Indian Statistical Institute, Kolkata

M. Tech. (Computer Science) Dissertation

Lemmatizers for Highly Inflected Indian Languages

A dissertation submitted in partial fulfilment of the requirements
for the award of Master of Technology
in
Computer Science

Author:
Kamlesh Nayak
Roll No: CS-1406

Supervisor:
Dr. Utpal Garain
CVPR Unit, ISI

M.Tech(CS) DISSERTATION THESIS COMPLETION CERTIFICATE
Student: Kamlesh Nayak (CS1406)
Topic:
Supervisor: Dr. Utpal Garain

This is to certify that the thesis titled "Lemmatizers for Highly Inflected Indian Languages" submitted by Kamlesh Nayak in partial fulfilment for the award of the degree of Master of Technology is a bona fide record of work carried out by him under my supervision. The thesis has fulfilled all the requirements as per the regulations of this Institute and, in my opinion, has reached the standard needed for submission. The results contained in this thesis have not been submitted to any other university for the award of any degree or diploma.

Date: *Utpal Garain*

Acknowledgements

I would like to thank my dissertation supervisor Dr. Utpal Garain for helping me to pursue the research work embodied in this thesis. I would also like to thank Mr. Abhisek Chakrabarty, SRF, ISI, for helping me in many ways for understanding the problem and doing this work. Last but not the least I would like to thank my classmate Swarnendu Chakraborty for helping me understand the working of Lucene.

Abstract

Lemmatization is the process for finding the appropriate root for a given surface word. For morphologically rich languages one root word might have many morphological variants due agglutination or inflection and therefore, for performing tasks like question answering systems, text summarization, topic identification, word sense disambiguation, information retrieval for such languages we need good lemmatizer to find lemmata of words. This thesis considers lemmatization problem for two major Indian languages namely, Hindi and Bengali which are considered as highly inflected languages. Two different techniques have been explored under this work. Firstly, the efficiency of an off-the-shelf lemmatizer, i.e. Lemming[5], is tested for the Hindi and Bengali. Lemming does use a log linear model for lemmatization and requires parts-of-speech (POS) and lemma annotated data for learning. Experiments show that lemming performs well if we could provide sufficiently large (about twenty thousand annotated words in continuous text) data sets for Hindi and Bengali. However, for many Indian languages such a resource is not available and in the second part of this thesis, we tried to develop a graph-based unsupervised lemmatizer where the only resource requirement is a large corpus and POS tagged dataset, i.e. annotation of lemma, which is an expensive resource, is not required. Finally, role lemmatizer for Information Retrieval has been investigated for Hindi & Bengali.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Lemmatization is the algorithmic process of determining the lemma for a given word. The process involves complex tasks such as understanding context and determining the part of speech of a word in a sentence requiring, knowledge of the grammar of a language.

In various languages, words appear in several inflected forms. For example, in English, the verb 'to walk' may appear as 'walk', 'walked', 'walks', 'walking'. The base form, 'walk', that one might look up in a dictionary, is called the lemma for the word. The combination of the base form with the part of speech is often called the lexeme of the word.

Lemmatization is closely related to stemming. The difference is that a stemmer operates on a single word without knowledge of the context, and therefore cannot discriminate between words which have different meanings depending on varying part of speech. However, stemmers are typically easier to implement and run faster, and the reduced accuracy may not matter for some applications.

For instance:

1. The word "better" has "good" as its lemma. This link is missed by stemming, as it requires a dictionary look-up.
2. The word "meeting" can be either the base form of a noun or a form of a verb ("to meet") depending on the context, e.g., "in our last meeting" or "We are meeting again tomorrow". Unlike stemming, lemmatisation can in principle select the appropriate lemma depending on the context.

But for NLP tasks where context is important in those cases lemmatization performs better than stemming. Tasks like text summarization, topic identification, word sense disambiguation, sentiment analysis, question answering systems, etc., lemmatization usually performs better than stemming.

Morphological rich languages are either agglutinative or inflectional. An agglutinative language is a type of synthetic language with morphology that primarily uses agglutination,i.e., words may contain different morphemes to determine their meaning, but each of these morphemes (including stems and affixes) remains in every aspect unchanged after their union, thus resulting in generally easier deducible word meanings if compared to inflectional languages, which allow modifications in either or both the phonetics or spelling of one or more morphemes within a word, generally for shortening the word on behalf of an easier pronunciation. Agglutinative languages tend to have a high rate of affixes or morphemes per word, and to be very regular, in particular with very few irregular verbs. Uralic languages like Finnish, Hungarian and Sami languages are agglutinative. Indian languages however are mostly inflectional. Bengali and Hindi both are highly inflectional languages. The performance of Lemming for Indian languages such as Hindi & Bengali has never been evaluated before.

1.1 Thesis Outline

The thesis is organised as follows. Chapter 1 gives an introduction about lemmatization, and various approaches to lemmatization have been described. Chapter 2 describes the state of the art approach for supervised lemmatization and its performance for hindi and bengali. Chapter 3 outlines the details of our proposed scheme of graph based unsupervised lemmatization. Chapter 4 we evaluate the performance of lemmatization in Information Retrieval for Hindi and Bengali. Finally, in chapter 5, we conclude the thesis and also mention our future goals with respect to our novel scheme.

