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 Sanskit 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

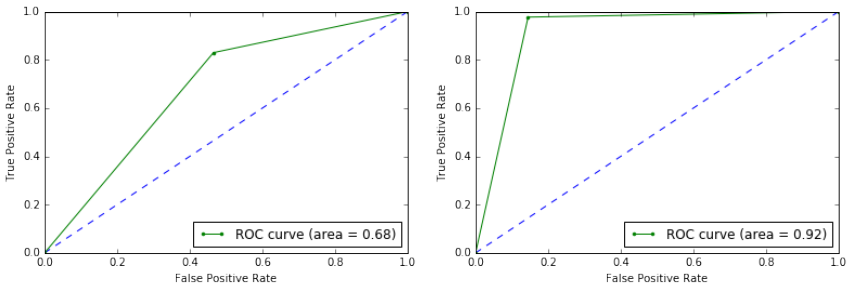
Table 3: Performance Evaluation of various classification algorithms.

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,

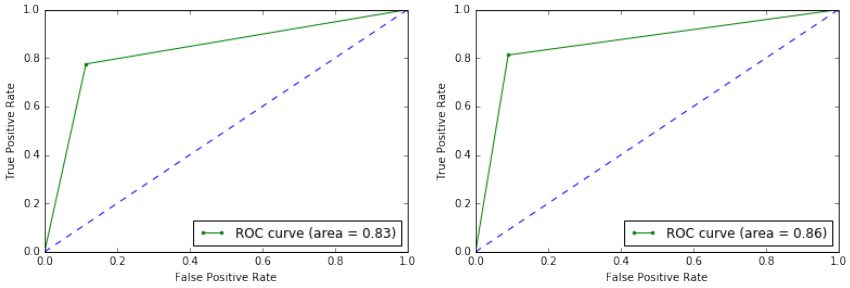
Parameters	Value
Number of neighbors	5
Weights	Uniform
Leaf size	30
Word embedding dimension	500
Context Window size	1
Minimum count	1

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



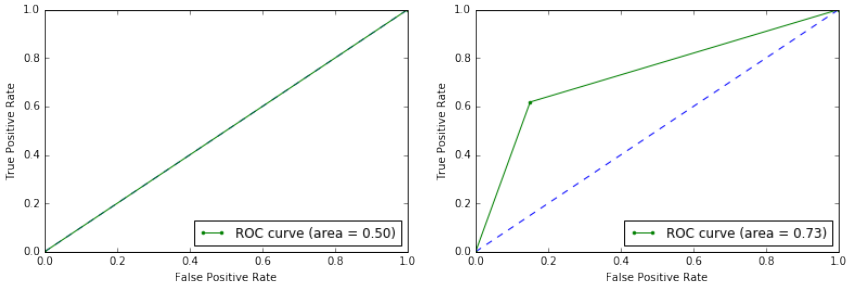
(a) Naïve Bayes

(b) K-Nearest Neighbor



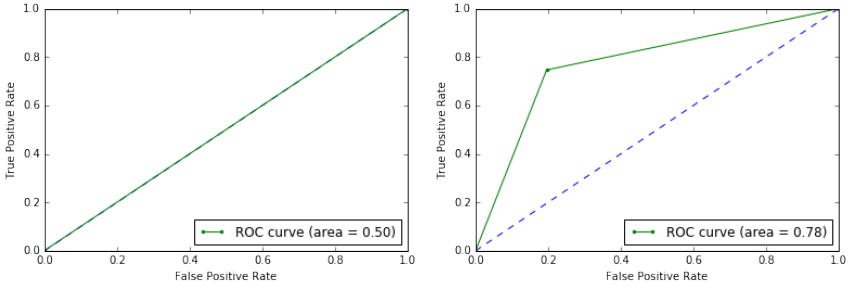
(c) Decision Tree

(d) Random Forest



(e) Support Vector Machine

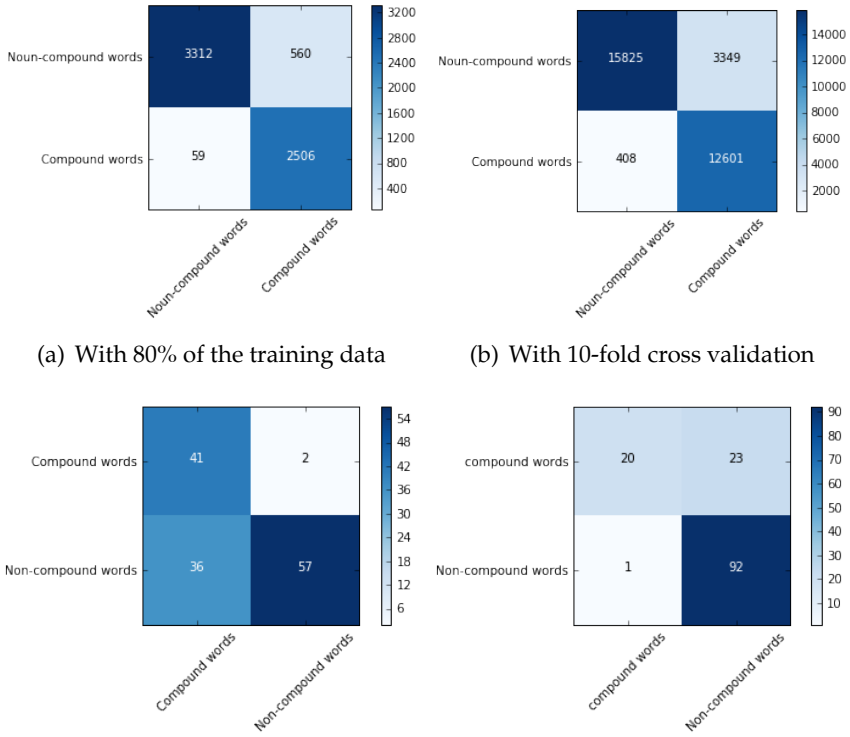
(f) Multi-Layer Perceptron



(g) Logistic Regression

(h) Adaboost

Figure 1: Receiver operating characteristic curves



(a) With 80% of the training data

(b) With 10-fold cross validation

(c) Identification of Samasa from Ashtanga Hridayam text using KNN

(d) Identification of Samasa from Ashtanga Hridayam text using Sanskrit heritage reader [3]

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 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.

Gérard 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.