Segmentation of Navya-Nyāya Expressions

Arjuna S.R. & Amba Kulkarni
Department of Sanskrit Studies,
University of Hyderabad,
Hyderabad, India
arjunsr1987@gmail.com, apksh@uohyd.ernet.in

Abstract

Navya-Nyāya (NN), a school of Indian logic and philosophy, has evolved a sophisticated language to deal with verbal cognition, logic and epistemology. This language is known for its use of long compounds, productive use of secondary derivational suffixes, and a special technical vocabulary. In this paper we present a specially designed domain specific splitter to split the NN compounds into its components. The performance of this splitter is tested on a set of compounds from a Navya-Nyāya text. The result on the test data show a recall rate of 91%. While the average number of splits was around 50, in 75% cases the correct split was found to be the first one.

1 Introduction

Segmentation is an important task in NLP. Sanskrit being dominated by an oral tradition, most Sanskrit texts are written as a continuous stream of phonemes, without any explicit word or sentence boundaries. Moreover such a continuous stream of phonemes also undergoes phonetic changes at the juncture of word boundaries. This makes the task of splitting more complex.

There are noteworthy efforts in the field of Sanskrit segmentation in the past years. Hyman (2008) describes a Finite State Transducer (FST) for the Paninian sandhi rules. Huet (2009) has discussed the segmentation in Sanskrit in detail and has built an efficient Finite State Automata (FSA) based segmenter. Mittal (2010) describes two approaches; one using FST and the other one based on Optimality Theory, by defining the posterior probability function to choose among the valid splits. Kumar et al. (2010) used different posterior probability function and obtained better results. Natarajan and Charniak (2011) proposed sandhi splitting based on the Dirichlet process. Mittal, Kumar et al. and Natarajan and Charniak report the evaluation of their segmenters on a Sanskrit text corpus developed by the Sanskrit Consortium of India. This corpus does not contain very long sequences of characters. In the real corpus such as plays by Kālidāsa, strings containing more than hundred phonemes are very common. The possible segmentations of such strings run into millions. Huet and Goyal (2013) describe the design of a lean interface for displaying all such possible segmentation in a compact form.

Sandhi operation (phonological changes at the juncture of word boundaries) is just one reason behind the long strings of characters in Sanskrit. Rich derivational morphology that results due to possible recursion in the morphological process is another reason behind the long character sequences. This recursion is discussed in (Akshar Bharati, 2006) and (Kulkarni and Shukl, 2009). For quick reference, we produce the automaton showing the recursion in Sanskrit morphology in Figure 1.

As can be seen from this figure, theoretically the possible forms are infinite in number and owing to the finite size of the alphabet these forms can potentially be of infinite length. This figure does not show the compound formation in Sanskrit. Compound formation is typically binary where each component can in turn be a compound. The re-

D S Sharma, R Sangal and J D Pawar. Proc. of the 11th Intl. Conference on Natural Language Processing, pages 287–294,
Goa, India. December 2014.

