$\mathcal{S}^3$ - Statistical Saṃdhi Splitting

Abhiram Natarajan and Eugene Charniak
Brown Laboratory for Linguistic Information Processing
Brown University, USA
abhiram.nat@gmail.com, ec@cs.brown.edu

Abstract

The problem of Saṃdhi-Splitting is central to computational processing of Sanskrit texts. Currently the best-known algorithm for this task, given a chunk, generates all possible splits and chooses the Maximum-a-Posteriori estimate as the final answer. Our contributions to the task of Saṃdhi-Splitting are two-fold. Firstly, we improve upon the current algorithm by proposing a principled modification of the posterior probability function to achieve better results. Secondly, we propose an algorithm based on Bayesian Word-Segmentation methods. We find that the unsupervised version of our algorithm achieves a better precision than the current algorithm with the original probabilistic model. We then present a supervised version of our algorithm that outperforms all previous methods/models.

1 Introduction

Sanskrit, considered to be among the oldest languages in the world, is elaborate in its oral specifications. It possesses a set of euphonic rules called Saṃdhi[1] rules, which when applied, cause phonological changes at word or morph boundaries. These rules are enumerated in Pāṇini's Aṣṭādhyāyī, and have been devised with the aim of achieving epigrammatic brevity. There are two kinds of Saṃdhi:

1. Internal Saṃdhi: Euphonic transformation at morph boundaries; $W_1x + yW_2 \Rightarrow W_1zW_2$, where x and y are the final and initial segments of W_1 and W_2, respectively, and z respresents the smoothed phonetic transformation of x and y together
   E.g: dīpena udvejayati ⇒ dīpenodvejayati
2. External Saṃdhi: Phonetic change at word boundaries; $W_1x + W_2 \Rightarrow W_1x' + W_2$, where the final segment of the first word x changes to x'
   E.g: utthitaḥ vidyādharaḥ ⇒ utthito vidyādharaḥ

We shall refer to the process of making the euphonic transformations as *Saṃdhising* and the text formed as a result as *Saṃdhied text*. The process of undoing the euphonic transformations shall be referred to as *Analysis* or *Saṃdhi-Splitting*[2], and the text formed as a result will be referred to as *Analysed text*.

It is easy to see that the task of *Saṃdhi-Splitting* is ambiguous. For instance, any of {(ca,api), (cā,api), (ca,āpi), (cā,āpi)} could have combined to form cāpi. Contextual knowledge is necessary for perfect *analysis* of Sanskrit text (Hellwig, 2009), and current methods have not evolved enough to be able to supply context. However the field of Statistical Machine Learning, by making reasonable assumptions, allows us to provide approximations of these processes. Previous work (Beesley, 1998; Hyman, 2007) indicates that Finite State Transducers (FST) could be used to generate morphologically valid splits. Mittal (2010) considers the FST approach as well as an approach based on Optimality Theory (Prince and Smolensky, 1993), by defining a posterior probability function to choose among all morphologically valid splits of a chunk (OT1). More recently, Kumar et al. (2010) report findings using a different posterior probability function with the same Optimality Theory approach (OT2).

Firstly, we derive our own posterior probability function, which is different from OT1 and OT2.

[1] Literally means "Putting Together".

[2] We shall use these terms interchangeably.

Proceedings of the 5th International Joint Conference on Natural Language Processing, pages 301–308,
Chiang Mai, Thailand, November 8 – 13, 2011.

We observe better results. Secondly, and more importantly, we present a *Saṃdhi-Splitting* technique based on the Bayesian Word-Segmentation methods presented by Goldwater et al. (2006a). These methods in turn are based on the Dirichlet process. The Dirichlet process is a continuous multivariate distribution used in nonparametric Bayesian Statistics, often as a convenient prior distribution. Gibbs sampling is used to sample from the posterior distribution of *analysed* texts given the *Saṃdhied text*.

