Cognate Detection to Improve Phylogenetics for Indian Languages

A Progress Report

submitted in partial fulfillment of the

requirements for the degree of

Bachelor Of Engineering

by

Yashasvi Mantha
(121710312059)

under the guidance of

Prof. Pushpak Bhattacharyya,
and
Diptesh Kanojia

Center for Indian Language Technology
Indian Institute of Technology Bombay
Mumbai

May - July 2019

Abstract

Phylogenetics is used to understand the diachronic relationships between various taxa, and different computational methods are employed for the improvement of phylogenetic methodologies. The use of computational techniques for estimating evolutionary historical relationship of languages/ancient texts (i.e. Computational Phylogenetics), has seen a tremendous increase over the past 20 years. In this work, we utilize distance matrix based computational algorithms to visualize, infer and improve phylogenetic trees. Our work analyzes different approaches for the construction of phylogenetic trees for 14 different Indian Languages. We perform an extensive literature survey and list phylogenetic methods which can help use perform the said task. We compute the distance between each language based on IndoWordNet (Bhattacharyya, 2010) data and infer what is more specifically known as typological trees (type of phylogenetic trees). To compute the distance matrix for inferring the aforementioned trees, we use cross-lingual word embeddings based method as a novel approach over the baseline approach of using orthographic similarity based methods. We also detect cognate word-pairs for each language pair and infuse them with WordNet data to compute more accurate distance matrices. With this work, we aim to find methods that would explain the evolution of Indian Languages in an objective manner and we further aim to improve on the work via utilization of deep neural networks based cognate detection methods in the near future.

Contents

List of Figures

Chapter 1

Introduction

The age of humans as species get older and older which has made us realise that human language is not just a recent add-on, but it has evolved with deep evolutionary roots. Some of the questions that come to our minds when we think languages as evolutionary taxa are: Which was the first language that came into existence? How was it evolved over time? How many languages originated form a particular language? *etc*. We try to answer these questions.

Phylogenetics is mainly used in biology for the study of the evolution of a particular taxa over time. It is mainly used to understand the diachronic relationships among individuals or groups of organisms. These relationships are structured through various phylogenetic inference methods like distance based approaches, maximum parsimony, maximum likelihood *etc* which can be computed using computational phylogenetics. The goal of computational phylogenetics is to assemble a graphic visualisation representing a hypotheses about the evolutionary ancestry of various taxa (here Indian Languages). This diagrammatic hypothesis is known as a phylogeny or a phylogenetic tree. We mainly employ the Distance based approach to assemble trees through computing the distance-matrix which gives us the limb lengths of labels (here languages). We survey two majorly used distance based methods: UPGMA(Sokal, 1961) and Neighbor-Joining(Saitou and Nei, 1987).

These methods are given a matrix of size *nxn*, where *n* is the number of labels (here 14). This *nxn* shaped matrix is called the distance matrix which contains the distance or closeness of every label to every other label (here languages). Now, these algorithms give the output of a tree reading formatted string, from which we can use various tools to draw and visualise the phylogenetic trees. Mostly the tools like: PHYLIP (Felsenstein, 1993), CLUSTAL W(Thompson *et al.*, 1994) *etc* use a variant of the nested parentheses format for describing trees, variously referred to as the “Newick” or “New Hampshire” format. This format has yet to be formally described and so is subject to slightly varying interpretations. Details of how we interpret files produced have been detailed.

Now, since we have the tree visualise we start work on improving the tree i.e. we now add context, cognate data, use cross embeddings for creating the distance matrix. We majorly use three similarity methods which are: Normalised Edit Distance (Nerbonne and Heeringa, 1997) and Angular Cosine (Van Dongen and Enright, 2012).

1.1 Lexical Resources

In this sub-section we briefly describe about the datasets we have used to run the experiment. This section also explains about the various techniques we have utilised to handle, clean and bring the data to a format which can be used by the algorithms we use to perform the experiments.

Primarily we use the parallel corpus and WordNet datasets for the experiments. We have collected the datasets for 14 different Indian Languages detailed in the table below:

Hindi(41K)	Marathi(54K)	Bengali(100K)	Assamese(15K)	Kannada(22K)
Malayalam(39K)	Gujarati(103K)	Oriya(35K)	Konkani(32K)	Nepali(200K)
Telugu(36K)	Sanskrit(150K)	Tamil(36K)	Punjabi(36K)	

These Language Data contain data ranging from two Lakh to 15 thousand lines.

1.2 Corpus

One of the main and primary things for running NLP (Natural Language Processing) experiments is to first collect corpus. A corpus refers to a collection of texts. These texts may be formed from a single language (*Monolingal*) or form multiple languages (*multilingual*). Here we use only *monolingual* corpus from which we make *crosslingual or multilingual* corpus. These 14 different corpus were used to train models for cross embedding which are described further where we are improving phylogenetic trees. make the corpora more useful for doing linguistic research, they are often subjected to a process known as annotation. An example of annotating a corpus is part-of-speech tagging (POS-tagging) in which information about each word's part of speech (verb, noun, adjective, *etc.*) is added to the corpus in the form of tags.

