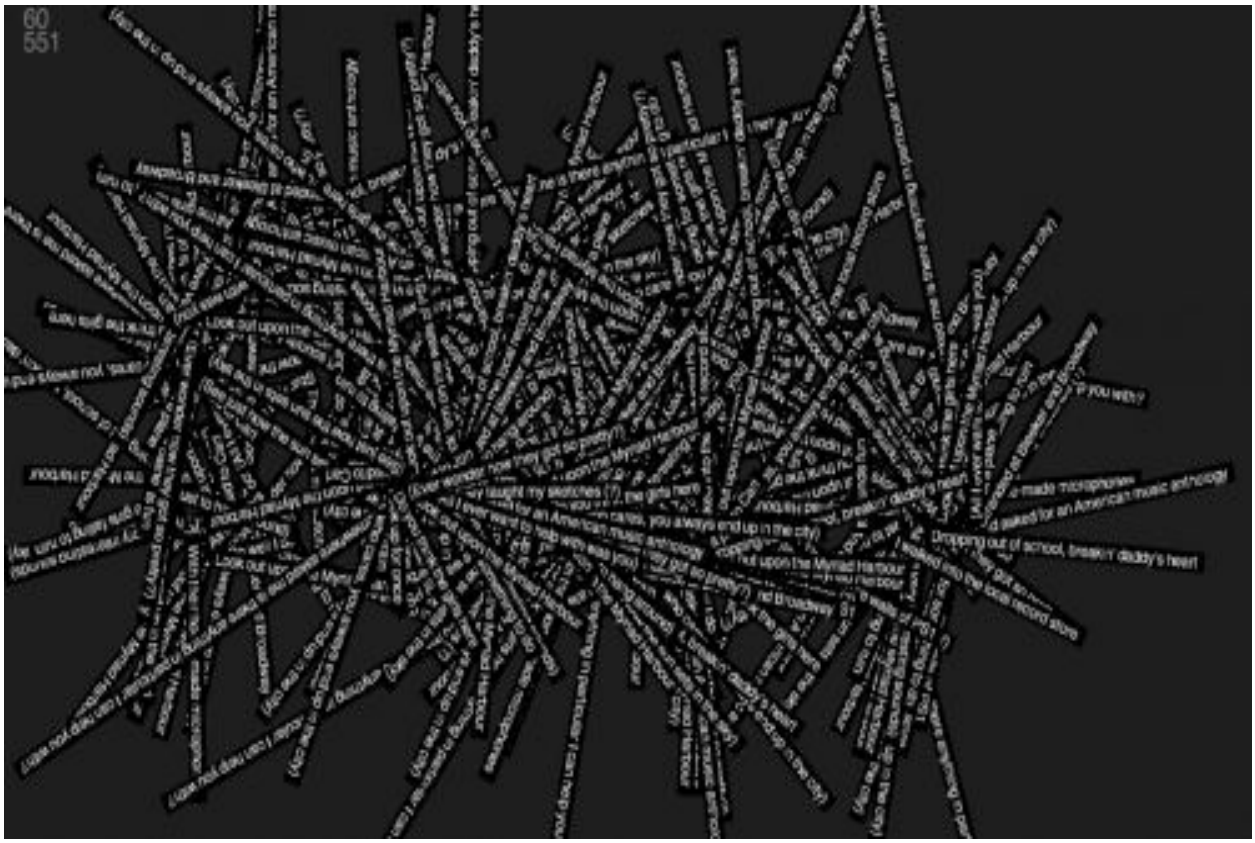
Information Retrieval and Extraction

Text processing framework for Hindi

Submitted by Team 7 Date - 13/04/16

Itisha Dewan (201301052), Praveen Yadav (201505517), Vaibhav Kumar(201325115)



“A DIFFERENT LANGUAGE IS A DIFFERENT VISION OF LIFE”

- FEDERICO FELLINI

Table Of Content

1 Introduction

The aim was to construct a text processing framework for Hini. The text processing framework comprised of the following subtasks:

- **Stop Word Detection**
- **Tokenisation**
- **Sentence Breaking**
- **Identifying Variations**
- **POS Tagging**
- **Concept / Keywords Identification**
- **Entity Recognition**
- **Categorisation**

Each task had its own challenges which will be mentioned in the further sections. Python was primarily used for implementation purposes. Python libraries like nltk were used. Also, for training the CRF models, CRF++ was used.

