International Journal of Engineering & Technology, 7 (2.31) (2018) 45-49

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

Kannada morpheme segmentation using machine learning

Sachi Angle[1*], B. Ashwath Rao[2], S.N. Muralikrishna[3]

[1]*Dept. of Computer Science & Engineering, Manipal Institute of Technology, Manipal Academy of Higher Education.*
[2]*Dept. of Computer Science & Engineering, Manipal Institute of Technology, Manipal Academy of Higher Education.*
[3]*Dept. of Computer Science & Engineering, Manipal Institute of Technology, Manipal Academy of Higher Education.*
**Corresponding author E-mail:sachiangle@gmail.com*

Abstract

This paper addresses and targets morpheme segmentation of Kannada words using supervised classification. We have used manually annotated Kannada treebank corpus, which is recently developed by us. Kannada bears resemblance to other Dravidian languages in morphological structure. It is an agglutinative language, hence its words have complex morphological form with each word comprising of a root and an optional set of suffixes. These suffixes carry additional meaning, apart from the root word in a context. This paper discusses the extraction of morphemes of a word by using Support Vector Machines for Classification. Additional features representing the properties of the Kannada words were extracted and the different letters were classified into labels that result in the morphological segmentation of the word. Various methods for evaluation were considered and an accuracy of 85.97% was achieved.

1. Introduction

Kannada, the official language of the state of Karnataka, is spoken by over 40 million people. The language has witnessed the assimilation and recording of continuous literature for over 1000 years. Kannada has rich morphological structure unlike languages like English. A Kannada word in an agglutinative form can be broken into the root and one or more suffixes. The suffixes add to the semantic meaning of the root word. The number and form of suffixes that can be attached to a Kannada word is dependent on the part-of-speech (POS) of the word. Part-of-Speech of the word can be a Noun, a Verb, an Adjective, an Adverb or any other type as detailed in [1]. Standardization of Kannada Part-of-Speech is documented in the Bureau of Indian standards tagset [1]. Nouns can get inflected for gender, case and number. On the other hand, verbs can get inflected for gender, number, tense, aspect and modality. Also, the verbs and corresponding nouns within a phrase agree on gender and number. We illustrate this in Table I using an example.

Table I: Inflections to Nouns and Pronouns Examples of Kannada in WX Notation

Category	Form(in WX)	Inflection
Noun	maneVyalli	Case
Noun	mamawalu	Gender
Noun	maneVyavaru	Number
Pronoun	avanu	Person

It is very challenging to break a Kannada word form into its correct morphemes. Similar form is seen in other Dravidian languages like Telugu, Tamil and Malayalam. This type of morphological structure is also found in other Indo-Aryan languages apart from the Dravidian languages [2][3].
The romanization of Kannada letters in this paper is shown in the WX-format[4]. Kannada verbs get inflected for tense (e.g. kuliwanu which means `sat`), aspect (e.g. kuliwukolluwwA which means `was sitting` - future continuous), and modality (e.g. kuliwukolli which means `sit+polite+plural`). Morphology for Kannada can be defined at an inflectional level and this approach is called inflectional morphology. Inflectional morphology deals with words which comprise of a simple word with inflectional morphemes added to it to produce meaning. On the other hand, Morphology can also be defined at the level of combination of words and this is called derivational morphology. Derivational morphology deals with two or more simple words combined together to produce a new meaning often with an altogether different part-of-speech. Very little work has been done in the domain of Kannada Morphology.
Morphological Analysis is very essential for Machine Translation, Question Answering, Information Retrieval, Information Extraction, Spell checking, Lexicography and other natural language related applications. A Morphological Analyzer splits a word into its root word and all the subsequent morphemes along with their grammatical categories.
In this paper, Section 2 discusses related work. The dataset used for Morphological Analysis is described in Section 3. The Segmentation technique we have applied is discussed in Section 4. Section 5 presents the results of the study. In Section 6, we discuss scope and limitations of the work. Section 7 concludes and summarizes the work.

