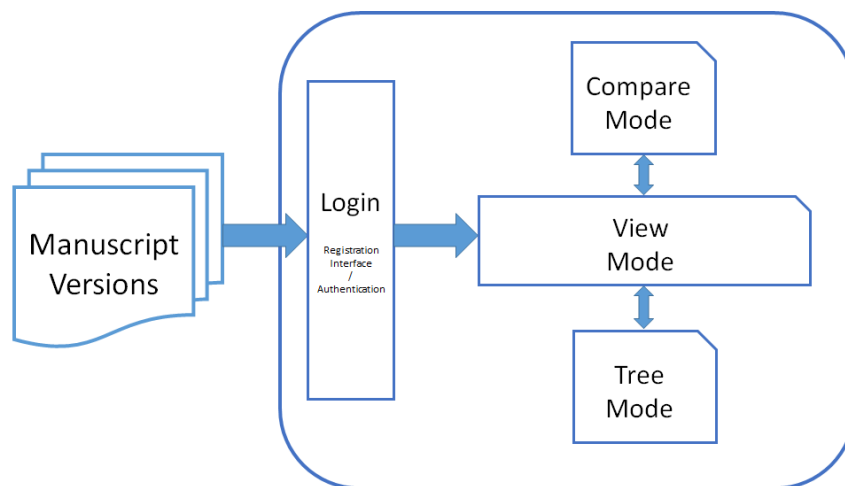


Figure 3: The basic architecture of our tool

3.5.1 Tool Interface

The tool is built as an online web-based interface¹⁶ hosted locally on an Apache Server. It is built using PHP, Javascript and utilizes jQuery for querying the backend. The tool backend utilizes MySQL to efficiently store the manuscript data in a relational database format. MySQL queries from the tool frontend are sanitized before they are sent towards the backend to escape injection attacks. The tool comprises of an authentication interface which is based on username/password based login. The users have to be approved by an administrator after registration, which is available on the login page. The tool users can be granted different privileges based on their usage and expertise in the area. The tool source code can be downloaded and stored offline for local usage¹⁷.

3.5.2 Methodologies for Distance Computation

The phylogenetic tree mode utilizes distance matrix creation based on code written in Python, which can be run for selected manuscripts. Our methodology requires as input the distance matrix between manuscript versions to infer the phylogenetic trees. This distance matrix is computed based on the distance among the functional units, which are divisions in the text as described in Section 1. In case of the unavailability of the division of functional units, the matrix can be computed based on the complete text acting as a single functional unit. The computation of this matrix can be done based on lexical similarity based measures as a baseline method. Our novel approach utilizes word embeddings from a large Sanskrit Corpus-based model, the details of which are below in this section.

Lexical Similarity-based Distance: Baseline Approach

The baseline approach utilizes three different metrics for the computation of lexical similarity. We use Cosine Distance, Normalized Edit Distance, and Jaro-Winkler Distance to compute three scores, which are later averaged into a single score. We also come up with a weighted average mechanism which provide 50% weight to NED, and 25% weight to each CoD and JWD to generate a more efficient tree.

- Normalized Edit Distance Method (NED): The Normalized Edit Distance approach computes the edit distance (Nerbonne and Heeringa, 1997) for all word pairs in a functional unit and then provides as output the average distance between all word pairs or 'Unit Distance'.
- Cosine Distance (CoD): The cosine similarity measure (Salton and Buckley, 1988) is another similarity metric that measures the cosine of the angle between two vectors projected in a multi-dimensional space. In this context, the two vectors are the arrays of character counts of two words. We calculate the cosine distance as (1 - Cosine Similarity).
- Jaro-Winkler Distance (JWD): Jaro-Winkler distance is a string metric measuring an edit distance between two sequences. It uses a prefix scale P which gives more favourable ratings to strings that match from the beginning, for a set prefix length L.

The above similarity metrics use different ways to compute similarity between each word pair and hence produces varying distance matrices. For computational purposes, we provide all the metrics equal weightages initially, and compute the distance matrix using the average score of all three methods. For manuscripts p and q , the average inter-manuscript distance is defined as:

$$LD_{pq} = \frac{(NED_{pq} + CoD_{pq} + JWD_{pq})}{3}$$

¹⁶Tool URL ANONYMIZED

¹⁷Tool Source Code ANONYMIZED

We experiment over weightages and later provide different weightages to each method. Empirically, we find best results by setting the weight as described above. For languages p and q , the weighted average inter-manuscript distance is defined as:

$$LD_{pq} = (NED_{pq} * 0.5) + (CoD_{pq} * 0.25) + (JWD_{pq} * 0.25)$$

Word embeddings based distance measures: Our Approach

We calculate the cosine distance between all word pairs belonging to the same functional unit from the embedding space. Thus, the average over the word pair distances gives us ‘Unit Distance’. Similar to the baseline method, we average over all unit distances to find out the inter-manuscript distance for each manuscript pair and compute the distance matrix. Since angular cosine distance distinguishes nearly parallel vectors better (Cer et al., 2018), we also use angular cosine distance and calculate the inter-manuscript distance for each manuscript pair, in a similar fashion.

We train the models with the following hyperparameters. We create the SKIPGRAM model based on 100 dimensions due to a limited amount of the corpus collected¹⁸. We restrict the context window to 5 and use 0.1 as the learning rate. The maximum length of word n-gram we use is one word. We retain the sampling threshold at a default 0.0001. We use softmax as the loss function and train the models for five epochs¹⁹.

3.5.3 Tree generation using distance-based clustering methods

We implement two distance-based methods for our work, namely, the Neighbor Joining method and the UPGMA method. We further describe these methods below, along with the reasons for choosing these methods.

Distance-based Methods

Distance analysis compares two aligned manuscripts at a time and builds a matrix of all possible sequence pairs. During each comparison, the number of changes (base substitutions and insertion/deletion events) is counted and presented as a proportion of the overall sequence length. These final estimates of the difference between all possible pairs of manuscripts are known as pairwise distances. A variety of distance algorithms are available to calculate the pairwise distance (between versions), for example, Proportional (p) distances. We use the baseline approach and our approach to compute these pairwise distances. Once the pairwise distances are calculated, they must be arranged into a tree. There are many ways to “arrange” the Taxa according to their distances. One way to cluster or optimize the distances is to join Taxa together according to their increasing differences, as embodied by their distances.

UPGMA Method

The Unweighted Pair Group Method with Arithmetic mean (UPGMA) method (Sokal and Rohlf, 1962) produces rooted trees and requires a constant-rate assumption, i.e., they assume an ultrametric tree in which the distances from the root to every branch tip are equal. At each step, the nearest two clusters are combined into a higher-level cluster. The distance between any two clusters A and B , each of size (i.e., cardinality) $|A|$ and $|B|$, is taken to be the average of all distances $D(x, y)$ between pairs of objects x in A and y in B , that is, the mean distance between elements of each cluster. In other words, at each clustering step, the updated distance between the joined clusters and a new cluster X is given by the proportional averaging of the distance between A given X and the distance between B given X .