2 Timeline

The entire timeline was divided into three phases. The first phase comprised of creating a scope document for the entire project. In the scope document we had identified some challenges, but later during the course of the project we identified some more. In the second phase, most of the tasks were complete except for that of Identifying Variations and Entity Recognition. In the third phases, the leftover tasks were completed and then the already completed ones were modified to check whether the performance upgrades or not. The entire project was successfully completed and the code for it can be found on GitHub. A github page corresponding to the project was also hosted, and a report slide and video was also made.

3 Analysis of the tasks

Below is the procedure adopted in accomplishing the various tasks that were mentioned. The challenges faced and further what could be incorporated has also been mentioned.

3.1 Tokenization

Tokenisation is the process of breaking the stream of text into words, phrases or any other important unit for that matter. The task here was to identify these tokens for the Hindi Language.

Given, that we have a lot of sentences in Hindi, we would want the entire text to be segregated into tokens i.e words. Along with this we would also want that other tokens like dates, designations, abbreviations do retain their original form and don't get messed up.

Compared to English, somehow the task of tokenizing Hindi becomes simpler. This is because of in English we have words like “isn’t”, which has an apostrophe. In Hindi, we do not find such words. Hence, for the case of English a decision needs to be made whether we want to consider the word with the apostrophe or without it. Such decisions aren’t required in the case of Hindi.

The nltk tokenizer was used to tokenize the entire Hindi text. The task become fairly simple as the tokenizer handles a variety of cases.

3.2 Sentence Breaking

In Hindi we denote the end of a sentence by what is called a 'poorna viraam' which is denoted by '।'. Our goal here was to identify that and taking care of all the scenarios, break the entire text into sentences.

The ‘purna viram’ has a particular unicode encoding. The unicode is u'\u0964'.

The method that we had adopted , simply used the tokens obtained from the tokenizer to find out where this character occurred and simply consider the tokens between this

and the previous ‘poorna viraam’ character to form the sentence. This was found to work fairly well. The problems that could arise were that there might be no spaces between the ‘poorna viraam’ and the ending or the starting word after that, but this was already being handled by the tokenizer.

The result that we had looked something like this :

```
<Sentence id="10">
1	नैनीताल
2	डिस्ट्रिक
3	में
4	रामनगर
5	और
6	हल्द्वानी
7	के
8	आलवा
9	अन्य
10	ब्लॉक
11	में
12	ये
13	स्कीम
14	काम
15	कर
16	रही
17	है
18	।
</Sentence>
```

The sentences were assigned a sentence id and the tokens were incorporated within it.

3.3 Stopword Detection

These are the most common words that can be found in a particular sentence. Generally, these are the words that are very helpful in bringing grammaticality to a particular sentence.

We adopted two methods for this purpose. In the first method a simple frequency count of the entire words in the corpus was made. The words with the highest frequency in the corpus was considered to be the list of stopwords. The number of stopwords that was desired could be supplied in one of the arguments.

In the second method, we tried to use to the tf-idf scheme for this purpose. Since, we had a corpus which had a set of documents, each with text corresponding to a particular

article, we used it find out the set of stop words. Each word would then be marked by their importance in their documents as well.

In this case however, both performed in the same way. A list of the top 10 stopwords provided by both these methods is as follows:

के
में
है
की
से
को
का
ने
हैं
भी

One the ways in which we could have probably improved this would be if the context in which such words could be taken be accounted for. The context would help identify the importance of these words. If a particular word occurs in each context, then for sure it would have been part of the stop list. However, the frequency count to some extent incorporates this fact. The count a particular word would only increase if it is occurring in a variety of contexts.

3.4 Part of Speech Tagging

The task here was to tag a particular using a predefined set of POS tags. It is a method by which the word categories are disambiguated.

For example : In the following sentence,

"The boy is running"

Word	Part of Speech
The	Determiner
boy	Noun
is	Auxiliary verb
running	Verb

We would want something similar to be done with Hindi as well.

For this purpose we used two different methods. Below is the methodology followed and along with it are suggestions, that could have given us better results.

First we choose to implement the HMM (Hidden Markov Model). The language model that we used here was a trigram model. For the decoding sequence we implemented the Viterbi algorithm. The markov assumption here was that the tag of a word would depend on the previous two words in a particular sentence. But in learning the language model there was a bit of a problem. The problem is with the counts that is provided by the language model. Many a times, we faced a situation that, when we had to tag a particular sentence, a particular trigram was never seen before. Due to this, the count had become 0, and because of that, the tagging was drastically poor.

To overcome this situation, we had to take care of counts. Zipf's law here seemed to be useful. So, according to it, the unseen trigrams would have some probability of occurring, which would be similar. The answer to get this probability of occurrence was smoothing.