2. Related work

Machine Learning is extensively used in NLP applications such as speech-to-text conversion, machine translation, information retrieval, question answering, and text proofing and also morpheme segmentation. Morfessor is one such example for unsupervised learning to extract morphs.The model is formulated in a probabilistic maximum a posteriori (MAP) framework. There are a few mathematical frameworks that can be used for formulating models for morphology learning and word segmentation. The techniques include maximum likelihood (ML) modeling, Probabilistic maximum a posteriori (MAP), Minimum Description Length (MDL) principle for word segmentation[5]. Most of the available literature shows that the unsupervised learning is used for morpheme segmentation and is probabilistic in nature as it is well suited for the application. There exists a lot of work on unsupervised morphological segmentation methods. A review of these techniques is given by Hammarström and Borin in [6].

Teemu Ruokolainena et. al. have presented a paper on semi-supervised morhpme segmentation using Conditional Random Fields[7]. Due to the limited availability of annotated dataset, semi-supervised learning technique has been used. Only some part of the data is annotated which is used for the initial training. In the literature we find similar techniques for morpheme segmentation using Expectation Maximization (EM) and Decision Tree based techniques for rule extraction.

The role of the stemmer for the task of information retrieval has been presented by Larkey and Connell [8]. Some of the pioneering work on building a Morphological Analyzer for Kannada was carried out by T N Vikram and Shalini R Urs [9]. A prototype morphological analyzer for Kannada is presented here. The analyzer is based on Finite State machines that handle 500 distinct Noun and Verb stems of Kannada. The morphological analyzer simultaneously served as a stemmer, part of speech tagger and spell checker.

A Morphological Analyzer and Generator for Kannada was also developed by Shambhavi B R et al. [10]. It was developed using a paradigm based approach. Paradigms are referred to in classifying word morphology. Separate Noun and Verb paradigms were developed. A trie data structure was used to store all the words. The root word along with suffixes in Unicode format were put in a trie data structure. A Kannada sandhi splitter was also developed. In the morph analyzer, the word is initially searched for in indeclinable words (avyaya). If found, the word class is returned. If not found, it is searched in a declinable word list (Nouns). If not found then suffixes are stripped till the root word is found. If still not found, then it is searched for in the Conjugable word list. If found after stripping suffixes, the corresponding word category and morph information is returned. Morphological Generator on the other hand, accepts root word, and other grammatical categories of suffix information and generates a word. However, this approach requires and consumes high amounts of memory since a trie data structure is used. A total of 3,700 root words with 88,000 of their inflections were stored in the trie data structure.

A Rule based approach for Morphological Analysis and Generation was tried out by Ramasamy Veerappan et al. [11]. In this, the system was tested on twenty thousand root words comprising of nouns, verbs, adjectives and adverbs. A paradigm based morphological analyzer for verbs was designed and categories of verb paradigms based on Morphological Structure were defined.

A vastly different approach to Kannada Morpheme segmentation was carried out by Suma Bhat [12]. In this, segmentation of Kannada words was done using unsupervised learning. A total of three different methods were employed to assess the efficiency of resulting segmentation. The three methods are Goldsmith's method of unsupervised learning of morphology, Morfessor Categories and High-performance language independent Morphological Segmentation. Goldsmith's method of unsupervised learning of morphology is centered around the idea of minimum description length. The learning heuristic proceeds in steps of discovering basic candidate suffixes of the language using weighted mutual information. This is used to find a set of suffixes, and then Minimum Description Length is used to correct errors generated by heuristics. In Morfessor categories, substrings occurring frequently in several different word forms are proposed as morphs and the words are represented as a concatenation of morphs. The third algorithm is an extension of Keshava and Pitler's algorithm. An accuracy in terms of F-measure of 82.35 for nouns and a F-measure of 59.09 for verbs was obtained.