¹⁸The standard number of dimensions for word embeddings, given a big corpus, is 300

¹⁹More epochs usually lead to a better learned/trained model; we retain the best epoch output with a minimum loss to be utilized for our work

Neighbor Joining Method

Neighbour-Joining (Saitou and Nei, 1987) is a bottom-up (agglomerative) clustering method for the creation of phylogenetic trees. It applies general data clustering techniques to sequence analysis and uses genetic distance as a clustering metric. The simple version of the neighbour-joining method produces unrooted trees, but it does not assume a constant rate of evolution (i.e., a constant timeline) across lineages.

4 Tool Features and Functionalities

The tool comprises of the following additional features and functionalities as described below:

4.1 Manuscript Pictures



Figure 4: Screenshot of view mode displaying manuscript picture along with the text in the view mode

In addition to the tree generation and other salient features like a comprehensive data entry mode, the tool comprises of an additional feature where it enables the user to view the pictures of the manuscript document as a proof to substantiate the data. Philologists can attach pictures of the manuscript entry in the data entry mode as an option along with typing the manuscript data for the database entry. This picture (shown in Figure 4 as a screenshot), if uploaded by the philologist, is shown with the data entry in the view mode (Section 3.2).

4.2 Critically Edited Text

The tool also allows one to view the critically edited text in the view mode of the tool. The critically edited text allows a user to have a summarized view with additional opinions for the philologists. This helps a user decide which portion of the manuscript they want to consider for creating phylogenetics trees.

4.3 Critical Apparatus

The critically edited text is usually accompanied by a critical apparatus. The critical apparatus for a text consists of the set of variations made to the critically edited text. These changes are important to note down as they are an essential part of the preservation of historical texts. These changes allow one to notice the originally written text and how it changed over some time. The tool allows a user to view the critical apparatus in view mode as well.

4.4 Text Visualizer

Manuscripts can be envisioned as a tree in a hierarchical manner which helps philologists analyse them, conventionally. We propose a different method of viewing the manuscripts based on their distance. This text visualizer of the manuscripts allows one to view the manuscripts as leaf nodes connected using edges where one can manually change the leaf nodes in the visualizer setting. The visualizer uses the database and computes a distance matrix to visualize the graph. The graph is then creating using javascript based library which enlists all the manuscripts in an

interactive way where one can manually change the leaf nodes and create their own version of a tree.

Additionally, we also implement the visualizer to depict the relation between the text and earlier texts. It can also display the inter-relations between the text and its commentaries along with the testimonia. It provides the user with an option to view these visualizations together and also as separate visualization. This feature allows the user to gather temporal information from the visualization as the database contains dated entries for the testimonia, commentaries, and some manuscripts. This will help the reader to study the evolution of the text as happened in the course of time.

4.5 Text Commentaries

There are some direct and indirect commentaries available which comment on the KV text. The two direct commentaries are Nyāsa (Ny) and Padamañjarī (Pm).

The tool allows a user to view these commentaries on each sūtra by providing a button, clicking on which, the commentary available for this sūtra is displayed to the user. This button acts dynamically on the page and is only visible as a clickable button if a commentary is available for the said sūtra which is under view on that page. This option provides additional insight into the text and allows a more holistic view of the work done on the KV text. Another button to view a sub-commentary is also provided. We also provide the option to view a consolidated version of the textual evidences available through the commentaries, as mentioned above.

Kulkarni (2002b) mentions the effort on the part of its author to collect information from the Ny and the Pm, which can act as an evidence to reconstruct the text of the KV. Kulkarni and Kahrs (2019b) enlist the variants of the text of the KV as found in the Pm through more than 300 quotations.

“There are instances where both the Ny and Pm record the same pratika. There we can say that both the commentaries received the text of the KV in a similar form. There are also cases when both these commentaries are silent about certain readings. And when they remain silent about certain important units of the text, say a vārttika, then it increases the probability that that vārttika might not have been there in the original text of the KV as received by these two commentaries. There are also cases when the pratika recorded by the N and Pm vary. Such cases pose a problem for an editor. In these cases, the problem gets another dimension if the reading of both N and Pm is seen recorded in some number of mss.”

Kulkarni and Kahrs (2019a) show that the textual evidence available in these two commentaries can be classified under two broad categories: Direct and Indirect. While Direct evidence is clearly visible in the text of the Ny and Pm, indirect evidence can be further classified under two categories: paroksha and atiparoksha. They, in turn, can further be classified into six and three categories, respectively. This categorization is shown below in Figure 5. The button in this tool does show all these categories of evidence, thereby displaying the text of the KV as known to these two commentaries.

Indirect commentaries are the commentaries on the direct commentaries. Tantrapradipa (Tp) is a commentary on Ny. Therefore, it becomes an indirect commentary on the KV. Some portions of Tp which are available are used in this work. Tp allows us to determine readings in the Ny, thereby indirectly helping reconstruct the text of the KV.

4.6 Earlier Texts

On the interface, we also provide an option to view the earlier texts. The purpose of this is to provide the reader with the historical view of the text. After clicking on the earlier texts button, the user is provided with an option to choose between “Paninian” and “Non-Paninian” texts. By choosing the option to view “Paninian” texts, the interface shows the earlier texts in the

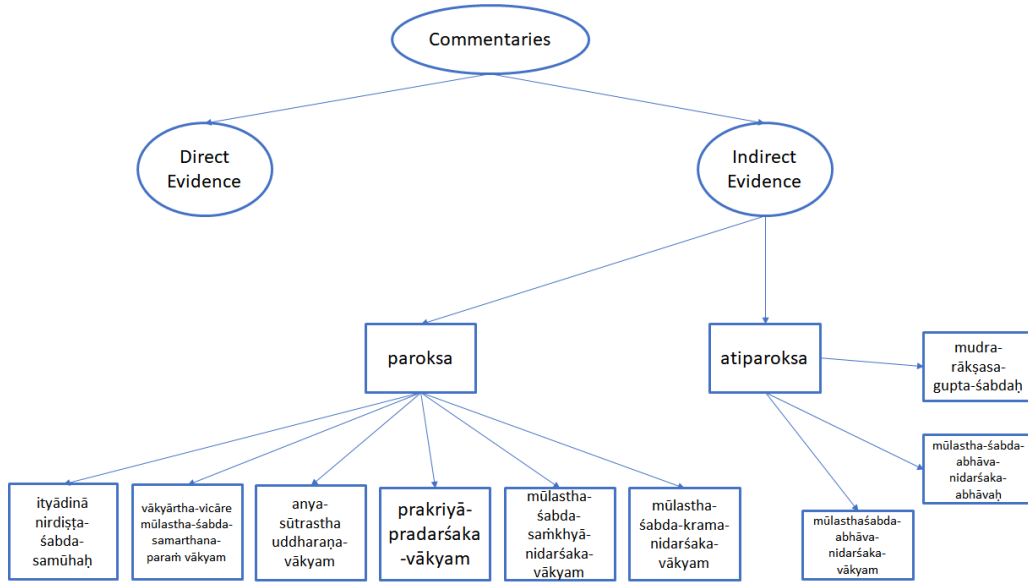


Figure 5: Classification of evidence from the commentaries on the KV(Kulkarni and Kahrs, 2019a).