We used Laplace's smoothing for this purpose. What smoothing essentially does is that it shaves off some probability mass from that what we have seen, and provides it to that what we haven't. Smoothing helped in improving this task.

Next, the model that we wanted to implement was the CRF based model i.e the conditional random field model. We used a library for this purpose and hence the only thing that we needed to do was preprocess the entire data so that it could be fed to the learner.

The CRF based model turned out to be working better than the HMM based model which we had programmed from scratch.

Accuracies that were seen were :

CRF	~91%

HMM	~81%

The reason for this could be the fact that, the CRF model requires some features to be fed to it. These features are then converted to feature functions which was help in finding out the best sequence of states. The features are a very intrinsic property of the CRF. In the case of the HMM we had to rely on a language model. Hence, if the language model wasn't trained on a proper set then out tagger's performance would significantly reduce. But apart from the language model there is no other information provided to it. Hence, some important features that we could have provided are left behind. CRF overcomes this by selecting a set of features to deploy.

Also, we could have improved the performance of HMM based tagger, had we used some different kind of smoothing for the language model. We could have used a Kneser-Ney smoothing, which is supposed to be one of the best performing smoothing methods.

Example of our output:

sUraja	NNP
kI	PSP
kiraNeM	NN
aba	PRP
pahAdoM	NN
para	CC
KAlI	JJ
hAWa	NN
nahIM	NEG
AegI	VM

3.5 Concept / Keyword Identification

This task could be renamed as chunking. We essentially want to chunk the entire sentences. The only difference between this task and that of POS tagging is that, we want to the boundary level details as well. Hence, in this case structural tags are used instead of the POS tags. The structural tag is a triple consisting of POS tags, structural relation and chunk tag.

The next step is very simple, we learn it in the same way as we had done in case of POS tags. In this case also, we would be required to smooth the data.

Example :

```
((          NP
sUraja      NNP
kI          PSP
))
((          NP
kiraNeM     NN
))
((          NP
aba         PRP
))
((          NP
pahAdoM             NN
))
((          CCP
para        CC
))
```

3.6 Entity Recognition

The training data we work on is tab separated token then its POS tags and then the class of Entity it belongs to . For example here's one token and its tab separated features :
मैसूर NNP NP_B ENAMEX_LOCATION_B

The following different entities classes were there - ENAMEX_PERSON_B , ENAMEX_LOCATION_B(लाल बाग) , ENAMEX_PLANTS_B(चंदन) , TIMEX_PERIOD_B(१८वीं शताब्दी) , NUMEX_COUNT_B(२००) , TIMEX_YEAR_B (१८६४)

Issues we faced in Indian Languages :

1. Lack of Capitalization: There is no such feature in hindi so it is difficult to generate rules or features.
2. Hindi is inflectional and morphologically rich and are free word order.
3. Lack in Resources : Web mostly have list of Named entities in English only.

4. In dictionary of Hindi , many common nouns also exists as proper nouns. E.g. Lata, Suraj, Akash etc are names of persons as well as common noun. Hence we need to resolve the ambiguity.

We had various approaches for NER from there -

- HIDDEN MARKOV MODEL - Hidden Markov Model is a statistical model in which the system being modeled is assumed to be a Markov process with unobserved state. Each state has a probability distribution over the possible output tokens. Therefore the sequence of tokens generated by an HMM gives some information about the sequence of states.
- CONDITIONAL RANDOM FIELD - Conditional Random Field is a probabilistic framework for labeling and segmenting structured data, such as sequences, trees and lattices. The underlying idea is that of defining a conditional probability distribution over label sequences given a particular observation sequence, rather than a joint distribution over both label and observation sequences.
- MAXIMUM ENTROPY BASED MODEL - We have used MaxEnt model to build the baseline NER system. MaxEnt is a flexible statistical model which assigns an outcome for each token based on its history and features. Given a set of features and a training corpus, the MaxEnt estimation process produces a model. For our development we have used a Java based opennlp MaxEnt toolkit to get the probability values of a word belonging to each class. That is, given a sequence of words, the probability of each class is obtained for each word. To find the most probable tag corresponding to each word of a sequence, we can choose the tag having the highest class conditional probability value. But this method is not good as it might result in an in-admissible assignment. Some tag sequences should never happen. To eliminate these inadmissible sequences we have made some restrictions. Then we used a beam search algorithm with a beam of length 3 with these restrictions.

Why we chose CRF over HMM - The primary advantage of CRFs over hidden Markov models is their conditional nature, resulting in the relaxation of the independence assumptions required by HMMs in order to ensure tractable inference .

