Text Processing Framework for Hindi

IRE Major Project Presentation

Date - 14/04/16

GROUP 7

- Itisha Dewan
- Praveen Yadav
- Vaibhav Kumar

Introduction

Text Processing Framework for Hindi

Under this project our task was to do the following :

- Tokenisation
- StopWord Detection
- Sentence Breaking
- POS tagging
- Keyword Recognition
- Entity Extraction
- Categorisation
- Identifying Variations

Explanation of the various subtasks

Tokenisation

- Tokenisation is the process of breaking the stream of text into words, phrases

- Most of us would have heard this when talking about compilers

Method

- A simple regex would work in this case
- Regex can seperate out the tokens
- Caution : there are a lot of tokens, which need special care
- For example : dates, abbreviations etc.
- Also, the processing cannot be done like we would do for general English
- We require UTF scheme for this purpose

Sentence Breaking

The task here is to break the sentences individually

Methodology

- Quite a trivial task
- We just need to look at ‘poorna viraam’
- Need to handle a few edge cases
- Also, can make use of tokens
- Using the token list, identify where the ‘poorna viraam’ occurs
- Then between two such characters would lie the entire sentence

Stop Word Detection

- These are the most common words that can be found in a particular language.

Methods

Two methods used

- One relies on frequency
- Other relies on tf-idf measure

Method based on frequency

- Process the entire text
- Get a list of words present in the text
- Keep their frequency counts as well
- Sort based on the frequency count of words in decreasing order
- Select the tok k words per choice, and manually see which k would be a good choice

Method based on tf-idf

- Process the text document wise
- For each word, count its overall frequency
- For each word, count the number of documents it appeared in
- Count the total number of documents
- Using tf-idf scheme, assign value to each word
- Sort it based on decreasing order
- Choose top words in similar fashion as previously done

Part Of Speech tagging

Identify the part of speech for a particular word

Hidden Markov Model

- Implemented our own HMM without any library
- We used here Trigram model : That means tag of a word will depend on previous two words in a particular sequence
- For Decoding - Viterbi algorithm

Challenges

- to tag a particular sentence, a particular trigram was never seen before

Overcoming Challenges - Smoothing

Zipf’s law

- unseen trigrams would have some probability of occurring, which would be similar

Laplace’s smoothing

- shaves off some probability mass from that what we have seen, and provides it to that what we haven’t.

Conditional Random Field model for POS Tagging

- We used <Name Here> Library for this purpose
- Accuracy Comparison -
  CRF - 91%
  HMM - 81%
- Reason why CRF better than HMM is discussed further in slides

Categorization

Aim Task

Was to categorise the data in different news articles into a variety of topics

Various Categories

fashion ,business ,entertainment, travel, sports, automobiles, foods , criminals , health , politics, technology

Methods Available for this

- Naive Bayes Classifier
- K Nearest Neighbours

K Nearest Neighbour

- K, which were 3, 4, 5, 6, and 7.
- 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. To convert the documents into vectors, the tf-idf scheme was used.
- The implementation was entirely written by us with no other libraries used.
- To broke the tie - Used a method in which we could fall back to lower K’s until and unless we could break the tie.