Paninian tradition, in this context, the Vyakarana Mahabhashya (VMbh). This allows the user to see whether there is any historical connection between the KV and the VMbh. It is noted that VMbh is not available on at least more than 2300 sūtras. In those cases, obviously, the tool shows “Text Not Available”.

When viewing “Non-Paninian” texts, the interface shows the earlier texts in the Non-Paninian traditions namely Katantra, Chaandra, etc. This allows the user to see whether there is any historical connection between the KV and these traditions. This historical connection is also presented in the text visualizer. The visualizer also provides an option to compare manuscript version in the database with the earlier texts. This allows the user to study the inter-relation of a particular version of the text of the KV and the earlier paninian and non-paninian texts.

4.7 Testimonia

The text of the KV is quoted in the later texts grammatical as well as non-grammatical. Kulkarni (2002b) collected and arranged chronologically more than 1000 such quotations as available from the later paninian grammatical tradition. Kulkarni (2002c) studied one quotation of the KV as found in the Shabdkaustubha and showed the inter-relation of KV manuscripts and Shabdkaustubha. The testimonia button displays all these quotations for the sūtra under study.

4.8 Printed Editions

The KV was printed for the first time in 1876. Kulkarni (2000) traced the manuscript sources of this edition. Ever since then, the text of the KV got printed more than ten times (See Footnote 4). When “Printed Editions” is clicked, the interface displays all the printed editions’ text of the sūtra. This historical development in the printed editions is also presented in the text visualizer. It is hoped that the amount of variation available in the printed editions will serve as a basis to understand the manuscript variants.

4.9 Reverse Engineering and the Critical edition

This functionality allows a user to create the manuscript versions of the text based on the critical edition and the apparatus. We use the critical edition of the text and apply the variations mentioned in the apparatus to populate the manuscript versions. We believe that this function acts as a validator for the data present in the tool database.

5 Conclusion and Future Work

In this paper, we describe a tool which captures the historical evolution of a text and allows a user to view the transmission of a text through its history in a comprehensive manner. The tool allows a user to digitize a complete text and its versions through a data entry mode. The data entry mode allows one to partition the text data, based on functional units for a more accurate phylogenetic evaluation. The tool also comprises of view mode, and compare mode which can allow a user to view various parts in the text, along with the comparison of the parts in different manuscripts. Based on the data entry and/or division of functional units in the data, the tool also allows one to compute a distance matrix in the backend, which can be further used to compute a phylogenetic tree in the tree mode. The tool comprises of more features like showing manuscript pictures, visualization of manuscripts like a graph etc. In this paper, we show how this tool successfully digitizes one specific text, and we hope this can also be applied in a general domain. Utilizing all the features of the tool described above, it enables us to identify 19th Century as an important stage, in the evolution and development of this text, as the manuscripts belonging to this period add 2.2.6.3 to the main text. The justifications for this observation are noted by Kulkarni (2002a). The tool may have its technological advantages but still needs humans to interpret the text. We believe this tool can help the community digitize and view the manuscript data in a format which can be helpful to philologists for drawing further insights from the text and to understand the text for better.

In future, we would like more functionalities and different tree inferring methods to the tool. Currently, it only supports distance-based methods as described in the paper above. We would also like to provide options such as fuzzy matching between the text and the commentaries based on which a portion of the commentary can be aligned to a particular portion of the text. This automation can ease the philologists' work by automatically showing them alignments between the commentary portions and the main text. We would also like to implement generation of phylogenetic trees at the micro level (sūtras) as well as the macro level (padas, adhyayas and entire text).

References

- [Bronkhorst2009] Johannes Bronkhorst. 2009. The importance of the kasika. *Studies in the Kasikavrtti. The Section on Pratyaharas: Critical Edition, Translation and Other Contributions*, pages 129–140.
- [Cer et al.2018] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, et al. 2018. Universal sentence encoder. *arXiv preprint arXiv:1803.11175*.
- [Chaplot et al.2014] Devendra Singh Chaplot, Sudha Bhingardive, and Pushpak Bhattacharyya. 2014. Indowordnet visualizer: A graphical user interface for browsing and exploring wordnets of indian languages. In *Proceedings of the Seventh Global Wordnet Conference*, pages 338–345.
- [Csernel and Patte2007] Marc Csernel and François Patte. 2007. Critical edition of sanskrit texts. In *Sanskrit Computational Linguistics*, pages 358–379. Springer.
- [Dash2004] Sasmita Dash. 2004. Critical edition of kashika 4.1.
- [Deo2001] Pooja Deo. 2001. Critical edition of kashika 3.1.
- [Falk1993] Harry Falk. 1993. *Schrift im alten Indien: ein Forschungsbericht mit Anmerkungen*, volume 56. Gunter Narr Verlag.
- [Fitch1971] Walter M Fitch. 1971. Toward defining the course of evolution: minimum change for a specific tree topology. *Systematic Biology*, 20(4):406–416.
- [Hanneder2010] Jürgen Hanneder. 2010. Text genealogy, textual criticism and editorial technique: guest ed. Jürgen Hanneder... Verlag der Österr. Akad. der Wiss.