1.2 Rule based Lemmatization Approaches

1.2.1 Levenshtein Distance and Dictionary based Approach

Levenshtein distance is a metric used to measure the difference between two sequences. Levenshtein distence is the minimum number of characters which needs to be inserted, deleted or substituted to transform one string to another. Mathematically, the Levenshtein distance between two strings a, b (of length $|a|$ and $|b|$ respectively) is given by $lev_{a,b}(|a|, |b|)$ where,

$$lev_{a,b}(i,j) = \begin{cases} max(i,j) & \text{if } min(i,j) = 0 \\ min \begin{cases} lev_{a,b}(i-1,j) + 1 \\ lev_{a,b}(i,j-1) + 1 \\ lev_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise} \end{cases}$$

where $1_{(a_i \neq b_j)}$ is the indicator function equal to 0 when $a_i = b_j$ and equal to 1 otherwise, and $lev_{a,b}(i,j)$ is the distance between the first i characters of a and the first j characters of b.

There is a file containing 30,000 possible lemmas. The algorithm compares user's input word with all target lemmas and the Levenshtein distance for each source and target word is calculated and stored. After completion of the process, target lemmas with minimum Levenshtein distance from source word is returned.

The algorithm also provides the option of selecting a value for approximating the distance between the source word and the target words. Lets say that the user enters the approximation value as 0, in that case all the target words returned have minimum Levenshtein distance from the source word but if, approximation value is 2 then all target words returned have a Levenshtein distance $\leq$ (minimum+2) from the source word.

This approach also distinguishes between words like entertained and entertainment, since entertain is the lemma for entertained but not for entertainment. Entertainment is a noun and different from entertained.

1.2.2 Affix Lemmatizer

In this approach lemmatization rules are trained automatically to handle prefix, suffix and infix changes to generate lemma from the surface word. Here is an example of how this training algorithm works. Lets say that we have the english word-lemma pair :
baking $\rightarrow$ bake.
If it were the sole input to the training program then it would have produced the following lemmatization rule :
*ing $\rightarrow$ *e
The asterisks are wildcards and placeholders. With the above rule the lemmatizer will be able to construct correct lemma for some other words as well that has not been used in training. For example, "waste" is the lemma for "wasting", which can be produced by the lemmatization rule. A similar approach has been used in BenLem(A bengali lemmatizer) by Chakrabarty et.al (2016).

Building a rule set for training pairs

For a program to construct all the lemmatization rules an extended list of surface word-lemma pairs are required that the program can learn from-at least tens of thousands and possibly over a million entries. The most important task that the training algorithm must perform is that it must correctly find the lemma for out of vocabulary(OOV) words.

The algorithm is trained to create a data structure consisting of rules that the lemmatizer must traverse in order to arrive at a rule which is used for the transformation. The training process is such that while the data structure is being built,each surface word is being tentatively lemmatized using the data structure that has been created upto that stage. If the rule selected produces the right lemma for the surface word, nothing is done. Otherwise, the data structure is expanded with a new rule such that the new rule a) is selected instead of erroneous rule and b) produces the right lemma for the surface word. The training process terminates when the full forms in all pairs in the training set are transformed to their corresponding lemmas.[2]

Internal structure of rules: prime and derived rules

During training the Ratcliff/Obershelp algorithm (Ratcliff and Metzener 1988) is used to find the longest non-overlapping similar parts in a given surface word lemma pair. For example, in the pair **afgevraagd** → **afvragen** the longest common substring is vra, followed by af and g. These similar parts are replaced with wildcards and placeholders:
*ge*a*d → ***en
The above rule is the prime rule for the training pair, i.e. the least specific rule to lemmatize the surface word correctly. Derived rules are rules with more specific pattern which are created from the prime rules by adding characters, and by adding or removing wildcards. A surface word which is matched by a pattern of a derived rule is also matched by the pattern of the original rule but vice-versa is not true.
A large number of rules can be derived from a rule with at least one wildcard in its pattern, but only a limited number can be actually tested. So, in order to keep the number of candidate rules in check, the following strategy was used, the pattern of a candidate is minimally different from its parents pattern: it can have one extra literal character or one wildcard less or replace one wildcard with one literal character.

External structure of rules in a tree

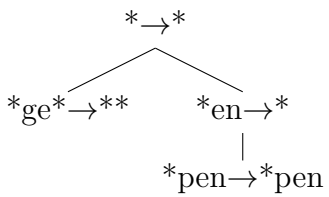

Figure 1: Rule tree

The above figure represents the way in which the lemmatization rules are represented by a tree. The left to right preferential ordering is followed between the children of a rule. Hence, only one rule fires for a surface word and one lemma is produced.For example, because the rule *ge* → ** precedes its sibling rule *en → *, whenever the former rule is applicable, the

latter rule and its descendants are not even visited, irrespective of their applicability. The tree implements negation ,i.e., if the N^{th} sibling of a row of children is fired, then it means that the previous N-1 children did not match the surface word.

Rule Selection Criteria

The lemmatization rules are added to the tree in online fashion. During training if the tree that had been formed makes lemmatization errors in that case one or more corrective children have to be added to the rule.
If the pattern of the new child rule also gives erroneous lemma for some training words that are incorrectly lemmatized by the parent rule then in that case a right sibling rule needs to be added.This is repeated until all training words that the parent does not lemmatize correctly are matched by the leftmost child rule or one of its siblings.

1.3 Data Driven Morphological Analysis

Morphological analysis usually consist of two subtasks : the assignment of morphological features to a wordform and lemmatization. In this technique lemmatization and morphological analysis is performed as a classification task. (Chrupala 2006) has taken a different approach to lemmatization. Lemma classes are automatically created in this technique. Shortest edit script between reversed word-form and the corresponding lemma is regarded as the class label. Then a classifier is used to tag a particular word-form to corresponding lemma class. On application of the tagged lemma class,i.e., edit script on the word-form the lemma is generated. This approach to lemmatization is quite similar to POS tagging or morphological tagging.