The paper is organised as follows. In Section 2, we discuss the Optimality Theory approach in detail. In Section 3, we introduce our framework for *Saṃdhi-Splitting*. We present both the unsupervised and supervised versions of our algorithm in this section. Section 4 then provides specific details of our implementations and Section 5 contrasts the results obtained by all the methods/models considered. Section 6 concludes the paper.

2 Current Methods

The literature on *Saṃdhi-Splitting* is too huge for us to do justice. We shall stick to the best published results so far, which have been obtained by the OT method (Kumar et al., 2010; Mittal, 2010). Procedure 1 is a pseudocode representation of the OT method.

Procedure 1 : Saṃdhi-Splitting using OT

1: **for each** $chunk \in \mathcal{D}$ **do**
2: $\mathbb{S} \leftarrow$ getAllPossibleSplits($chunk$)
3: $\mathbb{S}' \leftarrow \{x : x \in \mathbb{S}, x$ is morphologically valid$\}$
4: $l \leftarrow \arg\max_s \{\hat{P}(s) : s \in \mathbb{S}'\}$
5: **print** l

The procedure is quite straightforward. Each candidate chunk is recursively broken (Line 2) to generate all possible splits. Each split is passed through a morphological analyser and the splits that contain one or more invalid morphs are discarded (Line 3). Finally, the Maximum a posteriori (MAP) estimate is chosen as the answer (Line 4).

$\hat{P}(s)$ is the posterior probability function, which we derive as follows. Consider a morphologically valid split $s =< c_1, \ldots, c_m >$, where $c_1 \ldots c_m$ are its constituents, which was obtained after applying rules $r =< r_1, \ldots, r_{m-1} >$[3] on a chunk $\mathcal{C}$. We would need to find the most probable split, which is given by $\arg\max_{s \in \mathbb{S}'} P(s|\mathcal{C})$. Using the noisy channel model framework (Shannon, 1948), we get

$$\arg\max_{s \in \mathbb{S}'} P(s|\mathcal{C}) = \arg\max_{s \in \mathbb{S}'} P(\mathcal{C}|s) \times P(s) \quad (1)$$

We know that s is a result of applying r on $\mathcal{C}$ at specific points. Now, $P(\mathcal{C}|s)$ reads as the conditional probability of obtaining $\mathcal{C}$ given that r was applied at exactly those points. Clearly, this is 1. Thus the MAP estimate would be a split that maximises the probability of the prior

$$l = \arg\max_{s \in \mathbb{S}'} \hat{P}(s) \quad (2)$$

Note that we use the notation $\hat{P}$ instead of P because we can never know the true value of $P(s)$, but can only estimate it from our training set. Let us now turn our attention to $\hat{P}(s)$. We read $\hat{P}(s)$ as the probability of generating the morphs $< c_1, \ldots, c_m >$. Thus we get:

$$\begin{aligned} \hat{P}(s) &= \hat{P}(c_1) \times \hat{P}(c_2 \,|\, c_1) \times \hat{P}(c_3 \,|\, c_1, c_2) \times \ldots \\ &= \prod_{j=1}^{m} \hat{P}(c_j) \qquad \text{[Assuming Independence]} \end{aligned} \quad (3)$$

Equation 3 describes a generative model in that it shows us how we can generate the morphs $< c_1, \ldots, c_m >$. Mittal (2010) defines $\hat{P}(s)$ as

$$\frac{\prod_{i=1}^{m-1} \left(\hat{P}(c_i) + \hat{P}(c_{i+1})\right) \times \hat{P}(r_i)}{m} \quad (4)$$

Kumar et al. (2010) define $\hat{P}(s)$ as

$$\frac{\left(\prod_{i=1}^{m} \hat{P}(c_i)\right) \times \left(\prod_{j=1}^{m-1} \hat{P}(r_j)\right)}{m} \quad (5)$$

In each of these models, the maximum likelihood estimators for $\hat{P}(c_i)$ and $\hat{P}(r_j)$ are used, i.e. $\hat{P}(c_i)$ is set to the relative frequency of c_i in the training set, and $\hat{P}(r_j)$ to the relative frequency of the usage of r_j in the training set.