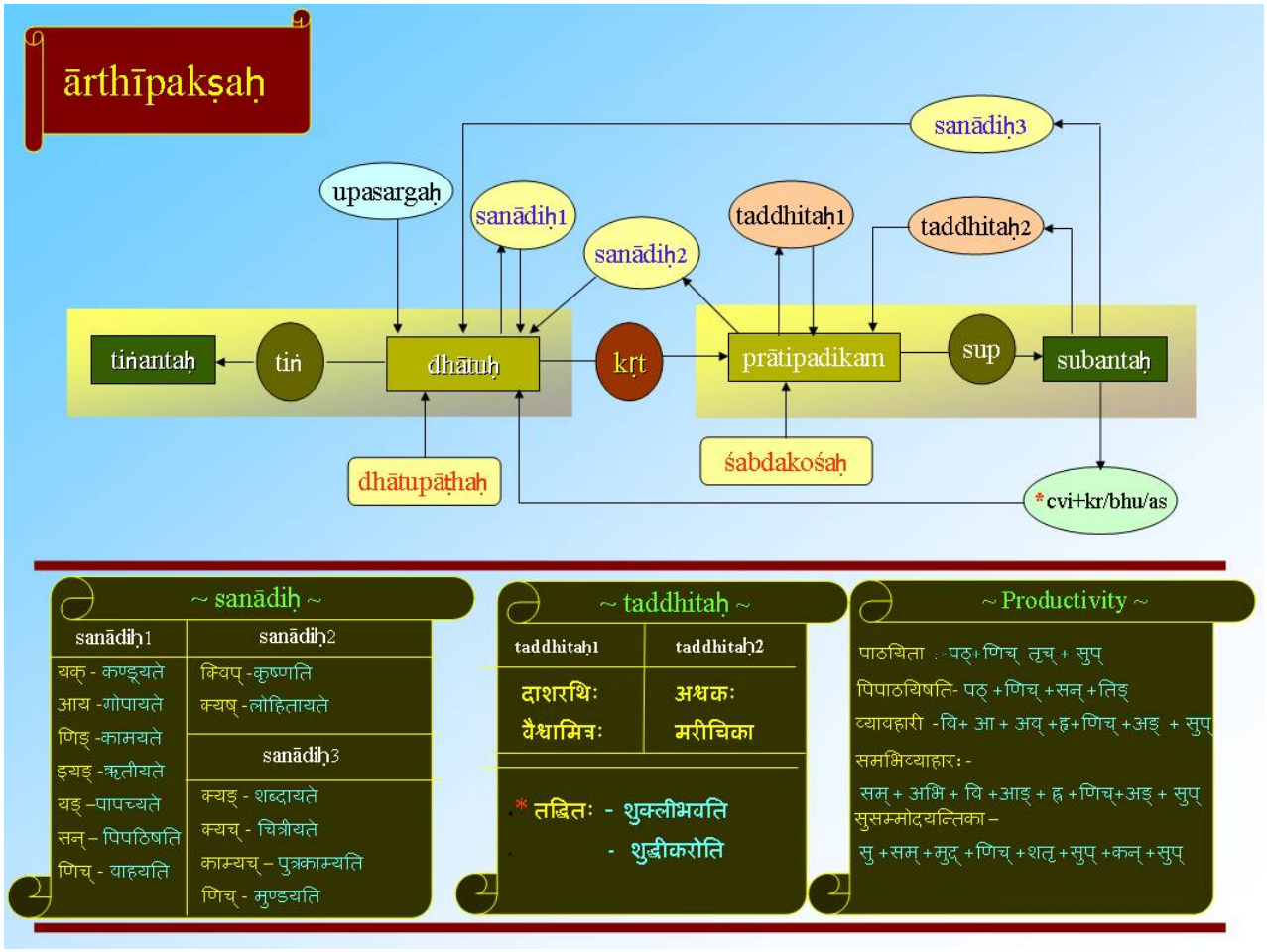

Figure 1: Recursion in Sanskrit Morphology

Table 1: Legends

dhātu (verbal root)	*dhātupātha*(list of verbal roots)	*kṛt*(non-finite verbal suffix)
samāsa(compound)	*sanādi*(derivational suffixes)	*śabdakoṣa*(lexicon)
subanta(noun)	*sup*(nominal suffix)	*taddhita*(secondary derivational suffix)
tiṅ(finite verbal suffix)	*tiṅanta*(finite verb form)	*upasarga*(verbal prefix)

cursion in the definition results in compounds of arbitrary long length. The Navya-Nyāya 'Neo-Logic' (NN) school of Indian tradition sees the culmination of productive compound formation in the form of compounds running through pages. The components of such compounds are typically formed with more than one *taddhita* (secondary derivational) suffixes. Such compounds also use the technical language of NN.[1]

Here is an example of linguistic expression in Navya-Nyāya (NNE) involving a compound with ten components:

samavāya-sambandha-avacchinna-gandhatva-avacchinna-gandha-niṣṭha-ādheyatā-nirūpita-adhikaraṇatāvatī.

For the sake of readability we have split the components, but in the printed texts this is written as a single word with underlying phonological changes as

samavāyasambandhāvacchinnagandhatvāva-cchinnagandhaniṣṭhādheyatānirūpitādhikara-ṇatāvatī.

In spite of a continuous stream of characters in-

[1]The technical language of NN consists of a few conceptual terms and it provides a mechanism to express the underlying cognitive structure corresponding to a linguistic expression in an unambiguous way.

volving arbitrarily long compounds, the cognitive structure being described by such an expression helps a human mind to understand them.