A Morphological Generator for Kannada based on Finite State Transducers was developed by Bhuvaneshwari C Melinamath and A G Mallikarjunmath [13]. The Generator is developed by accepting root, pattern, and features. The system uses a Kannada Lexicon and various morphophonemic rules. An accuracy of more than 90% for Nouns and around 85% for verbs was achieved.

Morphological Analyzer for Agglutinative Languages Using Machine Learning Approaches was developed by Dhanalakshmi V et al. [14]. Here, the Morphological Analyzer was developed for Tamil. The researchers adopted Machine Learning approaches based on Support Vector Machines. The accuracy obtained in this method is 95.45%. Similar to Kannada, words in Tamil inflect for gender, number, person in case of nouns and verbs, and additionally on tense, aspect, mood, causation, attitude in case of verbs. The authors identified 17 noun paradigms and 32 verb paradigms. Each root word is assigned one of the paradigms. Sequence labeling approach is used in morphological analysis. First, the input words are separated into segments and syllables using Consonant-Vowel representation. Next, segmenting the words into morphemes according to the morpheme boundaries is carried out. In the final step, assigning grammatical classes to each morpheme is achieved. The implementation was carried out by SVMTool. The systems for verbs and nouns were trained with 1,30,000 and 70,000 words respectively.

3. Dataset

Along with other grammatical features, many treebanks have morpheme segmentation of words stored. For many western languages, since morpheme segmentation is a trivial task, it is not annotated. In the Penn Chinese treebank[15], a total of 100K words are annotated for POS and morphological features, and this is stored in the treebank. In the Turkish treebank[16], each word in a sentence is marked for Morphemes. The morphemes of a word are placed in a sequence separated by a '+' character.

We have employed a similar approach in our treebank.

We have used Kannada treebank in our work. The Kannada treebank was recently developed at Manipal Institute of Technology, Manipal in collaboration with International Institute of Information Technology, Hyderabad, funded by the Department of Electronics and Information Technology, New Delhi. A total of 198K words from general domain, 46K words in parallel corpus and 10K words from conversational text was annotated for Morph features, POS features and dependency features. The developed treebank has been checked for qualitative measures such as sanity check, inter-annotation agreement and manual re-verification. The dependency features employ Paninian framework. Various error checking tools have been employed to enhance the quality of the annotation. The general domain corpus has been taken from the Indian Language Machine Translation (ILMT) corpus. Each word of the sentences in the corpus is annotated in ingeniously developed Shakti Standard Format (SSF). Sanchay, an annotation tool specially developed for Indian languages is used. A sample sentence from the treebank is shown below.

```
 <Sentence id='1'>
1 (( NP <fs af=',,,,,,,' name='NP' drel='k1:VGNN'>
1.1		ದಾಸ್ತಾನು	N__NN	<fs
af='ದಾಸ್ತಾನು,n,,sg,3,d,0,0'name=`ದಾಸ್ತಾನು'>
1.2		ವಿಭಾಗವು	N__NN	<fs af='ವಿಭಾಗ,n,,sg,3,o,ಅವ್+ಉ,av+u'
name='{ವಿಭಾಗವು}'>
))
2 (( NP	<fs name='NP2' drel='k7p:VGNN'>
2.1		ತಯಾರಿಕಾ	N__NN	<fs af='ತಯಾರಿಕಾ,n,,sg,3,d,0,0'
ame='{ತಯಾರಿಕಾ}'>
2.2		ವಿಭಾಗಕ್ಕೆ	N__NN	<fs af='ವಿಭಾಗ,n,,,sg,3,o,ಕ್ಕೆ,kkeV' name='
ವಿಭಾಗಕ್ಕೆ'>
2.3		ಹತ್ತಿರದಲ್ಲಿ	N__NST <fs af =
'ಹತ್ತಿರ,nst,,,,o,ಅದ್+ಅ+ಅಲ್ಲಿ,ax+a+alli' name='ಹತ್ತಿರದಲ್ಲಿ'>
 ))
3 (( VGNN		<fs af=',,,,,,,' name='VGNN'
drel='k1:NULL__VGF'>
3.1		ಇರುವುದು V__VM__VNG		<fs
af='ಇರು,v,,sg,3,,ಉವ್+ಉ+ಉದ್+ ಉ,uv+u+ux+u' name='ಇರುವುದು'>
 ))
```

```
4 (( NP     <fs af=’,,,,,,,’ name=’NP3’ drel=’adv:NULL__VGF’>
4.1         ಎಲ್ಲ         JJ          <fs af=’ಎಲ್ಲ,adj,,pl,3,d,0,0’ name=’ಎಲ್ಲ'>
4.2         ದೃಷ್ಟಿಯಿಂದಲೂ          N__NN
<fs af=`ದೃಷ್ಟಿ,n,,sg,3,o,ಇಯ್+ಇ+ಇಂದ+ಅಲ್+ ಉ,iy+i+iMxa+al+U'
name=`ದೃಷ್ಟಿಯಿಂದಲೂ'>
))
5 ((NP      <fs af=’,,,,,,,’ name=’NP4’ drel=’k1s:NULL__VGF’>
5.1         ಉತ್ತಮ      N__NN     <fs af=’ಉತ್ತಮ,n,,sg,3,d,0,0’
name=’ಉತ್ತಮ'>
))
6 (( NULL__VGF      <fs af=’,,,,,,,’ name=’NULL__VGF’
stype=’declarative’ voicetype=’active’>
6.1         NULL       V__VM__VF          <fs af=’,,,,,,,’ name=’NULL’
troot=’ಆಗಿದೆ’ mtype=’gap’>
))
7           ((          BLK <fs af=’,,,,,,,’ name=’BLK’
drel=’rsym_eos:NULL__VGF'>
7.1         .           RD__PUNC           <fs af=’.,punc,,,,,,’ name=’.’>
))
</Sentence>
```