[3] Rule r_i is applied between c_i and c_{i+1}

Note that unlike Equation 4 and Equation 5, Equation 3 does not have rule probabilities. This can be explained by keeping in mind that Equation 3 describes a generative model. If we generated morphs and knew that they had to be combined to form a chunk, the rules are uniquely determined. In Section 5, we shall compare the results obtained by using each of Equations 3, 4 and 5 on standard datasets.

3 *Saṃdhi*-Splitting based on the Dirichlet Process

Our method is similar to the two-stage modelling framework described by Goldwater et al. (2006a; 2006b). The framework has two components, a morph *generator* which generates morphs likely to be found in a lexicon, from some probability distribution, and an *adaptor* which determines how often each of these morphs occur. Section 3.1 and Section 3.2 describe the unsupervised version of our algorithm while Section 3.3 describes the supervised version.

3.1 Foundations

In the realm of the framework, a *saṃdhied* chunk $\mathcal{C} = c_1 \ c_2 \ \ldots \ c_m$ can be viewed to be created as follows:

1. A *generator* P_s generates a super-set sequence of morphs[4]: $\mathbb{M} = M_1 \ldots M_n$ from a probability distribution P_s.

2. The *adaptor* P_γ generates a sequence of integers: $\mathbb{Z} = z_1 \ldots z_m$, each of which are identifiers of one particular item from $\mathbb{M}$. This means that $1 \leq z_i \leq n$, and $z_i = x \Rightarrow c_i = M_x$.

Once the morphs of a chunk are generated, *Saṃdhi* rules are applied between them to form $\mathcal{C}$. Henceforth $TwoStage(P_\gamma, P_s)$ shall be used to denote a two-stage framework with P_γ as the adaptor and P_s as the generator. Let us move to the Chinese Restaurant Process, which we use as our *adaptor*.

[4]In the case of *Internal Saṃdhi*, we consider morph boundaries and in the case of *External Saṃdhi*, we consider word boundaries. The theoretical foundations hold good for both cases. Thus the reader must always keep in mind that the concepts presented in this section apply at word boundaries too.

3.1.1 Chinese Restaurant Process

Having studied many corpora of natural language utterances, Zipf (1932) made a famous empirical observation that word frequencies follow a power-law distribution, i.e., the frequency of a word is inversely proportional to its frequency rank. This means that the most frequent word in a corpus occurs twice as often as the second most frequent word, thrice as often as the third most frequent word, and so on. Mathematically speaking, if w_r is the r-ranked word in a corpus, then

$$f(w_r) \propto \frac{1}{r^c} \tag{6}$$

where $f(w_r)$ denotes the frequency of occurrence of w_r, and $c \approx 1$. The veracity of this law has hence been verified by a study on present-day English (Kucera and Francis, 1967).

Typically, power-law distributions are produced by stochastic processes in which outcomes accrue probability based on the probability they already have. Such processes are called preferential attachment processes. Let us turn our attention to the Chinese Restaurant Process (Aldous et al., 1985) (CRP), which is one such process. The process is best explained by specifying how to draw a sample from it. Consider a restaurant with an infinite number of tables, each with infinite capacity. Customers enter the restaurant, one at a time, and seat themselves at a table of their choice. They choose an occupied table with probability proportional to the number of people already present at the table, or an unoccupied table with probability proportional to some real-valued scalar parameter α. The first customer always sits at the first table. Then onwards, the i^{th} customer sits at a table T_i, and T_i follows the distribution:

$$P\left(T_i = k \,|\, \text{Previous Customers}\right) = \begin{cases} \frac{n_k}{i-1+\alpha} & 1 \leq k \leq N_{i-1} \\ \\ \frac{\alpha}{i-1+\alpha} & k = N_{i-1}+1 \end{cases} \tag{7}$$

where n_k is the number of people occupying table k and N_{i-1} is total number of tables occupied by the previous $(i-1)$ customers.