To increase the efficiency we apply several rules internally -

Rules like : Word end with **नगर** location . Another example, 'pura', 'bAda', 'nagara' etc. are location suffixes. We used binary features corresponding to the lists - whether a given word has a suffix from a particular list.

Steps :

1.java

```
((      NP
        <ENAMEX TYPE="PERSON">
        मुगल    JJ
        राजवंश  NN
        </ENAMEX>
        के      PSP
))
```

Is changed to -

```
मुगल    JJ      NP_B    ENAMEX_PERSON_B
राजवंश  NN      NP_I    ENAMEX_PERSON_I
के      PSP     NP_I    O
```

2. it was broken down to 5 folds, where 4 folds would be used as training and the left one would be used for testing.

3. We used CRF++ for training and testing which is a library implementing CRF algorithm.

To Train a CRF using CRF++, you need 2 thing:

A Template file : (where you define features to be considered for training)

A Training data file : (where you have data in CoNLL format)

4. The command to train is : $ crf_learn -t crf_template hindismalltrainset.txt model.txt

This command will output binary of model. In this text file you can see feature weights

as well. It takes template file as input and training file .

5. The command to test is : $ crf_test -m model.txt.txt hindismalltestset.txt > output

6. Calculated accuracy - **91%**

3.7 Categorisation

The task here was to categorise the data in different news articles into a variety of topics that was provided in the data.

The categories listed were as follows:

fashion
business
entertainment
travel
sports
automobiles
foods
criminals
health
politics
technology

The vocabulary size was found to be close to **372598**

The options that we had thought of initially was to implement the naive bayes but further reading prompted us to use a different approach. We finalised on using KNN as our classifier.

For this purpose, we had to first preprocess the entire set of text documents. We then divided out entire data into 5 fold. Hence, we decided to go for a five fold cross validation.

But before that, we had to randomise the entire data. After randomising the data, it was broken down to 5 folds, where 4 folds would be used as training and the left one would be used for testing. We then used five different values for K, which were 3, 4, 5, 6, and 7.

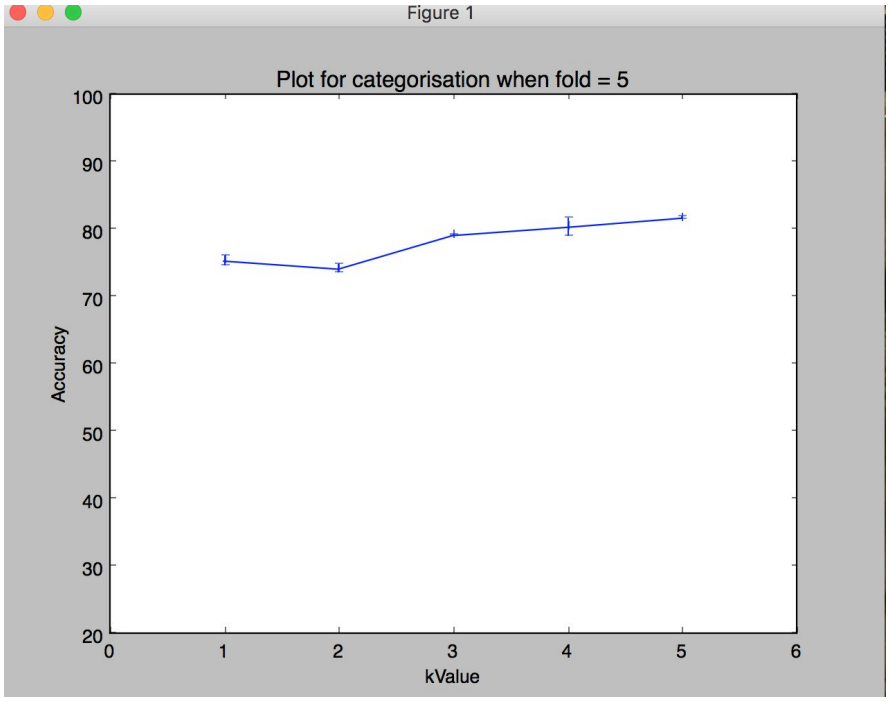
So, first in the training phase, the entire documents were respectively converted to their vectors. When a new test document came in, the nearest neighbours were checked for, and the category which belonged to most of the neighbour, was selected. Initially we had broken the ties randomly during the second phase. We later decided to have a method in which we could fall back to lower K's until and unless we could break the tie.

To convert the documents into vectors, the tf-idf scheme was used. The implementation was entirely written by us with no other libraries used.