- [Joshi et al.1995] Shivram Dattatray Joshi, JAF Roodbergen, et al. 1995. The Aṣṭādhyāyī of Pāṇini, volume 4. Sahitya Akademi.
- [Kielhorn1887] Franz Kielhorn. 1887. Notes on the mahabhashya, 6. the text of panini's sutras, as given in the kasika-vritti, compared with the text known to katyayana and patanjali. *Indian Antiquary*, 16:178–184.
- [Kulkarni and Kahrs2015] Malhar Kulkarni and Eivind Kahrs. 2015. On unah um and evidence for one undivided sutra in the text of the Kāśikāvṛtti. pages 1–14.
- [Kulkarni and Kahrs2019a] Malhar Kulkarni and Eivind Kahrs. 2019a. Some more reflections on the role of the Nyāsa and the Padamañjarī in reconstructing the textual history of the transmission of the Kāśikāvṛtti. pages 35–48.
- [Kulkarni and Kahrs2019b] Malhar Kulkarni and Eivind Kahrs. 2019b. Variant readings in the text of the Kāśikāvṛtti as noted by the Padamañjarī. *Journal of the Oriental Institute (Vadodara)*, 67:143–159.
- [Kulkarni2000] Malhar Kulkarni. 2000. On identifying the manuscript(s) at the base of the first printed edition of the Kāśikāvṛtti. pages 203–212.
- [Kulkarni2002a] Malhar Kulkarni. 2002a. On a vārttika on p. 2.2.6 in the Kāśikāvṛtti. *Annals of the Bhandarkar Oriental Research Institute*, 83:201–205.
- [Kulkarni2002b] Malhar Kulkarni. 2002b. A study of quotations of the Kāśikāvṛtti in the late paninian grammatical tradition. pages 73–78.
- [Kulkarni2002c] Malhar Kulkarni. 2002c. A study of quotations of the Kāśikāvṛtti in the late paninian grammatical tradition. pages 73–78.
- [Kulkarni2003] Malhar Kulkarni. 2003. The sharada manuscripts of the Kāśikāvṛtti. pages 353–364.
- [Kulkarni2008] Malhar Kulkarni. 2008. The sharada manuscripts of the Kāśikāvṛtti: Part ii. pages 419–428.
- [Kulkarni2012a] Malhar Kulkarni. 2012a. Franz kielhorn and the text of aṣṭādhyāyī as given in the Kāśikāvṛtti: A study. 84:31–50.
- [Kulkarni2012b] Malhar Kulkarni. 2012b. The malayalam manuscripts of the Kāśikāvṛtti: A study. 6:103–112.
- [Kulkarni2012c] Malhar Kulkarni. 2012c. Some issues in editing the ganapathas in the Kāśikāvṛtti. 6:213–258.
- [Kulkarni2015a] Malhar Kulkarni. 2015a. Memory: a device in traditional sanskrit learning. *Memory and Human Wellbeing: Interdisciplinary Perspectives*, pages 57–72. Edited by Yahei Kanayama, Malhar Kulkarni and Toshiya Uebe.
- [Kulkarni2015b] Malhar Kulkarni. 2015b. Quotations in grammatical texts and the tradition of manuscript transmission of the Kāśikāvṛtti. 43:182–190.
- [Kulkarni2016] Malhar Kulkarni. 2016. Franz kielhorn and the text of the aṣṭādhyāyī as given in the Kāśikāvṛtti: A study - ii. 32:205–212.
- [Maas2009] Philipp A Maas. 2009. Computer aided stemmatics-the case of fifty-two text versions of carakasamhitā vimānasthāna 8.67-157. *Wiener Zeitschrift für die Kunde Südasiens/Vienna Journal of South Asian Studies*, 52:63–119.
- [Maas2010] Philipp A Maas. 2010. On what became of the carakasamhitā after ṛḍḍhabala's revision. *eJournal of Indian Medicine*, 3(1):1–22.
- [Navigli and Ponzetto2010] Roberto Navigli and Simone Paolo Ponzetto. 2010. Babelnet: Building a very large multilingual semantic network. In *Proceedings of the 48th annual meeting of the association for computational linguistics*, pages 216–225. Association for Computational Linguistics.
- [Navigli and Ponzetto2012] Roberto Navigli and Simone Paolo Ponzetto. 2012. Babelnet: The automatic construction, evaluation and application of a wide-coverage multilingual semantic network. *Artificial Intelligence*, 193:217–250.

- [Nerbonne and Heeringa1997] John Nerbonne and Wilbert Heeringa. 1997. Measuring dialect distance phonetically. In *Computational Phonology: Third Meeting of the ACL Special Interest Group in Computational Phonology*.
- [Pedersen et al.2013] Bolette Pedersen, Krister Linden, Kadri Vider, Markus Forsberg, Neeme Kahusk, Jyrki Niemi, Lars Nygaard, Mitchell Seaton, Heili Orav, Lars Borin, et al. 2013. Nordic and baltic wordnets aligned and compared through “wordties”. *Proceedings of NODALIDA 2013*.
- [Phillips-Rodriguez et al.2009] Wendy J Phillips-Rodriguez, Christopher J Howe, and Heather F Windram. 2009. Some considerations about bifurcation in diagrams representing the written transmission of the mahābhārata. *Wiener Zeitschrift für die Kunde Südasiens/Vienna Journal of South Asian Studies*, 52:29–43.
- [Saitou and Nei1987] Naruya Saitou and Masatoshi Nei. 1987. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular biology and evolution*, 4(4):406–425.
- [Salton and Buckley1988] Gerard Salton and Christopher Buckley. 1988. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523.
- [Sathaye2017] Adheesh Sathaye. 2017. The scribal life of folktales in medieval india. *South Asian History and Culture*, 8(4):430–447.
- [Schmidt and Colomb2009] Desmond Schmidt and Robert Colomb. 2009. A data structure for representing multi-version texts online. *International Journal of Human-Computer Studies*, 67(6):497 – 514.
- [Sokal and Rohlf1962] Robert R Sokal and F James Rohlf. 1962. The comparison of dendrograms by objective methods. *Taxon*, pages 33–40.
- [Swofford1999] DL Swofford. 1999. *Phylogenetic analysis using parsimony (and other methods) paup* 4.0*. Sinauer, Sunderland.
- [West1973] Martin L West. 1973. *Textual criticism and editorial technique applicable to Greek and Latin texts*. Walter de Gruyter.

Pāli Sandhi – A Computational Approach

Swati Basapur

Research Scholar

Karnataka Samskrita University
dswatisrini@gmail.com

Shivani V

Professor

Karnataka Samskrita University
shivani.ksu@gmail.com

Sivaja S Nair

Post Doctoral Fellow

Karnataka Samskrita University
sivaja.s.nair@gmail.com

Abstract

For any Indian language, the accuracy of the morphological analyser, depends on the pre-edition of the input text. In Pāli language, like any other Indian language, the combination of words like sandhis and samāsas are frequently seen. This poses difficulty in the proper analysis of the source text. It is essential to have computational tools that help to split the words, useful in the analysis of the text. This paper discusses complexities involved in creating a computational grammar for sandhi tool in Pāli language.

1 Introduction

Pāli is a widely studied classical language, mainly because it is the language of Pāli canon. A growing interest in Pāli makes it important to develop computational tools for the language. Morphological analyser/generator is one such effort in this direction. All the combined words, (sandhis, samāsās, etc.) used in the text have to be manually split before using it as an input to the morphological analyzer in Pāli language. Since it is a tedious effort, pre-editing tools such as sandhi splitter/joiner and samāsa analyser were envisaged. Though similarities were observed in Pāli and Samskrita grammar, it was observed that Pāli grammar was much more complex. This paper discusses the computational approach taken to develop a sandhi splitter/joiner module and the complexities involved therein. In order to develop sandhi module discussed in this paper Kaccāyana grammar¹ has been referred to; as it's rules are comprehensive and supported with a lot of examples.