All the efforts related to segmentation described earlier had focused on general Sanskrit texts. But for much more complex and domain-specific inputs like NNE, which is known for long compounds, use of technical vocabulary, and productive use of secondary derivational suffixes (*taddhita*) a specially trained segmenter is needed.

In this work we present a segmenter specially designed for NNE taking into consideration the special vocabulary and secondary derivative suffixes NNE uses. In the next section we describe the earlier work on handling NNEs. In the third section we describe our approach for getting the desired splits on the top. The fourth section discusses the results of the experiment, followed by the conclusion.

2 Heritage segmenter for NNEs

The first attempt to develop a domain specific segmenter for NNE is reported in Arjuna and Huet (2014). A sample of NNEs was collected from the *Āloka* commentary on *Tarkasaṅgraha* (Varadacharya, 2007) and *Pañcalakṣaṇīsarvasvam* (Sastry, 2005). There were 49 NNEs from the *Āloka* commentary and 352 NNEs from *Pañcalakṣaṇīsarvasvam*. These compounds were split manually into the components, which formed the Gold data. We also extracted the possible combination of secondary derivational suffixes that are found in the selected texts. Figure 2 shows these possible combinations.

Arjuna and Huet (2014) summarize the difficulties in handling NNEs as follows.

1. Long compounds,
2. Technical vocabulary,
3. Productive use of *taddhita* suffixes, and
4. Semi-formal compound structure.

Heritage segmenter was enhanced to handle the first three of these. The salient features of the enhancement are

1. New databanks were added for the inflected forms of the *taddhita* suffixes viz. *-tal* (Fem), *-tva* (Neu), and *-matup* (in all three genders),
2. Technical vocabulary of Navya-Nyāya was acquired in the lexicon,
3. Segmenter transitions were added to accommodate *taddhita* productivity,
4. Word mode for single *pada* was used rather than sentence in order to curb overgeneration, and
5. Lean interface described in Huet and Goyal (2013) was used in order to share the huge solution space.

The recall of the segmenter after this enhancement was 91%.

There are three major problems with this segmenter. The first problem is with the number of solutions. For a typical NNE, this segmenter results with thousands and sometimes even millions of solutions. Typically the topmost row gives the most probable choice, and thus for a compound with n components, n choices by the user results in the proper split of the compound. Thus even if there are thousands of solutions, the user has to look for only a handful of choices. Hence this problem is not that serious.

The second problem with this segmenter is with the granularity. The Heritage segmenter is enhanced with the technical vocabulary of Navya-Nyāya. But still it splits many technical words into components. For example, *nirūpita*, *avacchinna*, *samānādhikaraṇa* etc. are split as *ni-rūpita*, *ava-chinna*, and *samāna-adhi-karaṇa* respectively. In order to understand the NNEs that uses their own specialised technical vocabulary with well-defined meanings, to get a broader picture of a NNE, a Naiyāyika(Indian Logician) prefers to hide the derivation of these technical terms, and would like to see these words as single units without any splits.

Thus while admitting the fact that the term *samānādhikaraṇa* is compositionally equal to *samāna-adhi-karaṇa*, or *vyadhikaraṇa* being compositionally equal to *vi-adhi-karaṇa*, these being technical terms, a *Naiyāyika* would like