The above sentence has seven chunks with words under each chunk annotated for Part-of-Speech, and Morphological features. All the morphological features are captured in an abbreviated feature set. The abbreviated feature set includes root word, lexical category, gender, number, person, case marker, tenses, aspect and modality in that order. The semantic dependency information between the chunks is also captured. Each chunk in its header has drel attribute which contains dependency relation. The first part of drel relation has nature of dependency relation (tag) while the second has the name of the other phrase with which this phrase has a dependency relation [17].

4. Segmentation technique

The Support Vector Machines (SVMs) model has been used for classification. Support Vector Machines is a machine learning model used for classification or regression. For classification, the model plots each item of the training data set on an n-dimensional data space, where n is the number of features present. It aims to find the best hyperplane that can accurately classify the data items into their classes. SVMs use kernels to convert the input data space into a higher dimensional data space. This converts a non separable problem to a separable one and enables better classification of the data.

In our model we have used the Support Vector Machine Classification (SVC) implementation of the Scikit-Learn library. The model aims to predict the segmented form of an input word by taking one letter of the input at a time and predicting if it remains the same in the output, or if it is modified to another letter (or group of letters) to result in the morphologically segmented form of the output. We use an ‘*’ to depict the boundary of a morpheme or the root. For example, the word viBAgakkeV consists of the root viBAga and the suffix kkeV. This will be written as viBAga*kkeV*. Thus the model has to process each input letter and predict where the ‘*’ should be predicted to depict the morpheme boundaries.

The Kannada words were first romanized using a mapping function which maps the characters to their WX format [4]. The WX notation is used to represent Indian languages’ characters using the ASCII notation, thus making the data easier to handle and understand for non-native Kannada speakers. Each letter of each word is used as an input to predict the morphological structure of that word. This can only work if the length of the input word is always equal to the length of its morphologically segmented form. However, the length of the segmented output isn’t always the same as the original Kannada word, and so the output segmented forms have to initially be preprocessed such that a parallel alignment can be achieved between the input Kannada words and the morphologically segmented outputs.. There are two cases when this can happen:

Case A: When the number of characters in the morphologically segmented form is greater than that of the original word, letters of the segmented word may have to be grouped together such that each group of letters of the output matches one letter of the input. For example, sAmAnyavAgi, when segmented, becomes sAmAnya*av*A*Agu*i, where the ‘*’ indicates the boundary of the root, or a suffix. As, for this example, the segmented form has 4 letters more than the original word, it has to be partitioned such that every letter in the original word corresponds to a group of letters in the segmented form. The original unaligned output, and the resultant aligned output can be seen in Table II.