Below is the accuracy mentioned:

K	Accuracy
3	~75%
4	~74%
5	~79%
6	~80%
7	~81%

Here is a graph with mean and standard deviation:



One of the things that we could have done, would be to incorporate the fact of variations within the language (which is discussed below). It could be that a particular word is represented(spelled) differently, because of the varying nature of language. We would want to incorporate that as well.

The other thing that we could have done is to rely on some dimensionality reduction techniques like PCA (Principal Component Analysis), or perhaps even SVD could be used to reduce the size of the vocabulary and then see if the accuracy improves or not. The other thing that we could have tried would to be find out different features and try some clustering algorithm over it to see, if the data itself does reveal some hidden intrinsic property or not.

In the end, to improve the scalability for this, we could have tried to implement knn parallely.

3.8 Identifying Variations

The name would sound a bit confusing but the task here was to identify the variations within the language. There are several reasons for this variations. Specially, in the Indian

languages like Hindi, we can notice that the kind of Hindi that is spoken across different regions is different. The divergence of the regions thus gives rise to these variations.

The spelling differences for the same word is very crucial to identify. For this purpose, we had two options in mind. Either, to go for something which relies on identifying sounds, or to go for something, which relies on actually identifying these relations using a set of rules. We opted to go for the second one. The reason that we had was, it is very likely that a change in sound at a particular place in a particular word, may actually result in a different word. Hence, comparing the words based on sounds wouldn't be a good method. Also, in the end we would have to rely on some annotator to make sure that the pairs of word which have similar sound, are actually two different spellings for the same word. For this purpose, we choose to define a set of rules.

Our hint was that such kind of a technique would be used in case of Machine Translation in order to normalise. Hence, after reading some literature on it, we decided to implement the following set of rules:

1. Chandrabindu (a half-moon with a dot) and bindu (a dot on top of alphabet) can be used interchangeably. For example: (i) अँगरेज़, अंगरेज़ (ii) लाँच, लांच.

2. There are five consonant characters with nukta (a dot under consonant) viz. क़, ख़, ग़, ज़, फ़. With this rule, all consonants with nuktas and these consonants without nukta will be considered same. For example: (i) फोटो, फ़ोटो (ii) तेज, तेज़.

3. In order to normalize words with 'schwa' (the default vowel 'a' that occurs with every consonant), we delete all the halanth characters in the given word to generate spelling variant. For example: (i) भगवान, भगवान् (ii) अगरज, अग्रज (iii) अकसर, अक्सर.

4. 'Bindu' and 'न्' can be used interchangeably. For example: (i) कन्ठ, कंठ.

5. 'Bindu' and 'म् ' can be used interchangeably for words having 'म्' before the labial consonants like प, ब, फ, म, व in the word. For example: (i) अम्बु, अंबु (ii) पम्प, पंप.

6. There is one supplemental sound occasionally encountered in Hindi. This is the 'Visarga', noted in devanagari by the sign ('ः'). This sign appears only in tatsama

vocabulary items. The words having sign ('ः') can also be written without it and is treated equivalent. For example: (i) अंततः, अंतत (ii) अक्रमतः, अक्रमत.

7. Sometimes in place of 'ङ्'/'ण्'/'ञ' in the words, Chandrabindu (a half-moon with a dot) / bindu (a dot on top of alphabet) can be used and are equally correct. But it is very rare. For example: (i) ब्राण्ड, ब्राँड (ii) पञजा, पंजा (iii) गंङगा, गंगा.

8. 'ई' and 'यी' can be used interchangeably in words. For example: (i) नई, नयी.

9. 'ए' and 'ये' can be used interchangeably in words. For example: (i) लिए, लिये.

For each word in the corpus, we applied the above rules in order to get a normalized form of the word. For the normalized form of each word, we created a list of all the forms of the word that occur in the corpus i.e. all the different spellings of that particular word that appear in the corpus.

4 Data Set Used for different subtasks

SubTask Name	Corpus/Data Set Used
Part of Speech Tagging	Data obtained from LTRC
Entity Recognition	Hindi Annotated data in SSF form by LTRC Lab
Categorization	Data provided by Mentor (source Veooz.com)
Keyword Identification	Data obtained from LTRC

5 References

- http://airccj.org/CSCP/vol3/csit3639.pdf
- http://publications.cse.iitm.ac.in/157/1/iitmcsa.pdf
- Comparative Analysis of the Performance of CRF, HMM and MaxEnt for Part-of-Speech Tagging, Chunking and Named Entity Recognition for a Morphologically rich language - [LTRC, IIIT H]
- http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5640441&tag=1