When the CRP is combined with a morph *generator*, it can be used to generate a power-law distribution over morphs. Such a model can be described as $TwoStage(CRP(\alpha), P_s)$. We could

view this as restaurant where each table represents a morph. Every customer is also labelled with a morph and can either sit only at tables which are labelled with the same morph, or at an unoccupied table. A customer at a table represents one occurence of the morph that is represented by the table. To get the probability distribution of the i^{th} morph, we must sum over all the tables labelled with that morph:[5]

$$\begin{aligned} &P\left(c_i = c \mid c_{-i}\right) \\ &\quad = P(\text{assign customer to any of the } c \text{ tables}) \\ &\qquad + P(\text{assign customer to a new table}) \\ &\quad = \frac{n_c^{c_{-i}}}{i-1+\alpha} + P_s(c) \times \frac{\alpha}{i-1+\alpha} \\ &\quad = \frac{n_c^{c_{-i}} + \alpha \times P_s(c)}{i-1+\alpha} \end{aligned} \tag{8}$$

where $c_{-i} = c_1 \ldots c_{i-1}$ and $n_c^{c_{-i}}$ represents the number of occurrences of c in c_{-i}.

Let us examine Equation 8. It is in accordance with the principle of *preferential attachment*. This is because the probability of generating a morph that has already been generated increases as more instances of the morph are observed. Also note that the sparseness of the distribution generated increases with an increase in the value of the parameter α. This is explained by the fact that $\alpha \times P_s(c)$ is constant and it reduces w.r.t. the first summand over time. What this means is that the probability of generating a novel morph decreases (but never dissapears fully) as more data is observed.

Let us now turn our attention to the *generator*. For simplicity, we choose a unigram phoneme distribution for P_s. If a morph c_i contains the phonemes $a_1 \ldots a_d$

$$P_s(c_i) = p_\Phi (1-p_\Phi)^{d-1} \prod_{j=1}^{d} P(a_j) \tag{9}$$

where p_Φ is the probability of a *Saṃdhi* rule being applied at any juncture. The reasoning for the equation is quite straightforward - the *Saṃdhi* rule is not applied $(d-1)$ times, and applied at the d^{th} phoneme. The unsupervised version of the algorithm uses an uniform distribution over phonemes.

[5]Note that two different tables may represent the same morph.

3.1.2 Dirichlet Process Model

Given a coin with an unknown bias, say p, what would be a suitable distribution that reflects our expectations about p? It would be the Beta distribution. The Dirichlet distribution is a generalisation of the Beta distribution, in that it may have more than just two dimensions. The Dirichlet process is an infinite dimensional version of the Dirichlet distribution. Each sample from a Dirichlet process returns a distribution over an infinite set of random variables.

Let us consider a Dirichlet process with its parameters as α and P_s. A sample from this process, say G, will consist of a set of all possible morphs and their corresponding probabilities. Now, we can generate each morph c_i in the corpus from the distribution G. This can be represented as follows:

$$\begin{aligned} c_i &\sim G \\ G &\sim DP(\alpha, P_s) \end{aligned} \tag{10}$$

Following the argument in Goldwater (2006), it is not hard to find that the model described by $TwoStage(CRP(\alpha), P_s)$ is equivalent to the model represented by Equation 10.

We must also keep in mind that, analogous to Equation 8, the probability of i^{th} morph $c_i = c$ is given by

$$P(c_i = c \,|\, c_{-i}) = \frac{n_c^{c_{-i}} + \alpha \times P_s(c)}{i-1+\alpha} \tag{11}$$

where $c_{-i} = c_1 \ldots c_{i-1}$, and $n_c^{c_{-i}}$ denotes the number of occurrences of c in c_{-i}. Going ahead a little, we realise that our input corpus is a set of sentences, where each sentence contains a set of *Saṃdhied* chunks, where each chunk is in turn formed by *Saṃdhising* separate morphs. The program does not need to separate the chunks from each other, all it needs to do is to separate a chunk into its constituent morphs. Referring to Equation 3, we recall our generative model view of how a *Saṃdhied* chunk is generated. We describe the process by the following PCFG:[6]

$$\begin{aligned} P(r_i = r_{branch} \,|\, r_{-i}): &\quad S \rightarrow S\ C \\ P(r_i = r_{end} \,|\, r_{-i}): &\quad S \rightarrow C \\ P_s(c_i = c \,|\, c_{-i}): &\quad C \rightarrow c \end{aligned}$$

We shall derive an equation for $P(r_i = r_{branch} \,|\, r_{-i})$ in the next section.