Table II: Illustration for Case A

Input	s	A	m	A	n	y	a	v	A	g	i	-	-	-
Unaligned Output	s	A	m	A	n	y	a*	a	v*	A*	A	g	u*	i
Aligned Output	s	A	m	A	n	y	a*	av*	A*	Agu*	i	-	-	-

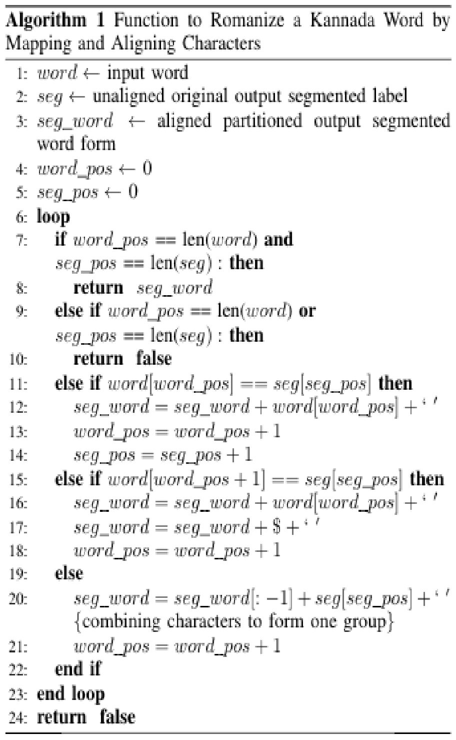
Case B: On the other hand, in some words, the length of the segmented form of the word is lesser than the word. For example, ovlYlYeVya when segmented, becomes ovlYlYeV*a. Here, the length of the segmented form is lesser than that of the original by one letter. In this case, to be able to match each letter of the original word with a letter of the segmented form, an extra ‘null’ character has to be added to the segmented word. We use ‘$’ for this purpose. This example is illustrated in Table III.

Table III: Illustration for Case B

Input	o	v	l	Y	l	Y	e	V	y	a
Unaligned Output	o	v	l	Y	l	Y	e	V*	a	-
Aligned Output	o	v	l	Y	l	Y	e	V*	$	a

In order to accomplish this accurate mapping and alignment of characters, a greedy algorithm is used. This algorithm parses one character of the input word at a time, comparing it with the character of the segmented word (Step 3 of the Algorithm), until the end of either the word, or the segmented form, or both is reached. If both, the word and the segmented output, have been parsed (Step 1 of the Algorithm), this implies that the word and its corresponding segmented form have an equal number of input letters and output segments respectively. This input-label pair can be added to the training and testing dataset. While the word is being parsed, if a mismatch occurs, it could either be because of an extra character (Case A), or a character less (Case B) in the segmented form of the word. First, the next character of the word is matched against the current character of the segmented word (Step 4 of the Algorithm). If a match occurs, a $ is added to the output indicating a character is less in the segmented form of the word (Case B). Else, if this is also a mismatch, the next character of the segmented word is matched against the current character of the input word (Step 5 of the Algorithm). If a match occurs, this letter is clubbed with the previous to form a partition, indicating an extra character in the segmented form of the word(Case A). If a match is still not found, the letter is clubbed with the previous letter anyway, for cases where the segmented form of the word may have a continuous number of extra letters (for eg. i => ixu*). If, at the end, only either the word or the segmented output have been completely parsed while the other hasn’t, then the word could not be accurately aligned with its segmented form, and thus cannot be added to the dataset. This method successfully partitioned 98.13% of the data.