2 Nature of Pāli Sandhi

Words in Pāli language, end in vowel or anusvāra (niggahita). This feature distinguishes it from Samskrita sandhi structure. Pāli sandhis can be divided into internal and external sandhis. Internal sandhis occur within a word and external sandhis are between words. Mainly sandhis are divided based on what pūrvapada (preceding word) ends with and what uttarapada (following word) begins with. They are divided as follows.

2.1 Svarasandhi

When vowels come in proximity as the end of the pūrvapada and the beginning of uttarapada following changes may occur. Say x is the ending vowel of Pūrvapada, y is the beginning vowel of uttarapada.

1. x may get elided, y remains same.
e.g. समेतु + आयस्मा → समेतायस्मा²
2. x remains same y may get elided.
e.g. चत्तारो + इमे → चत्तारोमे³

¹Tiwari, Laxminarayan and Sharma Birbal (1962) 'Kaccayana vyakarana[Pāli Grammar]', Tara Publications, Varanasi

²सरा सरे लोपं १.२.१

³वा परो असरूपा १.२.२

3. x gets elided and y could be replaced by asavarṇa vowel
e.g. न + उपेति → नोपेति⁴
4. x gets elided and y could be replaced by savarṇa long vowel.
सद्धा + इध → सद्धीध⁵
5. x might get converted to semivowel.
e.g. ते + अस्स → त्यस्स⁶
6. a consonant may get added between x and y .
e.g. लहु + एस्सति → लहुमेस्सति⁷
न + इमस्स → नयिमस्स

2.2 Pakatibhāva

When a word ends with a vowel and is followed by a consonant at the beginning of the uttarapada then both the words remain the same.⁸

e.g. तिण्णो + पारगतो → तिण्णो पारगतो

When a word ends with a vowel and is followed by a vowel at the beginning of the uttarapada then both the words remain the same.⁹

e.g. को + इमं → को इमं

If the preceding vowel is long it may become short.¹⁰

e.g. भोवादी + नाम → भोवादि नाम

If the preceding vowel is short, it may become long.¹¹

e.g. मुनि + चरे = मुनी चरे

2.3 Vyañjanasandhi

In sandhi, if a word ends with a vowel and is followed by a consonant, it is considered as vyañjanasandhi.

e.g. इध + पमादो → इधप्पमादो¹²

The rule for the above example says, if a word ends with a vowel and is followed by a consonant at the beginning of the following word, the latter gets doubled optionally. This word “optionally” is frequently found in the sūtra or its vṛtti. Following is an example where the doubling of consonant does not happen.

e.g. इध + मोदति → इध मोदति

2.4 Niggahitasandhi

If a word ends with niggahita, followed by a word beginning with a vowel or a consonant, it is considered as niggahita sandhi where niggahita undergoes changes.

⁴क्वचासवर्णं लुत्ते १.२.३

⁵दीर्घं १.२.४

⁶यमेदन्तस्सादेसो १.२.६

⁷यवमदनतरळा चागमा १.४.६

⁸सरा पकति व्यञ्जने १.३.१

⁹सरे क्वचि १.३.२

¹⁰रस्सं १.३.४

¹¹दीर्घं १.३.३

¹²परद्धेभावो ठाने १.३.६

e.g. धम्मञ्चरे = धम्मं +चरे.

Here, niggahita changes to ञ् according to the rule वगन्त वा वग्गे १.४.२.

According to this rule, if a vargīya consonant is preceded by niggahita, niggahita gets replaced with अनुनासिक of the same varga. This rule is similar to Pāṇinian rule यरोऽनुनासिकेऽनुनासिको वा ८.४.४५.

3 Computational rules for Sandhi Joiner/Splitter

Pāli sandhi rules stated in the Pāli grammar books of Kacchayana and others are discussed above. Keeping these rules in view, the computational rules for developing module were drawn based on Paninian rules of Sandhi. Following instances were considered:

3.1 Svara + Savarṇasvara

A svara (x) followed by a savarṇa svara (y), there are five possibilities:

1. lopa of x and y remains.
2. lopa of x and y gets elongated.
3. anusvāra or y/v/m/d/n/t/r/L may be inserted between x and y.
4. prakṛtibhāva.

For e.g अ +अ -> अ/आ/अनुस्वार insertion/य,व,द,म,त,र,ळ insertion/remain the same

तत्र + अयं ->

- >तत्रयं (replaced with अ) (1)
- >तत्रायं (replaced with आ) (2)
- >तत्रं अयं (अनुस्वार insertion) (3)
- >तत्ररयं (र insertion) (4)
- >तत्र अयं (remain the same) (5)

Outputs (3), (4) and (5) are not seen in sample gold data. Hence, these options can be hidden in the display.

3.2 Svara + Asavarṇasvara

A svara (x) followed by an asavarṇa svara (y), there are seven possibilities:

1. x remains and lopa of y.
2. lopa of x and y gets guṇa.
3. all rules of section 3.1

For e.g. आ +इ -> इ/आ/ई/अनुस्वार insertion/remain the same

लता +इव ->

- >लताव (1)
- >लतेव (2)
- >लतिव (3)
- >लतीव (4)
- >लतां इव (5)
- >लतामिव (6)
- >लता इव (7)

Outputs (5), (6) and (7) are not seen in sample gold data. Hence, these options can be hidden in the display.

3.3 svāra + vyañjana

A svāra (x) followed by vyañjana (y), there are six possibilities:

1. elongation of x and y remains.
2. x may get replaced with अ and औ and y remains.
3. anusvāra may be inserted between x and y.
4. prakṛtibhāva.

इ + च ->ई/अनुस्वार insertion/इ ->अ/इ ->औ/remain the same e.g.1 मुनि + चरे ->

- >मुनी चरे (1)
- >मुनी चरे (2)
- >मुनि चरे (2)
- >मुनिच्चरे (3)
- >मुनिं चरे (4)
- >मुनि चरे (5)

e.g.2 इध + पमादो ->

- >इधा पमादो (1)
- >इधो पमादो (2)
- >इध पमादो (2)
- >इधप्पमादो (3)
- >इधं पमादो (4)
- >इध पमादो (5)

Outputs (4) and (5) are not seen in sample gold data. Hence, these options can be hidden in the display.

3.4 niggahita(anusvāra) + svāra

A niggahita (x) followed by svāra (y), there are four possibilities:

1. lopa of x, elongation of upadha and y remains.
2. lopa of x and y remains.
3. lopa of x and म् /द् may be inserted between x and y.
4. x remains and lopa of y
5. prakṛtibhāva.

For e.g. anusvāra +अ ->elision of anusvāra and elongation of upadha/elision of anusvāra/elision of अ /insertion of म्

तासं +अहं ->

- >तासाहं (1)
- >तासहं (2)
- >तासमहं (3)
- >तासं हं (4)
- >तासं अहं (5)

Outputs (4) and (5) are not seen in sample gold data. Hence, these options can be hidden in the display.