[6]Probabilistic Context-Free Grammar

3.2 Gibbs Sampling for *Saṃdhi* Analysis

As suggested in Gilks et al. (1996), we use Gibbs Sampling to sample from the posterior distrubition of *Saṃdhi* Analyses. One iteration of the algorithm causes the program control to consider every possible splitting point in the data and form two hypotheses, say $\mathcal{H}_1$ and $\mathcal{H}_2$. These hypotheses contain the same structure, except at the point of consideration. If the point of consideration is part of a chunk c, $\mathcal{H}_2$ splits c into two individual morphs (say c_1 and c_2) while $\mathcal{H}_1$ does not. Let $\mathcal{H}^{\cap}$ denote the structure common to both $\mathcal{H}_1$ and $\mathcal{H}_2$.

Given the way Gibbs sampling works, we know that we only need the relative probabilities of $\mathcal{H}_1$ and $\mathcal{H}_2$. Also, we must recall that the order of arrival of customers in the metaphorical Chinese Restaurant does not affect seating probability, as shown by Aldous et al. (1985). This means that we could consider $\mathcal{H}^{\cap}$ to have already been generated. From Equation 11 we get

$$\begin{aligned} P\left(\mathcal{H}_1 \,|\mathcal{H}^{\cap}\right) &= P\left(c \,|\, \mathcal{H}^{\cap}\right) \\ &= \frac{n_c^{\mathcal{H}^{\cap}} + \alpha P_s(c)}{|\mathcal{H}^{\cap}| + \alpha} \end{aligned} \tag{12}$$

We also derive

$$\begin{aligned} &P\left(\mathcal{H}_2 \,|\mathcal{H}^{\cap}\right) \\ &= P\left(r_{branch} \cup c_1 \cup c_2 \,|\, \mathcal{H}^{\cap}\right) \\ &= P\left(r_{branch} \,|\, \mathcal{H}^{\cap}\right) \times \frac{n_{c_1}^{\mathcal{H}^{\cap}} + \alpha P_s(c_1)}{|\mathcal{H}^{\cap}| + \alpha} \\ &\quad \times \frac{n_{c_2}^{\mathcal{H}^{\cap}} + I(c_1 = c_2) + \alpha P_s(c_2)}{|\mathcal{H}^{\cap}| + 1 + \alpha} \end{aligned} \tag{13}$$

The extra '1' in the denominator of the third item of the product is because after c_1, we have an additional morph. Also, I is the indicator function, which is 1 when the expression inside its parathesis is true. Let us now examine $P\left(r_{branch} \,|\, \mathcal{H}^{\cap}\right)$. Clearly, each time we can choose only one among $\{r_{branch}, r_{end}\}$, so we have $P(r_{branch}) + P(r_{end}) = 1$. The number of r rules applied at the point of consideration would be $(|\mathcal{H}^{\cap}| + 1)$ - one each for the $|\mathcal{H}^{\cap}|$ morphs, and an extra one for the chunk under consideration. We apply a Beta($\frac{\rho}{2}, \frac{\rho}{2}$) prior to get:[7]

$$P\left(r_{branch} \,|\, \mathcal{H}^{\cap}\right) = \frac{n_{r_{branch}}^{\mathcal{H}^{\cap}} + \frac{\rho}{2}}{|\mathcal{H}^{\cap}| + 1 + \rho} \tag{14}$$

[7] Henceforth, please substitute this expression in Equation 13

The algorithm works as follows - we begin with a random *analysis* of the corpus. After that, using Equations 12 and 13, we sample every potential boundary point. An iteration is said to have completed when the sampler samples over the entire corpus. We run multiple iterations of this process, until convergence.