Architecture

This system is made up of two learning modules one for morphological tagging and one for lemmatization and one decoding module which searches for the best sequence of pairs of morphological tags and lemmas for an input sequence of wordforms. Both modules learn Maximum Entropy Classifiers. For the lemmatization model (Chrupala, 2006)'s method of inducing lemma classes is used.[4]
The class assigned to a word-form - lemma pair is a shortest edit script(SES). Shortest edit script between two words is defined as the minimum insertions or deletions at different positions of the word, that needs to be carried out in order to transform one word to another. For example if we want to transform a word "wasting" to "waste" then in that case the SES is as follows {<D,i,4>,<I,e,6>,<D,n,6>,<D,g,7>}.

The instructions can be interpreted as :

- delete character i at position 4
- insert character e before position 6
- delete character n at position 6
- delete character g at position 7

Since most of the inflections are at the end of the words hence, the strings are reversed before computation of the SES. So, the pair "wasting" and "waste" becomes "gnitsaw" and "etsaw". So, the SES for this pair becomes
{<D,g,1>,<D,n,2>,<D,i,3>,<I,e,5>}.
This SES can also be applied for the pair "baking" and "bake" in reversed form.

Features

In this architecture minimalistic language independent features have been used. This helps in the generalization but for better performance it is desirable to use some language and domain specific features.
For morphological tagging the following list of features are used :

- the word-form, lemma and morpho-tag of the previous two words
- suffixes of length 1-7 of the word-form
- prefixes of length 1-5 of the word-form
- spelling pattern of the word-form i.e., upper-case and lower-case letter, digits, hyphens, underscores and other punctuation.
- part of speech of the previous two words

For the lemmatization model a similar but smaller feature set is used :

- morpho-tag(i.e predicted) of the word-form
- suffixes of length 1-7 of the word-form
- prefixes of length 1-5 of the word-form
- spelling pattern of the word-form
- lowercased word-form of the focus token

Search

Maximum entropy models are trained on examples so that it can predict probability distributions over classes (i.e., morpho-tags and lemma-classes) for the focus word-form. It uses the context for the prediction since the context has been embedded as features for the classifier. That is the focus word w_i with context $c \in C$ for each possible morpho-tag $m \in M$ the morpho-tagging model gives $p(m|c)$, and for each possible lemma-class $l \in L$ the lemmatization model gives $p(l|c, m)$. The context also includes the focus word-form as well as the preceding and following word-forms in the same sentence.
The algorithm is a beam search which maintains a list of n-best sequences of $(m, l) \in M \times L$ (morpho-tag lemma-class) pairs up to the current position in the input word sequence. The conditional probability of a candidate sequence for words $(w_0..w_i)$ is given by :

$$P(m_0..m_i, l_0..l_i|w_0..w_i) = p(l_i|c_i, m_i)p(m_i|c_i)P(m_0..m_{i1}, l_0..l_{i1}|w_0..w_{i1})$$

The search proceeds as follows: for focus word w_i there are n (n being the beam size) highest probability sequences $((m_0, l_0)..(m_{i1}, l_{i1}))$. For each of those sequences we obtain a morpho-tag probability distribution from the morpho-tagging model. For efficiency reasons we pre-prune this set of tags: given the list of tag probabilities $(m_0, p_0)..(m_j, p_j)$ sorted in decreasing order, we keep all the tags $(m_0..m_i)$ where p_i satisfies the condition:

$$p_i / \sum_{k=0}^{i} p_k < T$$

where T is a threshold parameter. Each of the retained morpho-tags for word w_i is added to each candidate sequence and for each of those combinations we obtain lemma-class probability distribution from the lemmatization model. The lemma-class set is pruned according to the same method as for morpho-tags. The probability of candidate sequences is updated according to equation 2, and the n highest ranking candidate sequences for $w_0..w_i$ are retained as the algorithm proceeds to word w_{i+1}[4].

Chapter 2

LEMMING - A log-linear Lemmatizer

LEMMING is a modular log-linear model which does joint modeling for lemmatization and tagging and also integrates global features into the model. It can be trained on corpora annotated with gold standard tags and lemmata and does not depend on morphological dictionaries or analyzers. Lemmata is an important requirement whenever we have to map words to lexical resources and establish the relation between inflected forms, particularly critical for morphologically rich languages to address the sparsity of unlemmatized forms.[5]

2.1 Log-linear model

Chrupala (2006) has defined a new technique where lemmatization is performed as a classification task through the pre-extraction of edit operations transforming forms into lemmata. This technique adopts the same philosophy but with a few modifications. Formally, lemmatization is a string-to-string transduction task. Given an alphabet $\sum$, it maps an inflected form $w \in \sum^*$ to $l \in \sum^*$. This process is modelled by log-linear model :

$$p(l|w,m) \propto h_w(l).exp(f(l,w,m)^T\theta),$$

where f represents hand-crafted feature functions, θ is a weight vector, and $h_w : \sum^* \rightarrow \{0,1\}$ determines the support of the distribution, i.e, the set of candidates with non-zero probability[5].