The Algorithm of the function is stated below wherein *word* is the input word and *seg* is the original output segmented label. *Word_pos* and *seg_pos* parse through the word and the segmented form respectively and are initialised to index position 0. *Seg_word* is the partitioned output segmented form of the word that is accurately aligned with the input word and is used in the training and testing datasets. The partitions within the word and the segmented form are denoted using spaces, where each letter in the input word is aligned with a ‘segment’ in the output as shown in the examples above. Each time *Seg_word* is successfully aligned with the word, it is added to an output list (*List*).



Algorithm 1 Function to Romanize a Kannada Word by Mapping and Aligning Characters

```
1: $word \leftarrow$ input word
2: $seg \leftarrow$ unaligned original output segmented label
3: $seg\_word \leftarrow$ aligned partitioned output segmented
   word form
4: $word\_pos \leftarrow 0$
5: $seg\_pos \leftarrow 0$
6: loop
7:    if $word\_pos == len(word)$ and
      $seg\_pos == len(seg)$ : then
8:       return $seg\_word$
9:    else if $word\_pos == len(word)$ or
      $seg\_pos == len(seg)$ : then
10:      return false
11:   else if $word[word\_pos] == seg[seg\_pos]$ then
12:      $seg\_word = seg\_word + word[word\_pos] + ` '$
13:      $word\_pos = word\_pos + 1$
14:      $seg\_pos = seg\_pos + 1$
15:   else if $word[word\_pos + 1] == seg[seg\_pos]$ then
16:      $seg\_word = seg\_word + word[word\_pos] + ` '$
17:      $seg\_word = seg\_word + \$ + ` '$
18:      $word\_pos = word\_pos + 1$
19:   else
20:      $seg\_word = seg\_word[: -1] + seg[seg\_pos] + ` '$
         {combining characters to form one group}
21:      $word\_pos = word\_pos + 1$
22:   end if
23: end loop
24: return false
```

Fig. 1: Greedy Algorithm for Segmentation

On completion of this, every letter of every input word is aligned with a corresponding group of letters in the segmented form. Thus, if you take each letter of the input, you can predict the form (label) they would take in the segmented output. For example, considering the word ‘o v l Y l Y e V y a’ and its morphologically segmented output ‘o v l Y l Y e V* $ a’, an input of ‘o’ should predict ‘o’, of ‘v’ should predict ‘v’, and so on, until ‘V’ should predict ‘V*’ and ‘y’ should predict a ‘$’.

Every character of the input, and every label of the output is given a numerical identifier. Along with the input character, other features used include the prefix of the word that has already been parsed. That is, the letters of the word that have already been processed. This prefix of the current letter is given a numerical value based on the letters in the prefix and their positions. Whether the current character is a vowel, a consonant, a number, or a punctuation symbol, is another feature. The prefix of the current segment, that is, all the letters encountered after the last identified morpheme boundary, and whether the current letter is still in the root of the word or not, are also used as additional features. The Kannada treebank provides information of the part of speech the word belongs to, whether it is a singular word or plural, and direct or oblique. Using this data as features for each letter, the input features and output labels for the Support Vector Machine were gathered.

The Features used and the placeholders used to refer to them through the rest of the paper are listed as follows:

1. Input letter from the word: current letter
2. The letters on the word that come before current letter and which have already been parsed: prefix
3. Whether the current letter is a vowel, a consonant, a number, or a punctuation symbol: letter category
4. Whether the current letter is still a part of the root or if it is a part of the suffix: is root
5. The letters encountered after the last predicted morpheme boundary: current prefix
6. Part of speech: pos
7. Singular or Plural: is singular
8. Direct or Oblique: is direct

5. Results and discussion

When trained on a data set with the features - current letter, prefix, letter category, is root, and current prefix - the model achieved a good accuracy percentage of 82%. This was calculated using the accuracy which calculated the percentage of labels that were correctly predicted.