Simulated annealing (Aarts and Korst, 1989) is used to facilitate choosing of low probability transitions early in the algorithm, lest it gets stuck at a local maximum.

3.3 Supervised Version

As mentioned earlier, we use Gibbs Sampling to sample from the posterior distribution. Let us recall the relevant equations - Equations 12 and 13. In the supervised version of our algorithm, we use available training data (say D_{train}) to our advantage. Firstly, we define $\mathcal{H}^{\cap'} = \mathcal{H}^{\cap} \cup D_{train}$.

Also, instead of assuming a uniform distribution over phonemes, we infer phoneme probabilities from D_{train}:

$$P_s'(c_i) = p_{\Phi}(1 - p_{\Phi})^{d-1} \prod_{j=1}^{d} \hat{P}(a_j) \tag{15}$$

Let Π denote the set of all phonemes found in D_{train}. Given a phoneme a_x, we set $\hat{P}(a_x)$ to be the maximum likelihood estimate:

$$\hat{P}(a_x) = \frac{n_{a_x} + \alpha_0}{n_* + \alpha_0 |\Pi|} \tag{16}$$

where n_{a_x} is the number of times a_x occurs in D_{train}, $n_* = \sum_{p \in \Pi} n_p$, and α_0 is the unigram smoothing constant.

Thus we get

$$P\left(\mathcal{H}_1 \,|\mathcal{H}^{\cap'}\right) = \frac{n_c^{\mathcal{H}^{\cap'}} + \alpha P_s'(c)}{|\mathcal{H}^{\cap'}| + \alpha} \tag{17}$$

and

$$\begin{aligned} &P\left(\mathcal{H}_2 \,|\mathcal{H}^{\cap'}\right) \\ &= \frac{n_{r_{branch}}^{\mathcal{H}^{\cap'}} + \frac{\rho}{2}}{|\mathcal{H}^{\cap'}| + 1 + \rho} \times \frac{n_{c_1}^{\mathcal{H}^{\cap'}} + \alpha P_s'(c_1)}{|\mathcal{H}^{\cap'}| + \alpha} \\ &\quad \times \frac{n_{c_2}^{\mathcal{H}^{\cap'}} + I(c_1 = c_2) + \alpha P_s'(c_2)}{|\mathcal{H}^{\cap'}| + 1 + \alpha} \end{aligned} \tag{18}$$

The supervised version of the algorithm now works exactly the same way as the unsupervised version, except that Gibbs sampling is done using Equations 17 and 18.

4 Implementation Details

As a part of the Consortium project in India, a corpus of nearly 25000 parallel strings of *Saṃdhied* and *analysed* text has been formed (say D_1). OT trains on the 25000 string dataset and tests on a separate dataset of 2148 *Saṃdhied* Chunks (say D_2). We see that the test set used in OT only contains examples of *Internal Saṃdhi*. Thus, we merge the test set with the original 25000 string dataset ($D = D_1 \cup D_2$) and generate random samples of training data and test data. We use $\frac{3}{4}$ of the data for training (D_{train}) and $\frac{1}{4}$ for testing (D_{test}). Note that D_{test} can have instances of *Internal Saṃdhi*, *External Saṃdhi* as well as no *Saṃdhi*[8] at all. The scores we report are averaged over 5 random samples of D_{train} and D_{test} from D.

We also use the morphological analyser (Akshar Bharati, 2006) provided by the apertium group.

As for the parameters of the model, we fixed $\rho = 2$. We found that a decrease in p_Φ improved recall, but caused long words to get concatenated. At the same time higher values of p_Φ tended to cause over segmentation. It was also observed that higher α values resulted in higher lexicon recall, once again only up to a point. Although it was not possible to arrive a single best value for either α or p_Φ; we fixed them to be $p_\Phi = 0.5$ and $\alpha = 20$. The sampler is annealed with the following temperature schedule - $\frac{1}{\gamma} = 0.1$ to 1.0, in steps of 0.1.