2.1.1 Candidate selection

The choice of the support function $h(.)$ is very important for successful working of the model. Too permissive a function will lead to rise in computation cost while too restrictive may cause the correct lemma to receive no probability mass. In extension to Chrupala's work a deterministic pre-extraction of edit trees is used to define $h(.)$. To extract an edit tree for a pair form-lemma $< w, l >$, the longest common subsequence is found between

them. This process is continued recursively to find all prefix and suffix pairs. When no LCS is found the string pair is represented as a substitution operation which transforms the first string to the second. The LCS's are not encoded into the edit tree instead the length of the suffixes and prefixes and the substitution nodes are encoded into the edit tree, as show in the figure below.

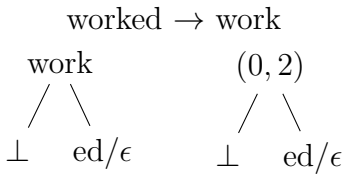

Figure 2: Edit tree for worked → work. The right tree is the actual edit tree used in the model, the left tree visualises what each node corresponds to.

The tree in fig.2 also transforms touched into touch. From the form-lemma pairs of the training set all possible edit trees are extracted, each of which may be applied to more than one pair $< w, l >$. In order to generate the candidates of a word-form all possible edit trees are applied onto the word-form (note that only a small subset of the edit trees can be applied to a word-form because most require incompatible substitution operations).

2.1.2 Features

Edit tree features The edit tree itself is a feature.The form-lemma pair $< w, l >$ is used as a feature so that irregular forms can be memorized by the model,(eg. the lemma of *was* is *be*).For each surface word affix upto length 10 is also used as a feature.

Alignment features Alignment features can be derived from an edit tree by aligning the characters in LCS nodes character by character and characters in the substitution nodes block-wise. Thus, the alignment of worked - work is : w-w, o-o, r-r, k-k, ed - ϵ. Each alignment pair constitutes a feature in the model[5].

Lemma features The lemma itself is used as a feature, such that the common lemmata for the language can be learned. Prefixes and suffixes of the lemma are used as features.

POS & morphological attributes Along with the above features POS information and the morphological attributes are used as features.

2.2 Joint Tagging and Lemmatization

Lemming is a pipeline model, i.e. at first we need to train to get a model for morphological tagging.(Here CRF model is used for training). The lemmatization and CRF components are combined in a tree-structured CRF. Given a sequence of forms w with lemmata l and morphological+POS tags m, the global normalized model is as follows:

$$p(l, m|w) \propto \prod_i h_{w_i}(l_i) exp(f(l_i, w_i, m_i)^T \theta + g(m_i, m_{i-1}, m_{i-2}, w, i)^T \lambda),$$

where f and g are the features associated with lemma and tag respectively and θ and λ are weight vectors. The graphical model is shown in fig. 3 below. The parameters are estimated by Stochastic gradient descent [7].

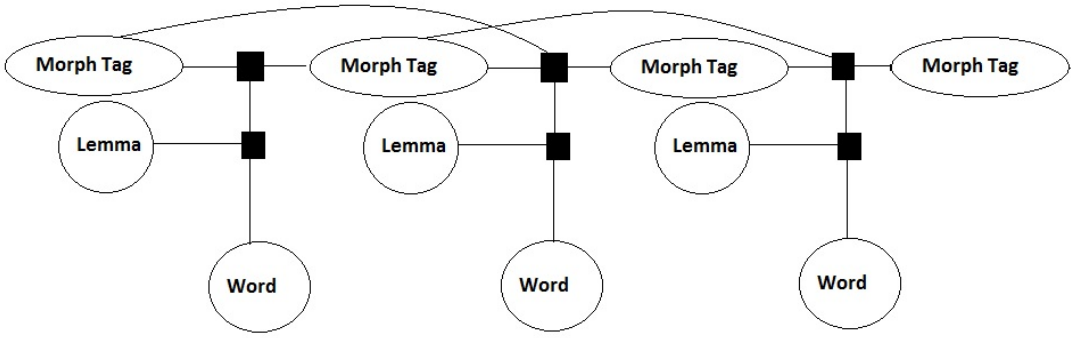

Figure 3: 2nd-order linear chain CRF

2.3 Experiments & Results

Datasets

LEMMING has been trained on datasets of two languages, Bengali and Hindi. Bengali dataset has been manually annotated by a Bengali linguist. The dataset contains 27095 samples, each of which consist of a surface word, lemmata for the word ,the POS tag of the word and is accompanied by its contextual neighbours. The number of POS tags in Bengali dataset is 10. The Hindi dataset has been prepared by merging datasets of three different domains. The Hindi dataset consists of 939040 instances which has the same format as that of the Bengali dataset. Word Sense Disambiguation dataset for Health domain and Tourism domain has been used which was used in Khapra et.al (2010). In this dataset most of the words had "unk" POS tag which means that they were not defined. This dataset was tagged with only 4 POS tags (i.e. adjective, verb, noun, adverb). The third dataset is COLING 2012 shared task data for dependency parsing dataset which had 10 POS tags and most the words in the dataset were tagged correctly. So, in order to tackle this disparity in POS tags of COLING dataset and the Health and Tourism dataset a MARMOT[5] POS tagger was trained on the COLING dataset for POS tagging. This tagger was then used to tag

the Health and Tourism dataset for those words which had "unk" as their POS tags. These three datasets then combined form a large manually annotated dataset for Hindi which was used for training of LEMMING.

Results

We have used 4-fold cross validation on the dataset to obtain the accuracy for Hindi and Bengali. The accuracy is simply the ratio of the number of surface words which were correctly lemmatized to the total number of surface words in the dataset. The accuracy for both the dataset is highlighted in the table shown below.