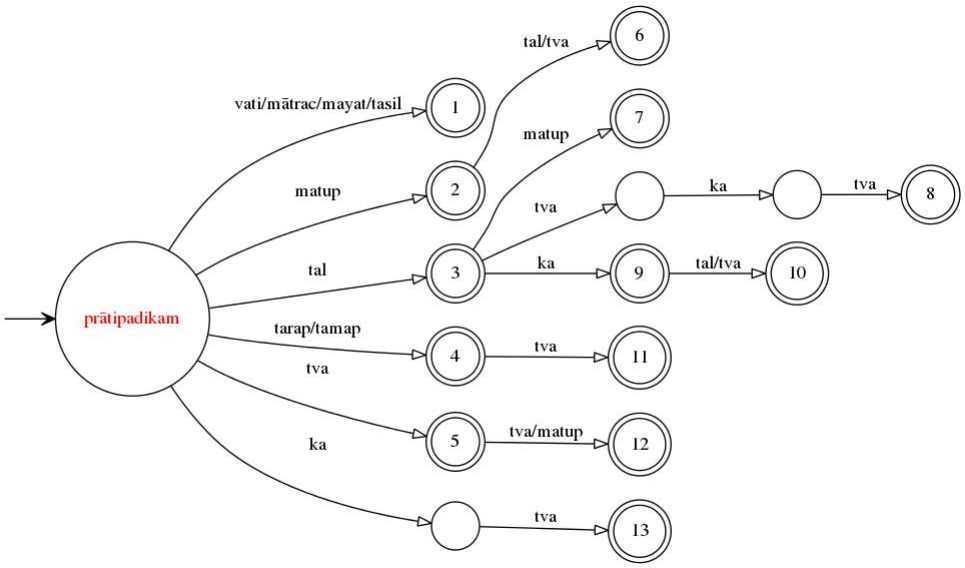

Figure 2: FSA showing possible *taddhita* suffixes in NNE

to look at them as a packaged entry with all the analysis hidden. Treating such technical words as a single unit would also result in lesser choices for user selection.

Finally while the user interface has its own advantages, one would like to reduce the user interaction as far as possible, pushing the correct solution to the top or preferably the first position for further automatic processing of such compounds.

The major focus of the work reported in this paper is on how to take advantage of the specialised vocabulary in order to prune out the bad solutions or push them down and push the better solutions to the top. This time we used Sanskrit morphological analyser developed at the University of Hyderabad, mainly because of more familiarity and easy accessibility. In the next section we describe our approach.

3 Our approach

As a first step we used the same domain specific corpus that was collected by Arjuna and Huet (2014) and enhanced the morphological analyser of University of Hyderabad to handle the *taddhita* suffixes reported in Figure 2. The 49 NNEs obtained from the *Āloka* commentary were used for the development purpose, and we set aside the 352 NNEs for testing purpose. All the collected NNEs were further analysed for their components. The statistics showed that there are a few nominal stems, which are not typical of NN, but occur frequently as a component in the NNEs. These stems are *artha*, *ātmaka*, *pūrvaka*, *vidha*, *kara* etc. which occur as a final component of a compound (*in fine compositi* or *samāsa-uttarapada*). Figure 3 shows the sequence of *taddhita* suffixes after which these stems occur.

We extended our morphological analyser to handle the derivational morphology – both the secondary derivations as well as frequent compounds shown in Figures 2 and 3 respectively. We also extended our lexicon with the technical terms in Navya-Nyāya.

The treatment of some *taddhita* suffixes such as '*ka*' and the treatment of *kṛt* suffixes (primary derivatives) in compound formation need special mention. The *taddhita* suffix '*ka*' results in a *bahuvrīhi* (exo-centric) compound. Such compounds, since have their head external to the compound, they are more like adjectives, and thus can decline in all the three genders. For example, if 'the one which has smoke as the cause' refers to a masculine referent, it will have the form *dhūmahetuka*, and if the referent is a feminine, it will have the form *dhūmahetukā*. But such words when occur as a component of a compound as iics (*in initio compositi* or *samāsa-pūrvapada*), they will always undergo an

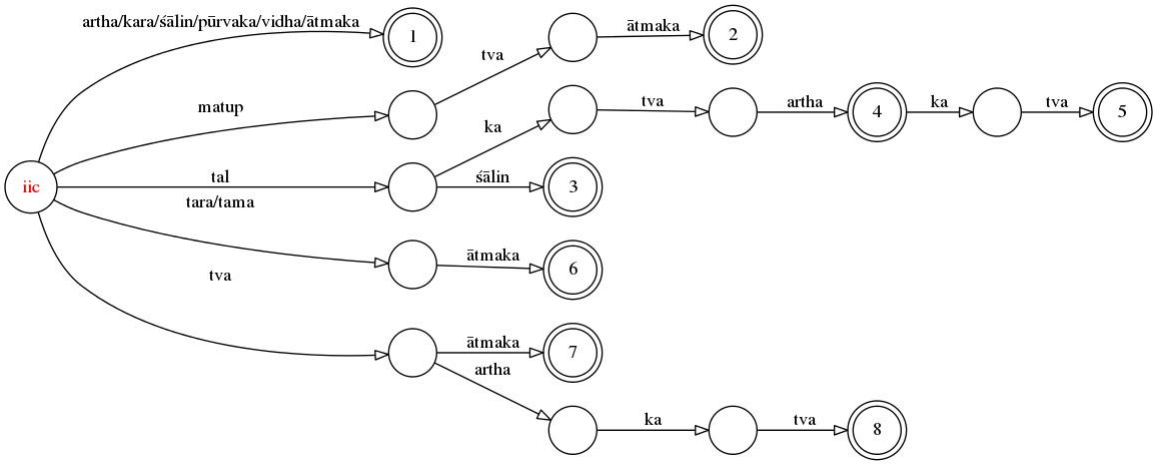

Figure 3: ifcs(*in fine compositi* or *samāsa-uttarapada*) found in NNEs

operation of **puṁvadbhāva**[2] resulting into a word in masculine gender. Thus we get *dhūmahetuka* as an iic form, whatever be the object it refers to.