3.5 niggahita(anusvāra) + vyañjana

A niggahita (x) followed by vyañjana (y), there are three possibilities:

1. x gets replaced with nasal of the same varga.
2. lopa of x and y remains.
3. lopa of x and ण् / ण् may be inserted between x and y.
4. prakṛtibhāva.

For e.g. anusvāra + च -> elision of anusvāra/anusvāra ->nasal of same varga/remain the same धम्मं + चरे ->

- >धम्मच्चरे (anusvāra to nasal of same varga) (1)
- >धम्मचरे (elision of anusvāra) (2)
- >धम्मं चरे (remain the same) (3)

Outputs (2) and (3) are not seen in sample gold data. Hence, these options can be hidden in the display.

3.6 Apavāda rules

pūrvapada (x) and is followed by vyañjana (y):

3.6.1 Apavāda 1

1. if x = पुथ, last letter is replaced by उ and y remains.
2. if x = पुथ, last letter is replaced by उ and y may get doubled.
3. prakṛtibhāva. पुथ + भूतं -> पुथुभूतं
पुथ + जनो -> पुथुज्जनो

3.6.2 Apavāda 2

1. if x = अव, x is replaced with औ and y remains.
2. prakṛtibhāva. अव + नद्धा -> औनद्धा
अव + नद्धा -> अवनद्धा

3.6.3 Apavāda 3

1. if x = पति, x is replaced with पटि and y remains
पति + हञ्जति -> पटिहञ्जति

pūrvapada (x) and is followed by vyañjana (y):

3.6.4 Apavāda 4

1. if x = पा, last letter of x is shortened and ग् is inserted between x and y.
2. prakṛtibhāva. पा + एव -> पगेव
पा + एव -> पाएव

3.6.5 Apavāda 5

1. if x = अभि, x is replaced with अब्भ् and y remains
अभि + उदीरितं -> अब्भुदीरितं

3.6.6 Apavāda 6

1. if x = अधि, x is replaced with अज्झ् and y remains
अधि + ओकासो -> अज्झोकासो

3.6.7 Apavāda 7

1. if $x = \text{अभि/अधि}$ and $y = \text{इ}$, lopa of last letter of x .
2. if $x = \text{अभि}$ and $y = \text{इ}$, lopa of last letter of x .
3. Apavāda 5 and 6 are applicable here.
अभि + इज्झितं -> अभिज्झितं
अभि + इज्झितं -> अब्भिज्झितं
अधि + ईरितं -> अधीरितं
अधि + ईरितं -> अज्झीरितं

3.6.8 Apavāda 8

1. if $x = \text{अति}$, and $y = \text{इ}$, lopa of last letter of x .
अति + ईरितं -> अतीरितं

3.6.9 Apavāda 9

1. if $x = \text{पति}$, x is replaced with पटि , lopa of last letter of x and y remains.
पति + अग्नि -> पटग्नि

From sutra सरा पकति व्यञ्जने १.३.१ and सरे क्वचि १.३.२ together, it can be derived that, if ending vowel of a word comes in proximity of beginning vowel/consonant of the following word, both remain the same (prakṛtibhāva). Therefore the output of this instance need not be shown in the display. Because every instance of sandhi, prakrutibhāva can happen. Similarly, sūtra निग्गहितञ्च १.४.८ indicates insertion of anusvāra for every sandhi instance. Hence output here also can be selectively shown.

4 Complexities Involved

While drawing rules for Pāli sandhi computation, the following complexities were encountered. In the first place, we notice words like क्वचा, वा which means sometimes or optional in many sutras. That makes most of the sandhis optional or having multiple results based on the situation of x and y . Below are some examples to demonstrate the complexities.

4.1 Occurrence of words क्वचा and वा

Majority of sutras i.e. out of 41 kaccāyana sandhi sutras almost 27 sutras have क्वचा and वा in sutra itself or the vṛtti. For e.g वमोदुदन्तानं rule क्वचा is in the vṛtti.¹³ This gives rise to multiple outputs for a given instance when generated computationally. In the case of Sanskrit this ambiguity is mostly fixed by rules themselves. If there are exceptions, they are grouped and gaṇa information is provided. In Pāli, one has to depend heavily on literature to get the forms that are used rather than those which can be generated computationally.

4.2 Inconsistency in examples from literature

Following are examples from piṭakasahitya:

भिक्षवे+इति => भिक्षवेति
ए+ इ => ए + _ वा परो असरूपा

आवुसो +इति => आवुसोति
ओ+ इ => ओ + _ वा परो असरूपा

It is observed that वा परो असरूपा rule is followed in the above examples. This rule is optional but it is applied most of the time wherever dissimilar vowels come in proximity of each other in

¹³Pg 19 Tiwari, Laxminarayan and Sharma Birbal(1962) 'Kaccayana vyakarana[Pāli Grammar]', Tara Publications, Varanasi

sandhi.

But in the following example from the same text, though dissimilar vowels are in proximity of each other, it is seen that सरा सरे लोपं and दीघं are applied. So this seems to be an exception to the above rule.

च + इध => चीध
अ + इ => _ + इ सरा सरे लोपं
_ + इ => _ + ई दीघं

Whereas in the examples below सरा सरे लोपं and दीघं are followed where similar vowels are in close contact in sandhi.

भुञ्जामि+इति => भुञ्जामीति इ+ इ => _ + इ सरा सरे लोपं
_ + इ => _ + ई दीघं Another example from the same text
पमुच्छति+इति => पमुच्छतीति
न+अत्थि => नत्थि¹⁴ सरा सरे लोपं

Here elongation of vowel has not occurred. Therefore even from literature, joining or splitting has to be done with caution.

4.3 Ambiguous Rules for Insertion of Letters

Insertion of letters in Pāli sandhi is ambiguous. For e.g. application of the rule called यवमदनतरळा चागमा. This rule says if uttarapada begins with svara then optionally य/व/म/द/न/त/र/ळ can be inserted. In the examples given in the vṛtti:

सम्मा + अञ्जा -> सम्मदञ्जा-> आ + अ -> द insertion
भन्ता + उदिक्खति -> भन्तावुदिक्खति -> आ + उ -> व insertion
अज्ज + अग्गे -> अज्जतग्गे-> अ + अ -> त insertion
अत्त + अत्थभिञ्जाय -> अत्तदत्थभिञ्जाय-> अ + अ -> द insertion

It is observed that for a given instance, the letter inserted is different for a similar condition. Extracting rules from such sutrās is difficult.

4.4 Multiple Possibilities while Splitting

Multiple sutrās are available for splitting the same instance.