Language	Accuracy
Hindi	91.81
Bengali	91.23

Table 1: LEMMING results for Hindi and Bengali

Chapter 3

Graph Based Unsupervised Lemmatization

For languages where a manually annotated dataset for lemmatization exist LEMMING performs very well. Languages for which no such annotated dataset exist lemmatization becomes a challenge. So, here we have tried to perform lemmatization, where the only resource needed is a corpus(i.e. large collection of text) along with POS tagger for a particular language. Our technique creates an annotated dataset by just using the corpus of the language which can then be used to train LEMMING. In order to check the accuracy of our technique we have tested it on languages for which there exist annotated dataset. The dataset created by our algorithm is compared with the manually annotated dataset for finding the accuracy.

Bengali and Hindi are highly inflected languages where one root word might have many morphological variants. Most of these variants are created by addition of suffixes to the root words. So, in here we have created the graph based on the suffixal inflections.[8] We start with a word list extracted from the Hindi news corpus of 292686 documents and a second word list extracted from the Bengali news corpus of 457315 documents. The Hindi word list consists of 448855 unique words and Bengali word list was made up of about 1268352 unique words.

The technique is similar to GRAS (GRAph based Stemmer) by Paik et.al(2011). Word2Vec tool[9] has been used to find the distributed representation of all words in the word list. The edge weight is calculated by finding the cosine similarity between the connected words (i.e. nodes). So, the weighting of the edges takes into account the semantic properties of the words rather than the only taking the orthographic similarity between them.

3.1 Graph Construction

Let G = (V, E) be a graph such that V represents the set of vertices and E represents the set of edges. Each word in the word list is a vertex in the graph. The edges in the graph represent the **transformation rules**. Let w_1, w_2 be two words such that they share a common prefix such that $w_1 = ps_1$ and $w_2 = ps_2$, the transformation rule for converting $w1$ to $w2$ is $< s_1 \rightarrow s_2 >$. For example for the word pair $< baking, bake >$ the transformation rule is as follows, $< ing \rightarrow e >$.

The statistically relevant suffix list is derived from the word list. The suffixes of length $1 - 7$ is derived from the word list and their frequency is calculated. These suffixes are sorted according to their frequencies. A graph is plotted with X-axis representing the suffixes and Y-axis representing their frequencies. The suffixes in X-axis are arranged in descending order of their frequencies. An interval of 10 suffixes is considered along X-axis and in that interval the average frequency value of the suffixes is calculated on the Y-axis let it be x. For the next consecutive interval average is calculated let it be y. The ratio $y : x$ is calculated if this value is greater than 0.95 then the last suffix of the interval for which x is average frequency is considered as cutoff and all suffixes left of it are considered as relevant and rest are discarded.

The word list is segregated into different files where each file contains words which begin with the same alphabet. So, all words beginning with the same alphabet is put into the same file. For Hindi a total of 44 files were formed and for Bengali 57 files were created. Transformation rules to transform one word to another along with their respective frequency's is derived for each file by comparing each word with rest of the words in the file. Then the total frequency for each transformation rule is derived for all files. It is computed by adding the counts of transformation rules for all files. The transformation rules whose frequency is above a certain threshold θ is considered as valid and rest are discarded. The optimal threshold value is experimentally found.

Algorithm 1: Derivation of relevant transformation rules

Result: Relevant transformation rules

Let $Z = \{F_1, F_2, ..., F_n\}$ be the set of files formed after segregation of words in the word list as per their first alphabet;

Let $S = \{s_1, s_2, ..., s_m\}$ be the relevant suffixes derived earlier;

for *each* $F \in Z$ **do**

  Let $F = \{w_1, w_2, w_3, ...\}$;

  for *all possible word pairs* $w_i, w_j \in F$ **do**

    Let $s_1^{'}\&s_2^{'}$ be the suffixes such that $w_i = ps_1^{'}, w_j = ps_2^{'}$, and p is the longest common prefix of $< w_i, w_j >$;

    if $s_1^{'}\&s_2^{'} \in S$ **then**

      if $|w_i| > |w_j|$ **then**

        Output the transformation rule $< s_1^{'} \rightarrow s_2^{'} >$ along with frequency;

      else

        if $|w_i| < |w_j|$ **then**

          Output the transformation rule $< s_2^{'} \rightarrow s_1^{'} >$ along with frequency;

        else

          Output the transformation rule $< s_1^{'} \rightarrow s_2^{'} >$ and $< s_2^{'} \rightarrow s_1^{'} >$ along with frequency;

        end

      end

    end

  end

end

Compute the total frequency after frequency for each file has been calculated for each transformation rule;

The computed relevant transformation rules is used for connecting the edges of the directed graph. The graph for Hindi basically consists of 44 disjoint components where each component represents a list of words of a file out of the 44 files created as per the first alphabet of the words. Similarly, for Bengali 57 disconnected components are formed. The word graph is directed since the transformation rules derive one word from another. The rules are formed such that always the smaller length word is always derived from the larger length word by the transformation rule. For words with the same length the transformation rule of derivation of both the words from each is derived, so, in this case there are edges in both direction of the word nodes. This assumption is used since the surface word is formed by adding an inflection to the lemma. Hence, the length of the surface word is bigger than the lemma. A small glimpse of the graph constructed is shown below :