Results

K Accuracy

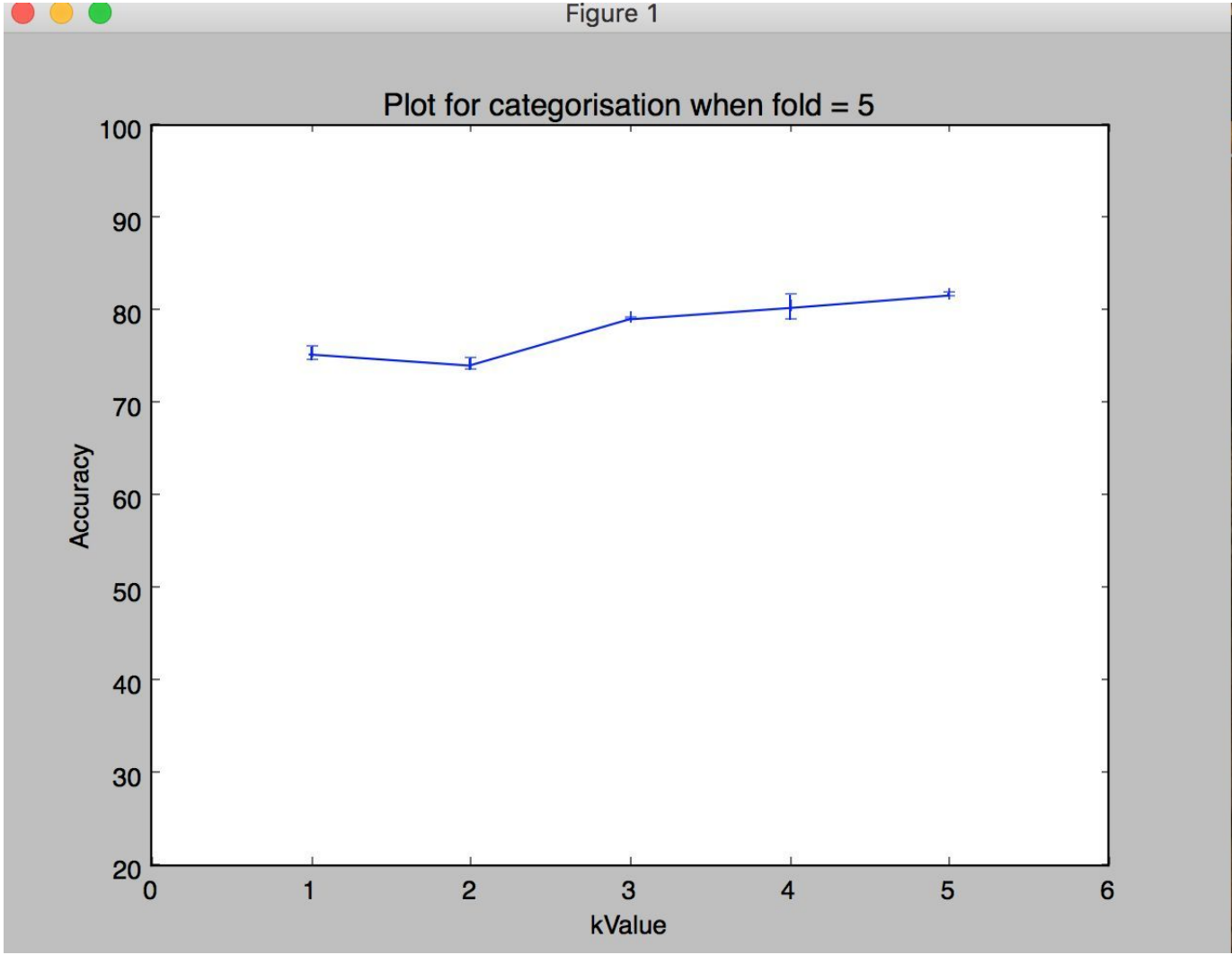
3 ~75%

4 ~74%

5 ~79%

6 ~80%

7 ~81%



Identifying Variations

What is it ?

- Definition -
  Finding Various words which are lexicographically different but means the same
- Hindi that is spoken across different regions is different.
  So There are many variations of words
- Example -
  i) अगँरेज़, अगंरेज़ (ii) लाँच, लांच.

Approaches available -

- Rely on identifying sounds which Required Soundex Library
- Using Rule-based methods to find variations

We chose second method because -

- 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

Example of Rules Created :

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

- 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) तजे, तज़े.

- 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) अकसर, अĖसर.

Named Entity Recognition

Using Conditional Random Field

Introduction

Named Entity Recognition ?

It is a subtask of information extraction that seeks to locate and classify elements in text into predefined categories such as the names of persons, organizations, locations, times, quantities, date etc.

What we recognized

ENAMEX_LOCATION_B

ENAMEX_PLANTS_B

TIME_PERIOD_B

NUMEX_COUNT_B

TIMEX_YEAR_B

Training Data

The data used for the training of the systems was provided by LTRC Lab. The annotated data uses Shakti Standard Format (SSF). For our development we have converted the SSF format data into the IOB formatted text in which a B −XXX tag indicates the first word of an entity type XXX and I −XXX is used for subsequent words of an entity. The tag O indicates the word is outside of a NE.

The training data for Hindi contains 97188 words.

Example of CoNNL format

मुगल	JJ	NP_B	ENAMEX_PERSON_B
राजवंश	NN	NP_I	ENAMEX_PERSON_I
के	PSP	NP_I	O
पतन	NN	NP_B	O
और	CC	CCP_B	O
ब्रिटिश	JJ	NP_B	ENAMEX_PERSON_B
साम्राज्य	NN	NP_ I	ENAMEX_PERSON_I

Challenges for Hindi NER

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.

Approaches Possible

1. Rule Based Approach
2. Hidden Markov Model
3. Conditional Random Field
4. Maximum Entropy Model

We chose CRF because -

1. The primary advantage of CRFs over HMM is their conditional nature, resulting in the relaxation of the independence assumptions required by HMMs in order to ensure tractable inference.
2. 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.

CRF++

- **CRF++** is a simple, customizable, and open source implementation of Conditional Random Fields (CRFs) for segmenting/labeling sequential data.
- For generic purpose and will be applied to a variety of NLP tasks, such as Named Entity Recognition, Information Extraction and Text Chunking.
- Can redefine feature sets
- Written in C++ with STL
- Less memory usage both in training and testing
- encoding/decoding in practical time

Result

Accuracy - 84%

Thank You