Similarly, the adjectives formed by the non-finite suffixes (*kṛt* suffixes) such as *ṇvul* (in the sense of an agent), or *kta* (in the sense of an object) also take the base form[3] when they occur in the compounds as iics. For example, a compound 'a lady cook' will be *pācakastrī*, and not *pācikāstrī*.

The Paninian *sūtra* that governs the formation of such compounds is *puṃvatkarmadhārayajātīyadeśīyeṣu* 6.3.42. We have enhanced our morphological analyser to take into account this phenomenon.

With this we were able to split all 49 examples from the development data. The possible splits in each case were in thousands and in 3 cases even in tens of thousands. This was mainly because, though the technical terms were available in the lexicon, machine in addition to this lexical term, also showed all possible splits of such words. For example, for *sambandhāvacchinna* is split as *sambandha+avacchinna*, and also as *sambandha+ava+chinna*, and so on. The second one should be pruned out since in the context of Navya-Nyāya, *avacchinna* being a technical word need not be split further. So we needed a splitter that discards splits of morphologically analysable long words. We describe below the algorithm of this splitter followed by its performance on the development data as well as the test data.

3.1 SCL-NN splitter

The main aim of this algorithm is to reduce the over-generation ensuring that the imposed conditions do not under-generate, and at the same time push the most preferred solution to the top of the possible solutions. The over-generation and the under-generation are measured only with respect to the NN vocabulary. Thus a split which is an over-generation from the NN point of view, may be a genuine split in the classical Sanskrit. The salient features of the algorithm are stated below.

1. The sandhi rules are of the form $u \rightarrow v + w; f$, where f indicates the number of cases this rule was observed in the Sanskrit Consortium Corpus. Even if u is just a concatenation of v and w, without any underlying phonetic change, then also we treat it as a sandhi rule, in order to use the frequency information.

   The splitter scans the string from the left and looks for the longest match each time. At each juncture, typically more than one sandhi rules are available. In case there are more than one applicable rules, the one with longer u is preferred over the smaller ones, and in case of two rules with matching u of

[2]Condition for *puṁvadbhāva* is given in Paninian *sūtra* 6.3.42. It says - In *Karmadhāraya* compound and in those cases where the second component of a compound ends in a *jātīya* or *deśīya* suffix, the word in feminine gender will assume the *bhāṣitapuṃska* 'expressed as a masculine' form.

[3]The technical term for such base forms in Sanskrit is the one with *puṁvadbhāva*

equal length, the one with higher frequency is chosen.

For example, consider a string *adhikaraṇatānirūpaka*. There are two possible splits for this viz. *adhikaraṇatā* + *nirūpaka*, and *adhikaraṇatā* + *anirūpaka*. The first split corresponds to a split rule *ān* $\rightarrow \bar{a} + n$, which involves a window of two phonemes. The second split corresponds to the split rule $\bar{a} \rightarrow \bar{a} + a$ which involves a context of only one phoneme. The preference for two phoneme rule produces the split *adhikaraṇatā* + *nirūpaka* before other split *adhikaraṇatā* + *anirūpaka*. Thus we ensure that the most likely output appears before the other solutions.

There are four ways in which $\bar{a}$ can be split, viz. *a*+*a*, *a*+$\bar{a}$, $\bar{a}$+*a*, and $\bar{a}$+$\bar{a}$, with frequency of occurrence in the Sanskrit Consortium corpus as 3413, 2072, 350 and 233 respectively. Machine uses these rules in the decreasing order of frequency to ensure that the most probable one is reported first.