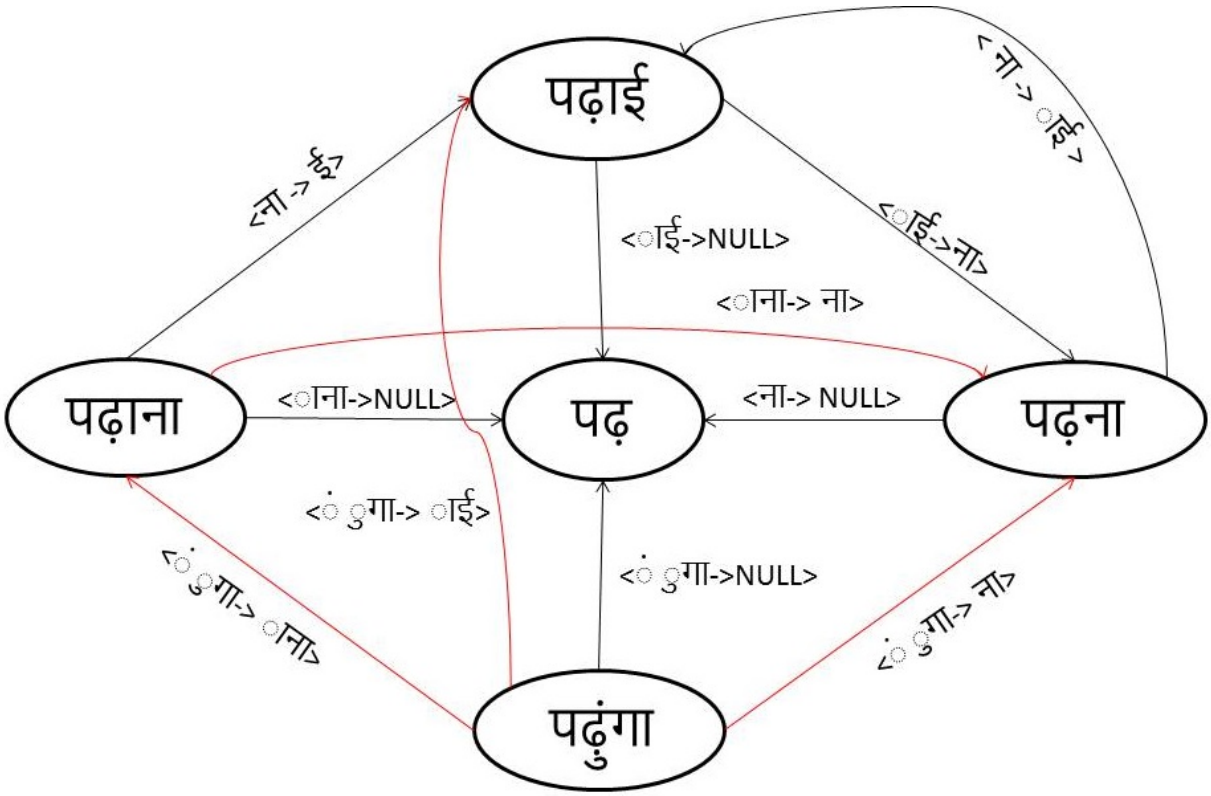

Figure 4: Word Graph : The edges represented in red has been pruned

As can be seen from the figure that the graph is dense and finding the correct lemma for a word is quite difficult.To find the lemma from the graph is not possible so in this part we need help of a dictionary of root words. So, we need to find the root words from the graph but for doing so we need to find the similarity quotient between the connected nodes to find the semantic similarity between them.

3.2 Computation of Similarity Quotient

Word2Vec tool[9] was used to find the vector representation of all words in the word list. Word2Vec was trained on the corpus of documents for Hindi and Bengali. On training Word2Vec on a corpus it returns a ".vec" file which contains 200 dimensional representation of all words in the corpus. So, we get two ".vec" files one for Hindi and the other for Bengali. Using the vector representation of two words their similarity quotient can be calculated by finding the cosine similarity of the two vectors.

$$similarity\ quotient(\vec{w_1}, \vec{w_2}) = \frac{\vec{w_1}.\vec{w_2}}{|\vec{w_1}|.|\vec{w_2}|},$$

where $\vec{w_1}$ & $\vec{w_2}$ are vector representation of words w_1 & w_2.
This metric is used to give weight to the edges of the word graph. A threshold γ is set for pruning out the edges which connect nodes which are not semantically similar. The threshold is calculated by taking the help of 1000 samples of surface word-lemma pairs

which were manually annotated. The similarity quotient for each of these pair is found. The mean and standard deviation of the similarity quotient of 1000 samples is found. The threshold is defined as

$$\gamma = Mean - Std.\ Dev.$$

The edges having similarity quotient below γ is removed. Finally, the remaining edges in the graph represents the edges which connect nodes that are both orthographically and semantically similar.

3.3 Automated Dictionary Creation

The word graph obtained is completely pruned and the final graph can be used to find the root words. The total quotient of a node in the graph is defined as the sum of similarity quotient of all edges connected with the node.

$$node\ quotient(x) = \sum_{x \in E} similarity\ quotient(e)$$

where E set of edges, x is the node for which we are computing the node quotient and e are the edges which are coonected to node x.
This quotient for each node defines that the nodes with higher quotient are connected with a large number nodes with whom it is also semantically similar. So, the nodes with high quotient values are considered as the root nodes in the graph. The threshold value δ is defined such that the nodes with quotient above δ are considered as the root nodes. The value of δ is defined experimentally. After this computation we get a list of root words which can be regarded as dictionary root words. This list of root words is utilised to find the surface words which are produced by adding suffixes to the root words.