For e.g. लतेव can be split as
लता + एव — 1 सरा सरे लोप
लता + इव — 2 क्वचासवण लुत्ते

Above example shows that the split has to be context-based.

In Sanskrit लतेव can be split in only one way i.e. लता + इव and this can be context-independent.

5 Sandhi Joiner

Sandhi Joiner was developed applying the rules enumerated in the previous section. The input to the tool is pūrva-pada and uttarapada. The result is all possible combined words based on the rules that are applicable to a given instance. It also indicates the respective rules which are applied to get that particular output. Sandhi Joiner has three modules - svarasandhi, vyañjanasandhi, and niggaḥitasandhi. Pseudocode for the tool is given section 4.1. Flow chart is given below. The kaccāyana rules used in the respective modules are listed in A Appendix - 1. The screenshots are attached in B Appendix-2. The computational module for the Sandhi Splitter is the reverse of sandhi joiner. The work for this module is under process.

5.1 Pseudocode

Begin

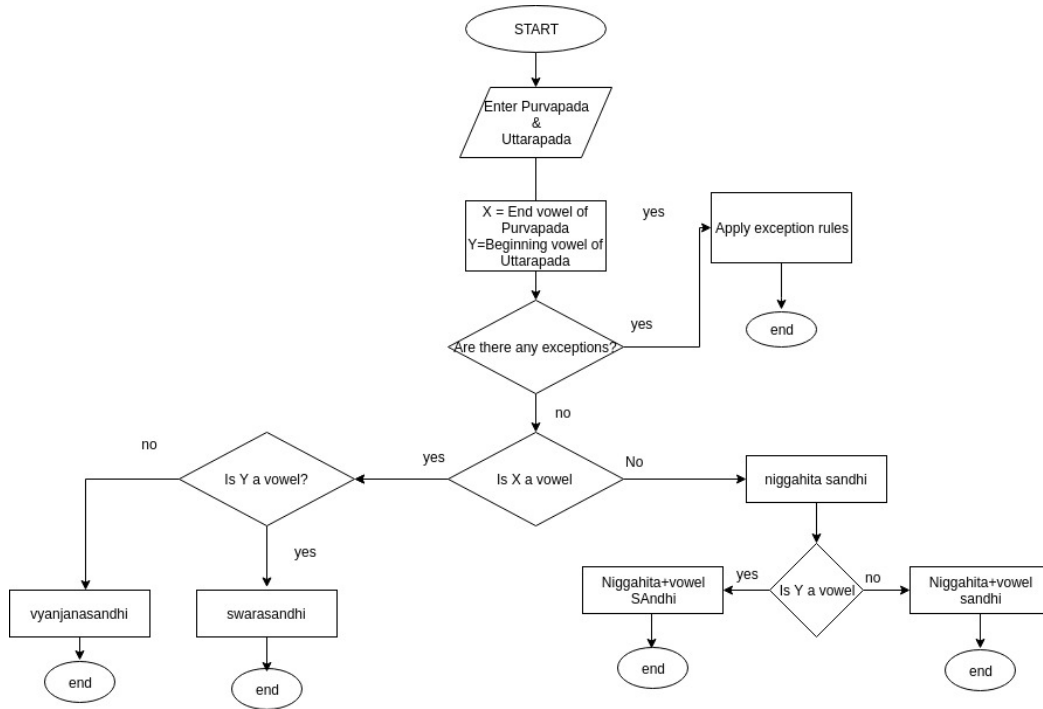
Input pūrva-pada and uttarapada

If exceptions exist then

Derive required output

¹⁴Muller E.(1962) 'A simplified grammar of Pāli Language', Trubner and company, London

Display output
 Exit program
 Assign X as ending varna(character) of pūrvapada
 Assign Y as beginning letter of uttarapada
 If X and Y are vowels then
 Go to svarasandhi module
 Derive required output
 Display output
 exit the program
 If Y is vyañjana then
 Go to vyañjanasandhi module
 Derive required output
 Display possible output
 If X is niggahita then
 Go to niggahitasandhi module
 If Y is svara then
 Go to niggahita-svara module
 Derive required output
 Display output
 exit the program
 If Y is vyañjana then
 Go to niggahita-vyañjana module
 Derive required output
 Display output
 exit the program
 End



5.2 Statistics

For validating the tool, 398 sandhi examples are collected from various Pāli grammar and other texts. This data was put through the sandhi tool and compared with gold data. Following are the statistics of the output.

Total number of words	398
Total number of outputs matching atleast one gold data	356
svarasandhi	158
vyañjanasandhi	82
niggahitasandhi	158
apavāda	13
single output matching gold data	26
two outputs matching gold data	13
three outputs matching gold data	130
five outputs matching gold data	19
six outputs matching gold data	14
seven outputs matching gold data	76
eight outputs matching gold data	58
Nine outputs matching gold data	19
outputs not matching gold data	44

By examining the statistics, we notice that svarasandhi and niggahitasandhi are equal in number. Out of 398 words, 84% outputs had at least one output matching with gold data. We observed that multiple outputs are more in case of svarasandhi. Since our focus is on the complexities of sandhi rules, limited examples are taken for validation. More sandhi data will be analyzed later.

6 Scope for Future work

1. More examples from Pali literature have to be collected to validate the tool.
2. Exhaustive Statistical study of the Pali literature has to be undertaken to decide which sandhi rule is frequently applied to a given instance.
3. Pruning the outputs based on statistics.
4. Integrating with a dictionary to reduce multiple outputs.

7 Conclusion

Making a full-fledged sandhi splitter/joiner is a complex process due to the ambiguous sandhi rules. As seen by the results of Sandhi Joiner, for a given instance, there is a probability of multiple outputs. This is because of the nature of Pāli words and the complex nature of the grammatical rules. With the understanding of the nature of language, to prune the outputs, a wider study of literature is required.

References

- Tiwari, Laxminarayan and Sharma Birbal (1962) ‘Kacchayana vyakarana[Pāli Grammar]’, Tara Publications, Varanasi
- Duroiselle Charles (1921) A practical grammar of the Pāli Language. British Burma Press, Rangoon
- Muller E.(1962) ‘A simplified grammar of Pāli Language’, Trubner and company, London
- Childers, Robert Caesar (1875) “A dictionary of Pāli language”, Trubner and co., London
- Davids, Rhys and William Stede (1921-25) “Pāli – English Dictionary”, Pāli Text Society.
- D’Alwis, James (1863) “Introduction to kachchayanans grammar of the Pāli language”, Colombo

Kashyap, Jagadish Bhikshu (1940) “Pāli Mahavyakarana”, Mahabodhi Sabha, Saranath, Banaras.

Saddhatissa, (1949) “Sara Pāli Shiksha”, Mahabodhi society, Saranath, Banaras

Silananda, Venerable U “Pāli roots in Saddaniti”, e-book.