For evaluation, standard statistical measures of Precision, Recall and F-Score were used. Precision indicates how many among the items found are correct; Recall indicates how many among the correct items are found. F-score is the harmonic mean of the Precision and the Recall.

Our implementations work with the Sanskrit Library Phonetic Basic format (SLP1). Since the morphological analyser uses the Hyderabad-Tirupati format (WX), we use the transcoding tools on the Sanskrit Library Website (Scharf and Hyman, 2010). One may refer Huet (2009) for more details about the formats.

[8]This explains why our implementations of OT achieve lower scores than what is reported in their paper. Readers must note that our implementation of OT, when trained on D_1 and tested on D_2, achieved nearly the same accuracy reported in their paper. Thus we believe our implementation of OT is accurate.

5 Results

Let OT1, OT2, OT3 denote versions of Procedure 1, using Equations 4, 5, 3 as the posterior probability functions, respectively. Precision is calculated as the percentage of correct segmentations among all the segmentations made by the algorithm. Recall is calculated as the percentage of correct segmentations made by the algorithm among all the correct segmentations that need to be made. F-Score is the harmonic mean of the Precision and Recall. Table 1 shows the scores obtained by our implementations of different versions of OT.

Method	Precision	Recall	F-Score
OT1	54.61	62.08	58.11
OT2	63.96	68.74	66.26
OT3	**70.52**	66.61	**68.51**

Table 1: Accuracies obtained by different versions of OT; OT3 outperforms OT1 and OT2 in Precision and F-Score

As we can see, using Equation 3 as the posterior probability function while obtaining the MAP estimate, we improve the previous best F-Score by nearly 3.5%. We shall now examine the results obtained by four different versions of the framework described in Section 3. Let these versions be denoted by NC1 ... NC4:

1. NC1 denotes the Two-Stage Framework in its most basic state. The algorithm possesses no linguistic knowledge whatsoever.

2. Kessler (1994) shows that each Sanskrit syllable must have only one Sonority peak. NC2 uses this knowledge to effect. The method is largely similar to NC1 except for the way in which sampling is done. We only sample when we are sure that the segments we generate do not violate Sonority Hierarchy. Refer Appendix A for more details about sonority hierarchy.

3. NC3 uses a morphological analyser. Sampling is only done when we know that the

segment we are leaving behind is morphologically valid.

4. NC4 uses a morphological analyser as well as training data. Sampling is done using Equations 17 and 18.

Method	Precision	Recall	F-Score
NC1	31.74	41.22	35.86
NC2	50.98	40.43	45.1
NC3	66.55	56.13	60.9
NC4	**76.21**	**64.84**	**70.07**

Table 2: Accuracies obtained by different versions of NC; NC4 Outperforms NC1...3, as well OT1...3 in Precision and F-Score

Results obtained by these versions are shown in Table 2. NC4 improves upon the F-Score achieved by OT2 by nearly 6%.

6 Conclusion

In this paper, we have presented an algorithm for *Saṃdhi*-Splitting which draws on expedients of Bayesian statistics. It is highly flexible in that it works even in the absence of (a) morphological analyser (b) training data. Also, one must remember that an entire corpus, as it is, can be fed to the algorithm, as opposed to OT which would require each *Saṃdhied* chunk separately. Finally, the fact that the algorithm runs in polynomial time, as opposed to OT which takes exponential time, further asserts the efficacy of the method.

References

Emile Aarts and Jan Korst. 1989. *Simulated annealing and Boltzmann machines: a stochastic approach to combinatorial optimization and neural computing*. John Wiley & Sons, Inc., New York, NY, USA.

V Sheeba Akshar Bharati, Amba P. Kulkarni. 2006. Building a wide coverage sanskrit morphological analyzer: A practical approach.