The nodes which are the root nodes are coloured to distinguish them from the other nodes. All edges which connect does not connect a root word and surface word is removed. The edges also connecting two root words are removed. By this the graph remaining contains only edges between surface words and root words. There also might be cases where a surface word is connected with two root words for these cases the similarity quotient is checked to find the root to which it is more similar. The edge which has lower similarity quotient is removed for the surface word. So, we get cluster of connected components for each root word connected with each of its surface words.

3.4 Creation of Training Set for LEMMING

For the creation of training set to train LEMMING we need a POS tagged list of words accompanied by its contextual neighbours. For each of the surface word in the list to find the lemma of the word we use the graph and find the root word to which the surface word is connected to. If the surface word is not connected to any root word in the graph in that case the lemma is same as the surface word. By this method we get a list of words with POS tag and lemma accompanied by its contextual neighbours. This can be used for training LEMMING.

3.5 Experiments & Results

The corpus used for obtaining the word list for Hindi and Bengali is taken from FIRE AD HOC 2011 which contained 292686 documents for Hindi and 457315 documents for Bengali. The parameters θ & δ needs to be computed through experimentation and the technique performed best for values mentioned below. The value of γ for both languages was found out to be :

$$\begin{aligned} \gamma(Hindi) &= 0.24 \quad \gamma(Bengali) = 0.32 \\ \delta(Hindi) &= 0.73 \quad \delta(Bengali) = 1.35 \\ \theta(Hindi) &= 350 \quad \theta(Bengali) = 700 \end{aligned} \tag{3.1}$$

The training set created by the above technique is compared with the manually annotated dataset which was used for training LEMMING for both languages. The accuracy of the created training set is the ratio of the number of lemma's correctly tagged to the total number of surface words. The results for Bengali & Hindi are as follows :

Language	Accuracy
Hindi	57.38
Bengali	60.92

Table 2: Results for Hindi and Bengali by Unsupervised Lemmatization

Chapter 4

Effects of Lemmatization on IR

There are various ways of improving the performance of the search engine by techniques like stop word removal, stemming and lemmatization. Here we have compared the performance of lemmatization against stemming for Hindi and Bengali. The model created by training LEMMING on manually annotated datasets for Hindi and Bengali has been used to lemmatize Hindi and Bengali documents along with the queries. The Hindi and Bengali documents and queries has been taken from FIRE 2011 AD HOC. The qrel files to find the mean average precision(MAP) and P@10 for the queries.

4.1 Lemmatization of Documents & Queries

The log-linear model created by training LEMMING is used for lemmatization of the documents and queries. The documents needs to be merged into one large file for lemmatization. LEMMING takes as input one token per line, so, the documents were merged into one file such that there is one token per line. Similarly, the queries are also merged into one file of the same format. Once the lemmatized file is returned it is in the form of a large file with each line containing the surface word with its POS tag and the lemma for it. This file was divided into individual files representing the documents from which they were merged before. The change in the new documents is that the surface words in the documents has been replaced by their respective lemmata. Simlarly the query words were replaced by their respective lemmata. So, now we have large collection of documents for both Hindi and Bengali where each surface word has been replaced by their respective lemmata.

4.2 Indexing of Documents

Indexing of the documents has been done by using Lucene version 4.9. The Bengali documents has been stemmed by a rule based stemmer developed by Ganguly et.al (2012). Hindi stemmer available in Lucene is used to stem the contents of the Hindi documents. Two indices were created for each language. An index is created for the lemmatized documents and another for the stemmed documents. The index has two fields, document ID which stores the document name and document text.

4.3 Query Search

The queries are taken from FIRE 2011 AD HOC for Bengali and Hindi. There are 50 queries in the dataset. Qrel(i.e. query relevance) files are given for Bengali and Hindi for finding the mean average precision(MAP) and P@10. The scoring function used is BM25. BM25 is a bag-of-words retrieval function that ranks a set of documents based on the query terms appearing in each document, regardless of the inter-relationship between the query terms within a document (e.g., their relative proximity). It is not a single function, but actually a whole family of scoring functions, with slightly different components and parameters. One of the most prominent instantiations of the function is as follows.

Given a query Q, containing keywords $q_1, q_2, ..., q_n$, the BM25 score of document D is :

$$score(D,Q) = \sum_{i=1}^{n} IDF(q_i).\frac{f(q_i,D).(k_1+1)}{f(q_i,D)+k_1.(1-b+b.\frac{|D|}{avgdl})},$$

where $f(q_i, D)$ is q_i's term frequency in document D, $|D|$ is the length of the document D in words, and avgdl is the average document length in the text collection from which documents were drawn. k_1 and b are free parameters, usually chosen, in absence of advanced optimization, as $k_1 \in [1.2, 2.0]$ and $b = 0.75$. In our case the value of $k_1 = 1.5$ & $b = 0.75$. $IDF(q_i)$ is the inverse document frequency weight of the query term q_i. It is usually computed as :

$$IDF(q_i) = \log \frac{N - n(q_i) + 0.5}{n(q_i) + 0.5}$$

where N is the total number of documents in the collection, and $n(q_i)$ is the number of documents containing q_i.