Tungar, Na Va (1939) “Pāli Bhasha Pravesha”, Samarth Bharat, Pune.

A Appendix - 1

Kaccāyana rules used in various sandhi modulewise

A.1 Case 1 : vowel + vowel

सरा सरे लोपं	१.२.१
वा परो असरूपो	१.२.२
क्वचासवण्ण लुत्ते	१.२.३
दीघं	१.२.४
पुब्बो च	१.२.५
यमेदन्तस्सादेसो	१.२.६
वमोदुदन्तानं	१.२.७
सब्बो चन्ति	१.२.८
दो धस्स च	१.२.९
इवण्णो यन्न वा	१.२.१०
एवादिस्स रि पुब्बो च रस्सो	१.२.११

A.2 Case 2 : vowel + consonant

सरा पकति व्यञ्जने	१.३.१
दीघं	१.३.३
रस्सं	१.३.४
लोपश्च तत्राकारो	१.३.५
परद्वेभावो ठाने	१.३.६
वग्गे घोसाघोसानं ततियपठमा	१.३.७

A.3 Case 3 : niggahita + vowel

मदा सरे	१.४.५
यवमदनतरळा	१.४.६
क्वचि लोप	१.४.९
व्यञ्जने च	१.४.१०
परो वा सरो	१.४.११
व्यञ्जनो च विसञ्जोगो	१.४.१२

A.4 Case 4 : niggahita + consonant

अं व्यञ्जने निग्गहीतं	१.४.१
वग्गन्त वा वग्गे	१.४.२

एहेय्यं	१.४.३
सये च	१.४.४

A.5 Case 5 : special sandhis

क्वचि ओ व्यञ्जने	१.४.७
निगहितञ्च	१.४.८
गो सरे पुथुस्सागमो क्वचि	१.५.१
पास्स चन्तो रस्सो	१.५.२
अब्भो अभि	१.५.३
अज्झो अधि	१.५.४
ते न वा इवण्णे	१.५.५
अतिस्स चन्तस्स	१.५.६
क्वचि पटि पतिस्स	१.५.७
पुथुस्स व्यञ्जने	१.५.८
ओ अवस्स	१.५.९
अनुपदिट्ठान वुत्तयोगतो	१.५.१०

B Appendix - 2

Screenshots of sample input and sample outputs are given below.

Figure 1: Input

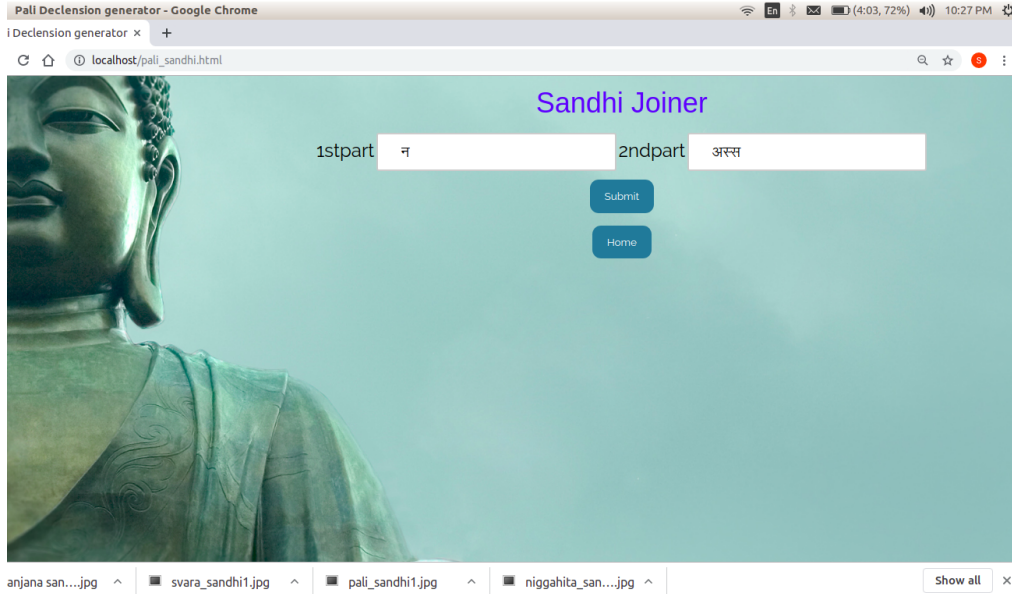
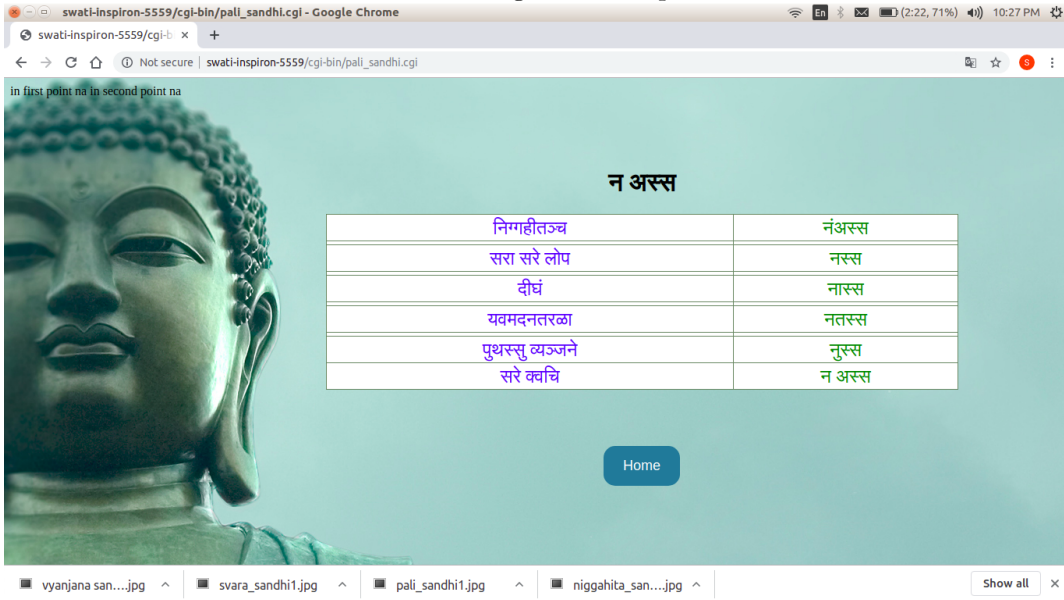


Figure 2: Output



in first point na in second point na

न अस्स

निग्गहीतञ्च	नंअस्स
सरा सरे लोप	नस्स
दीघं	नास्स
यवमदनतरळा	नतस्स
पुथस्सु व्यञ्जने	नुस्स
सरे ववचि	न अस्स

Home

vyanjana san...jpg svara_sandhi1.jpg pali_sandhi1.jpg niggahita_san...jpg Show all