David Aldous, Illdar Ibragimov, Jean Jacod, and David Aldous. 1985. Exchangeability and related topics. In *École d'Été de Probabilités de Saint-Flour XIII - 1983*, volume 1117 of *Lecture Notes in Mathematics*, pages 1–198. Springer Berlin / Heidelberg.

Kenneth R. Beesley. 1998. Arabic morphology using only finite-state operations. In *Proceedings of the Workshop on Computational Approaches to Semitic Languages*, Semitic '98, pages 50–57. Association for Computational Linguistics.

W.R. Gilks, S. Richardson, and D.J. Spiegelhalter. 1996. *Markov chain Monte Carlo in practice*. Chapman & Hall/CRC.

Sharon Goldwater, Thomas L. Griffiths, and Mark Johnson. 2006a. Contextual dependencies in unsupervised word segmentation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, ACL-44, pages 673–680, Stroudsburg, PA, USA. Association for Computational Linguistics.

Sharon Goldwater, Thomas L. Griffiths, and Mark Johnson. 2006b. Interpolating between types and tokens by estimating power-law generators. In *In Advances in Neural Information Processing Systems 18*, page 18.

Sharon Goldwater. 2006. *Nonparametric Bayesian models of lexical acquisition*. Ph.D. thesis, Brown University.

Oliver Hellwig. 2009. Sanskrittagger: A stochastic lexical and pos tagger for sanskrit. In *Sanskrit Computational Linguistics*, pages 266–277. Springer-Verlag, Berlin, Heidelberg.

Gérard Huet. 2009. Sanskrit computational linguistics. chapter Formal Structure of Sanskrit Text: Requirements Analysis for a Mechanical Sanskrit Processor, pages 162–199. Springer-Verlag, Berlin, Heidelberg.

Malcolm Hyman. 2007. From Paninian Sandhi to Finite State Calculus. In *First International Sanskrit Computational Linguistics Symposium*, Rocquencourt France. INRIA Paris-Rocquencourt.

Brett Kessler. 1994. Sandhi and syllables in classical sanskrit. In *The proceedings of the Twelfth West Coast Conference on Formal Linguistics*, pages 35–50, Stanford, CA, USA. CSLI Publications.

H. Kucera and W. N. Francis. 1967. *Computational analysis of present-day American English*. Brown University Press, Providence, RI.

Anil Kumar, Vipul Mittal, and Amba Kulkarni. 2010. Sanskrit compound processor. In *Sanskrit Computational Linguistics*, volume 6465 of *Lecture Notes in Computer Science*, pages 57–69. Springer Berlin / Heidelberg.

Vipul Mittal. 2010. Automatic sanskrit segmentizer using finite state transducers. In *Proceedings of the ACL 2010 Student Research Workshop*, ACLstudent '10, pages 85–90. Association for Computational Linguistics.

Alan Prince and Paul Smolensky. 1993. Optimality theory: Constraint interaction in generative grammar. Technical report, Rutgers University and University of Colorado.

Peter M. Scharf and Malcolm D. Hyman. 2010. Transcoding. http://sanskrit1.ccv.brown.edu/tomcat/sl/TranscodeText.

Claude E. Shannon. 1948. *A Mathematical Theory of Communication*, volume 27. Bell System Technical Journal, Champaign, IL, USA.

G. K. Zipf. 1932. *Selective Studies and the Principle of Relative Frequency in Language*. Harvard University Press.

A Sonority Hierarchy

Sonority hierarchy is as follows: Vowels $>$ Semivowels $>$ Nasals $>$ Spirants $>$ Voiced Stops $>$ Unvoiced Stops

- Vowels = AaEOeoiIuUfFxX
- Semivowels = yvrl
- Nasals = NYRnmM
- Spirants = hSzsH
- Voiced Stops = gGjJqQdDbB
- Unvoiced Stops = kKcCwWtTpP'

A syllable is said possess a sonority violation if it has more than one sonority peak (e.g. 'rjA'). However, at the beginning of a syllable, we must rate Spirants at the same level as Unvoiced Stops (e.g. 'sTApayati' is valid).