Accuracy = (Number of output labels predicted correctly) / (Total number of output labels)

To improve on this, additional features - is noun, is direct, and is singular - were included. Using the score function again, the percentage of correctly predicted labels was calculated to be 85.59%. Thus it increased by almost 4%.

Table IV: Accuracy Percentages Achieved with the Different Evaluation Methods and Sets of Features

Features	**Evaluation Method**	**Accuracy %**
current letter, prefix, letter category, is root, and current prefix	Accuracy	82.73
current letter, prefix, letter category, is root, current prefix, pos, is singular and is direct	Accuracy	85.59
current letter, prefix, letter category, is root, current prefix, pos, is singular and is direct	Custom Accuracy Function	84.57

The output labels consist of one or more characters (For eg. ‘Agu*’). Therefore, if an input is inaccurately classified to a class that is partially similar to the correct class (For example, the letter ‘A’ classified as ‘Agg’ instead of ‘Agu*’), the model shouldn’t be penalized to the same extent as it is when classifying an input to a completely different class (For example, the letter ’A’ classified as ‘V*’). Even if the model predicted a fraction of the output correctly, the prediction is deemed as entirely wrong by the Accuracy function. For this reason, another evaluation function, henceforth referred to as the Custom Accuracy Function, was also devised, which made use of the Levenshtein distance similarity measure, to calculate the percentage of each label that was correctly predicted. Levenshtein distance is the edit distance between two strings, given that the cost of deletion, addition and substitution are equal. It returns the number of characters that differ between the two strings. As the predicted output label could have a greater number, lesser number or equal number of characters as the the actual output label, the number of correctly matched characters can be obtained using the following equations:

Lev_Distance = Levenshtein_distance(prediction, word)
Greater = max(strlen(prediction), strlen(word))
Correctly Matched Characters = Greater - Lev_Distance

Here, Greater is the length of the longer string between the prediction string and the actual output string. The total number of correctly predicted letters is obtained by summing up the above equation for Correctly Matched Characters over all the words in the test data set. The total number of predicted characters is equal to the sum of Greater over all the words.

$$\text{Accuracy} = \frac{\text{sum (Correctly Matched Characters)}}{\text{sum (Greater)}}$$

The accuracy percentage is then the ratio of correctly predicted characters to the total number of predicted characters. Using this method, the accuracy percentage achieved was 84.57%.

The dataset consisted of 89622 words. Each word was split into its constituent letters, each of which was used as an input feature to the SVM model. The above results were obtained on splitting the data into the training set and testing set, where the training set comprised of 75% of the data. In an attempt to improve the accuracies obtained, observations were made by splitting the data into different proportions.

Table V: Difference in Accuracy Achieved Using Different Fractions of the Data as Training Data

Evaluation Methods: Percentage of data used for training:	Score Function	Custom Accuracy Function
65%	84.72%	83.69%
75%	85.59%	84.57%
85%	85.97%	84.83%

Therefore, exposing the machine learning model to 10% more data while training, improved the accuracy to 85.97%.

6. Limitations and scope

The classifier works with features like the properties of the word, which may not be available for every application. Without them, it achieved an accuracy of 82.73%.
Morphological segmentation of words can play a huge role in machine translation. Complete knowledge of the word allows for accurate translations. Segmenting the root and the suffixes also allows for decreasing the size of the data set by removing redundant words. When the application only needs to deal with the root word, treating two words with the same root but different suffixes as different words isn't efficient.

7. Conclusion

Word forms in Kannada, like other Dravidian languages, can have a lot of suffixes attached to its roots often giving words a complex structure. This complex structure is parsed and predicted using the SVM classifier after romanizing the words and preprocessing them. Apart from the first set of features used, the addition of features about the word - whether it's a noun or a verb, if it is singular or plural, etc - caused an increase in the accuracy of the model. After training on the Kannada treebank corpora, we have achieved an accuracy of 85.97% using the Score function, and an accuracy of 84.83% using the evaluation function based on Levenshtein Distance. Our study and results establish that higher and improved accuracies can be obtained using features that deal with the properties of the words, and by using even larger datasets. Indian languages in general have rich morphological form, therefore, the pipeline and processes discussed would serve useful in other NLP applications such as Machine Translation and Information Extraction.