4.4 Experiments & Results

The queries provided in FIRE 2011 are divided as per query length and type. There are three different representation of the same query. The queries are tagged as "title"(T), "desc"(D) and "narr"(N), where "desc" is basically a description of the title and narration is the expanded version of query. Treceval 9.0 tool and the qrel files provided in FIRE 2011 has been used to check the MAP and P@10. The qrel file contains a list of relevant documents for each query. The results for Hindi and Bengali are as follows :

		T	TD
Lemming	MAP	0.1226	0.1413
	P@10	0.3460	0.3740
Stemmer	MAP	0.1298	0.1442
	P@10	0.3500	0.3760

Table 3: IR-Results for Hindi for qrel (version-1)

		T	TD
Lemming	MAP	0.2441	0.2663
	P@10	0.3880	0.4200
Stemmer	MAP	0.2476	0.2665
	P@10	0.3880	0.4200

Table 4: IR-Results for Hindi for qrel (version-2)

		T	TD
Lemming	MAP	0.2603	0.3363
	P@10	0.4140	0.4940
Stemmer	MAP	0.2544	0.3361
	P@10	0.4240	0.5200

Table 5: IR-Results for Bengali

Chapter 5

Conclusion & future work

Lemmatization is a major task for highly inflected languages. In order to perform various NLP tasks we need the lemmata for such languages. Lemming is a log linear model which takes into account a global set of language independent features for training of the model. The lemmatizer was trained on two major Indian languages Hindi & Bengali which are highly inflected. The results for Hindi & Bengali were very promising. The bottleneck is that for training Lemming we need a sufficiently large data set. Since, this resource is not available for many Indian languages we tried to develop a graph based unsupervised lemmatization technique which requires only a large corpus and POS tagged dataset.

The graph based unsupervised lemmatization created an annotated dataset without any manual intervention. This created annotated dataset could be used to train Lemming. There was a gap in the quality of annotated dataset created by this technique as compared to the manually annotated dataset, hence the results were not so good. This is the first time such a technique for unsupervised lemmatization was tried and there is great scope for improvement of the technique. A dictionary is created as a byproduct of the technique but about 35% of the dictionary words were found out to be actual root words from dictionary. So, there is also scope for improvement in this front.

Information retrieval follows a bag of words approach, by lemmatization we try to impart language understanding into the IR engine. After conducting our experiment the results of lemmatizer against stemmer were almost the same. Since, a lemmatizer is a expensive tool to implement, stemmer is better for Information retrieval since it takes less time for stemming and gives almost same MAP as that of a lemmatizer. But further investigation is required to ascertain that stemmer would be a better tool than lemmatization by comparing for other datasets.

Bibliography

[1] Dimitrios P. Lyras, Kyriakos N. Sgarbas, Nikolaos D.Fakotakis, "Using the Levenshtein Edit Distance for Automatic Lemmatization: A Case Study for Modern Greek and English," Tools with Artificial Intelligence, 19th IEEE International Conference , pp.429-435, 29-31 October,2007.

[2] Bart Jongejan, Hercules Dalianis,"Automatic training of lemmatization rules that handle morphological changes in pre-, in- and suffixes alike", 47th Annual Meeting of the Association for computational linguistics (ACL) and the 4th International Joint Conference on Natural Language Processing (IJCNLP) of the AFNLP, p.p.145-153, August 2009.

[3] Grzegorz Chrupala. 2006. Simple data-driven context-sensitive lemmatization. *Procesamiento del Lenguaje Natural*.

[4] Grzegorz Chrupala, Georgiana Dinu, and Josef Van Genabith. 2008. Learning morphology with Morfette. In *Proceedings of LREC*.

[5] Thomas Müller, Ryan Cotterell, Alexander Fraser, Hinrich Schütze, Joint Lemmatization and Morphological Tagging with LEMMING. *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 22682274, September 2015.

[6] Dan Gusfield. 1997. *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology*. Cambridge University Press.

[7] Yoshimasa Tsuruoka, Junichi Tsujii, and Sophia Ananiadou. 2009. Stochastic gradient descent training for l1-regularized log-linear models with cumulative penalty. In *Proceedings of ACL-IJCNLP*.

[8] Jiaul H. Paik, Mandar Mitra, Swapan K. Parui and Kalervo Järvelin. GRAS : An Effective and Efficient Stemming Algorithm for Information Retrieval, *ACM Transactions on Information Systems* (TOIS), Volume 29, Issue 4, 2011.

[9] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). *Distributed representations of words and phrases and their compositionality*. In Advances in neural information processing systems, pages 3111−3119.

[10] Mitesh M. Khapra, Anup Kulkarni, Saurabh Sohoney and Pushpak Bhattacharyya, All Words Domain Adapted WSD: Finding a Middle Ground between Supervision and Unsupervision, *Conference of Association of Computational Linguistics* (ACL 2010), July 2010.

[11] Thomas Müller, Helmut Schmid, and Hinrich Schütze. 2013. Efficient higher-order CRFs for morphological tagging. *In Proceedings of EMNLP.*

[12] Debasis Ganguly, Johannes Leveling, and Gareth J. F. Jones, DCU@ FIRE-2012: Rule-based stemmers for bengali and hindi.

[13] Abhisek Chakrabarty and Utpal Garain, BenLem (A Bengali Lemmatizer) and Its Role in WSD, *ACM Trans. Asian Low-Resour. Lang. Inf. Process.*, Vol. 15, No. 3, Article 12, 2016.

[14] Abhisek Chakrabarty, Akshay Chaturvedi and Utpal Garain, A Neural Lemmatizer for Bengali, *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*, 2016.

[15] Pushpak Bhattacharyya, Ankit Bahuguna, Lavita Talukdar and Bornali Phukan, Facilitating Multi-Lingual Sense Annotation: Human Mediated Lemmatizer. *Global WordNet Conference*. 2014