2. Preference is given to the NN vocabulary over others. The expression *avacchinnakāryatā* is wrongly split as *avacchinnaka*+*āryatā* as the more preferred one rather than the correct split *avacchinna*+*kāryatā*, because greedy match prefers the longest component in the beginning. As we notice, the phoneme sequence '*ka*' can be potentially a *taddhita* suffix of the first component as well as an initial sequence of a NN technical vocabulary. We resolve such conflicts in favour of the NN technical vocabulary.

3. The splitting is done recursively following the depth first search. The boundaries at which the string is split and the split rule used are remembered. The string is not split twice at the same place with the same split rule. This is to avoid the further splitting of bigger components, and thereby increasing the precision. For example, a string *pratiyogitānirūpaka* is split as *pratiyogitā*+*nirūpaka* with a rule *ān* $\rightarrow$ $\bar{a}$+*n*, and as *pratiyogitā*+*anirūpaka* with a rule $\bar{a} \rightarrow a$+*a*. But the string *pratiyogitā* is not split further as *prati*+*yogitā*, nor is *nirūpaka* as *ni*+*rūpaka*.

4. The treatment of *puṁvadbhāva* in the derivational morphology of compounds help in pruning out the wrong splits such as *niṣṭhā*+*ādheyatā* for *niṣṭhādheyatā*. *Puṁvadbhāva* ensures that we get only the valid split *niṣṭha*+*ādheyatā*.

5. A split is considered to be an over-generation if it does not contain any NN technical term.

4 Analysis of the Result

Our aim was to improve the precision and also get the correct solution to the top of the list. We first discuss the precision and recall.

4.1 Precision and Recall

We tested 49 NNEs collected from *Āloka* commentary of *Tarkasaṅgraha* on both the Heritage splitter as well as the SCL-NN splitter. The number of possible splits produced by both these splitters is reported in Table 2 and Table 3.

As is obvious from the tables, the number of solutions is reduced drastically, increasing the precision.

The result of the test data of 352 examples (see table 4) from *Pañcalakṣaṇīsarvasvam* also confirms that the new algorithm prunes out all irrelevant splits. The recall is around 91%, which is as good as the recall of Heritage splitter, and at the same time the number of solutions is reduced substantially, increasing the precision almost 100 times.

4.2 Correct Solution

We compared all the generated solutions with the manually tagged Gold data. The table 5 shows

No of Solutions	No of Cases
0-100	10
101-1,000	15
1,001-100,000	18
$>$ 100,000	5
Time-out	1
Total	49

Table 2: Number of solutions of Heritage Splitter

No of Solutions	No of Cases
0-5	14
6-10	11
11-100	18
101-1000	5
$>$ 1000	1
Total	49

Table 3: Number of solutions of SCL-NN Splitter

No of Solutions	No of Cases	Percentage
0-5	196	55.7
6-10	56	15.9
11-100	72	20.4
101-1000	13	3.6
$>$ 1000	3	1
No Split	12	3.4
Total	352	100

Table 4: Number of solutions of SCL-NN Splitter

the number of cases corresponding to the position of the correct solution among the ones produced. In 42 cases, the first solution produced by the machine was the correct one. Later we tested examples from *Pañcalakṣaṇīsarvasvam*. The results are shown in the table 6.

Here is a sample input consisting of 319 phonemes and the first solution with 40 components, which happens to be the correct one.

Input:
sādhyatāvacchedakasambandhāvacchinna-sādhyatāvacchedakāvacchinnapratiyogitāka-sādhyābhāvatvaviśiṣṭhanirūpitasādhyatāvacchedakasambandhāvacchinnasādhyatāvacchedakāvacchinnapratiyogitākasādhyābhāvavṛttisādhya-sāmānyīyapratiyogitvatadavacchedakatvānyatarāvacchedakasambandhāvacchinnanirūpakatā-nirūpitaniravacchinnādhikaraṇatāśrayavṛttitva-sāmānyābhāvaḥ

Position	No. of Cases	Percentage
1	42	86
2	2	4
3	4	8
7	1	2
Total	49	100

Table 5: Position of the correct solution in the Development data

Position	No. of cases	Percentage
1	264	75
2-5	42	11.9
6-10	6	1.7
11-100	7	2.0
$>$101	2	0.6
No Split	12	3.4
No-correct solution	19	5.4
Total	352	100