Acknowledgement

We are thankful to Dr. Dipti Mishra Sharma, Professor and Head, Language Technologies and Research Centre (LTRC), International Institute of Information Technology, Hyderabad and her colleagues and students for their guidance and help in creating the Kannada treebank corpus.

References

[1] http://tdil-dc.in/tdildcMain/articles/134692Draft%20POS%20Tag%20standard.pdf

[2] Vikram S, "Morphology: Indian Languages and European Languages", *International Journal of Scientific and Research Publications*, Vol.3, No.6, (2013).

[3] Goyal V & Lehal GS, "Hindi morphological analyzer and generator", *First International Conference on. Emerging Trends in Engineering and Technology*, (2008).

[4] Gupta R, Goyal P & Diwakar S, "Transliteration among Indian Languages using WX Notation", *KONVENS*, (2010).

[5] Creutz M & Lagus K, "Unsupervised models for morpheme segmentation and morphology learning. ACM Trans", *Speech Lang. Process.*, Vol.4, No.1, (2007).

[6] Hammarström H & Borin L, "Unsupervised learning of morphology", *Comput. Linguist.*, Vol.37, No.2, (2011), pp.309-350.

[7] Ruokolainen T, Kohonen O, Virpioja S & Kurimo M, "Supervised morphological segmentation in a low-resource learning setting using conditional random fields", *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, (2013), pp.29-37.

[8] Larkey LS & Connell ME, "Structured queries, language modeling, and relevance modeling in cross-language information retrieval", *Information processing & management*, Vol.41, No.3,(2005), pp.457–473.

[9] Vikram TN & Shalini R Urs, "Development of prototype morphological analyzer for the south indian language of kannada",. *Asian Digital Libraries. Looking Back 10 Years and Forging New Frontiers*, (2007), pp.109–116.

[10] Shambhavi BR., Ramakanth Kumar P, Srividya K, Jyothi BJ, Spoorti Kundargi & Varsha Shastri G, "Kannada morphological analyser and generator using trie", *IJCSNS*, Vol.11, (2011).

[11] Veerappan R, Antony PJ, Saravanan S & Soman KP, "A rule based kannada morphological analyzer and generator using finite state transducer", *International Journal of Computer Applications*, Vol.27, No.10,(2011), pp.45–52.

[12] Bhat S, "Morpheme segmentation for kannada standing on the shoulder of giants", *24th International Conference on Computational Linguistics*, (2012).

[13] Melinamath BC & Mallikarjunmath AG, "A morphological generator for kannada based on finite state transducers", *Electronics Computer Technology (ICECT)*, Vol.1, (2011), pp.312–316.

[14] Dhanalakshmi V, Rekha RU, Kumar A, Soman KP & Rajendran S, "Morphological analyzer for agglutinative languages using machine learning approaches", *International Conference on Advances in Recent Technologies in Communication and Computing*, (2009), pp.433-435.

[15] Xia F, "The segmentation guidelines for the Penn Chinese Treebank (3.0)", *Technical Report*, (2000).

[16] Cakıcı R, "Morpheme segmentation in the METU-Sabancı Turkish Treebank", *Proceedings of the Sixth Linguistic Annotation Workshop. Association for Computational Linguistics*, (2012).

[17] Rao A, Muralikrishna SN & Nayak A, "Developing A Dependency Treebank for Kannada", *An International Journal of Engineering Sciences, Special Issue iDravadian*, (2014).

[18] Bharati A, Sangal R & Sharma DM, "SSF: Shakti standard format guide", *Language Technologies Research Centre, International Institute of Information Technology, Hyderabad, India*, (2007), pp.1-25.