Table 6: Position of the correct solution in the test data

Output:
sādhyatā- avacchedaka- sambandha- avacchinna-sādhyatā- avacchedaka- avacchinna- pratiyogitāka- sādhya- abhāvatva- viśiṣṭha- nirūpita-sādhyatā- avacchedaka- sambandha- avacchinna-sādhyatā- avacchedaka- avacchinna- pratiyogitāka- sādhya- abhāva- vṛtti- sādhya-sāmānyīya- pratiyogitva- tad- avacchedakatva-anyatara- avacchedaka- sambandha-avacchinna-nirūpakatā- nirūpita- niravacchinna-adhikaraṇatā- āśraya- vṛttitva- sāmānya-abhāvaḥ

5 Conclusion

Thus with the help of domain specific words, greedy approach in selecting the long components, avoiding alternative splits of an already split segment, and selecting the more frequent split rule over the less frequent one, and ensuring that the solution thus produced has at least one NN term in it, we could increase the precision, without compromising the recall, and also we could push the correct solution to the top of the list.

While building this splitter, we took an advantage of the fact that the NNEs are unambiguous. The algorithm does not allow more than one splits at the same position. Thus if a string w is split as $w_0 + w_1 + w_2 + w_4$, it can not be split again as $w_0 + w_5 + w_6 + w_4$. This splitter therefore, can not be used to split the strings that are ambiguous, since it does not allow two different ways of splitting a substring.

References

[Akshar Bharati2006] V Sheeba Akshar Bharati, Amba P Kulkarni. 2006. Building a wide coverage morphological analyser for sanskrit: A practical approach. First National Symposium on Modeling and Shallow Parsing of Indian Languages,IIT Mumbai.

[Arjuna and Huet2014] S. R. Arjuna and Gérard Huet. 2014. Semi-automatic analysis of Navya-Nyāya compounds, SALA-30, Hyderabad. SALA-30, University of Hyderabad.

[Holz and Biemann2008] Florian Holz and Chris Biemann. 2008. Unsupervised and knowledge-free learning of compound splits and periphrases. In *CICLing*, pages 117–127. Lecture Notes in Computer Science - Springer.

[Huet and Goyal2013] Gérard Huet and Pawan Goyal. 2013. Design of a lean interface for Sanskrit corpus annotation. In Dipti Mishra Sharma, Rajeev Sanghal, Karunesh Kr.Arora, and B.K.Murthy, editors, *Proceedings of ICON-2013*, pages 177–186.

[Huet2009] Gérard Huet. 2009. Sanskrit Segmentation, South Asian Languages Analysis Roundtable xxviii, Denton, Texas. South Asian Languages Analysis Roundtable XXVIII.

[Hyman2008] Malcolm D. Hyman. 2008. From Paninian Sandhi to Finite State Calculus. In *Sanskrit Computational Linguistics*, pages 253–265.

[Koehn and Knight2003] Philipp Koehn and Kevin Knight. 2003. Empirical Methods for Compound Splitting. In *Proceedings of the Tenth Conference on European chapter of the Association for Computational Linguistics - Volume 1*, pages 187–193. Association for Computational Linguistics.

[Kulkarni and Shukl2009] Amba Kulkarni and Devanand Shukl. 2009. Sanskrit morphological analyser: Some issues. *Indian Linguistics*, 70(1-4):169–177. in the Festscrift volume of Bh. Krishnamoorty.

[Kumar et al.2010] Anil Kumar, Vipul Mittal, and Amba.P.Kulkarni. 2010. Sanskrit Compound Processor. In *Sanskrit Computational Linguistics*, pages 57–69.

[Macherey et al.2011] Klaus Macherey, Andrew M. Dai, David Talbot, Ashok C. Popat, and Franz Och. 2011. Language-independent Compound Splitting with Morphological Operations. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, pages 1395–1404. Association for Computational Linguistics.

[Mittal2010] Vipul Mittal. 2010. Automatic Sanskrit Segmentizer Using Finite State Transducers. In *ACL (Student Research Workshop)*, pages 85–90.

[Natarajan and Charniak2011] Abhiram Natarajan and Eugene Charniak. 2011. S^3 - Statistical Sandhi Splitting. In *IJCNLP*, pages 301–308.

[Sastry2005] Sriram Sastry. 2005. *Pañcalakṣaṇī-sarvasvam*. Bharatiya Vidya Sansthan, Varanasi.

[Varadacharya2007] Varadacharya. 2007. *Tarka-saṅgraha with Āloka commentary*. Arya Grantha Prakashan, Mysore.