Here we are using a different version of corpus called parallel corpus. Parallel corpora or parallel text is a text placed alongside its translations. For example lets consider a sentence in *English*, *"Parallel corpora is cool"*. The parallel text alignment of this sentence in *French* would be *"Corpus parallèles est cool"*. A collection of sentences or a bag of words would create a parallel corpus. Generally, parallel corpus is compiled in the following format:

$$(ID\ ;\ Sentences\ OR\ Words) \tag{1.1}$$

1.3 WordNets

WordNet is a large lexical database of a language. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a different concept. These concepts are separated into different lines. The reason for the use of WordNet was mainly due to the fact that WordNets contain approximately of 44K collocation that go together. Hence, algorithms can be exposed to a large range of words. Another reason for the use of WordNet was becaues, WordNet is a datebase that keeps growing. Which means, the models that are trained with the help of WordNet can improve over time. But on the downside WordNets are a resource hungry entity. Large amount of resources are required to create, maintain and extend the WordNet. Rather than arranging the WordNet in alphabetical order, WordNets are organised in a thesaurus way (in a sense order). Which means every ID across a family of WordNets have the same context. This can be used to the users advantage for caring out experiments.

The most important relation for WordNet is similarity of meaning, since the ability to judge that relation between word forms is a prerequisite for the representation of meanings in a lexical matrix. Also the fact that every ID in the WordNet has the same sense or context is exploited to the maximum possible extend.

The WordNets we are utilising have 5 parts. Each part is delimited with a semi-colon(";"). The format is described below:

$$(ID\ ;\ Words\ ;\ Gloss\ ;\ Definition\ ;\ POS) \tag{1.2}$$

WordNet - ID

In WordNet data. Every line is given a unique ID. Which means every synset or context will have a unique ID. The ID ordering starts form *1* and increments for every synset. For a given WordNet family, an ID across all the WordNets will have the same context. Hence, helping the experiments to differentiate the context.

WordNet - Words

The most important part of the WordNet are the words. The second column in the WordNet is supposed to contain the words which are conveniently separated be a comma (“,”). This column may contain any number of words as long as they belong to a given context.

WordNet - Context

The third and the fourth part of the WordNet (i.e. Gloss, Example) represent context of the entire line.

WordNet - POS Tagging

The fifth and the final coloum of a WordNet data represents the POS (Part-of-speech) tagging. This is not rare—in natural languages a large percentage of word-forms are ambiguous. For example, even ”dogs”, which is usually thought of as just a plural noun, can also be a verb: ”The sailor dogs the hatch”.

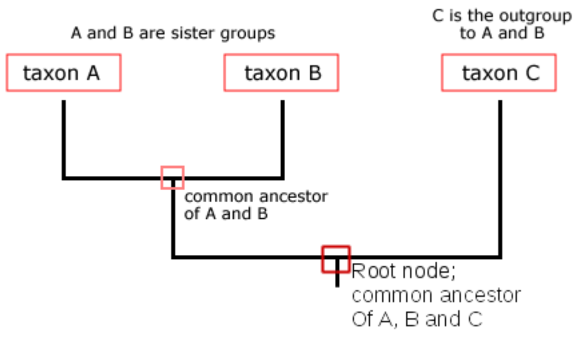
These WordNet data for the above 14 Indian languages were used from Bhattacharyya (2010). Languages like Kashmiri and Manipuri were discarded due to compatibility issues. This data was then normalised, cleaned, structured for future utilisation which has been discussed in detail in the sections below.

Chapter 2

Phylogenetic Trees

2.1 About Phylogenetic Trees

As detailed in the above chapter, phylogenetic tree is a visual representation of the relationship between different taxa. Much like a family tree, a phylogenetic tree has ancestor, root, sisters, successor *etc.* Following is an example of a phylogenetic structure:



A and B are sister groups
C is the outgroup to A and B
taxon A
taxon B
taxon C
common ancestor of A and B
Root node; common ancestor Of A, B and C

Figure 2.1: Root in Tree

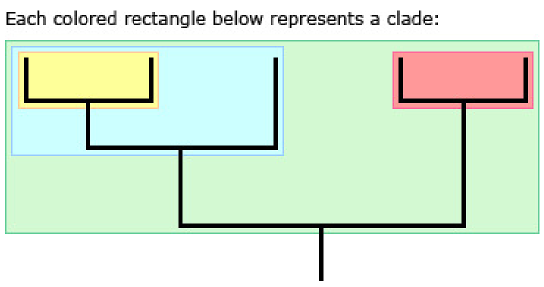


Figure 2.2: Clade in Tree

It is not necessary for every node to have a label. It is to be noted that only leaf nodes have labels. All other internal node are internal that just represent the hypothetical evolutionary ancestors. Leaf nodes are those nodes that don't have children. They are end nodes that represent a entity form the distance matrix inputted. All the other nodes are calculated by the algorithms. It can be said that the number of end nodes will always be equal to the number of labels(in other cases it would be *number of labels + 1*).

2.2 Types of Phylogenetic Trees

There are eleven types of Phylogenetic Trees, from which only three types are majorly being used in practice (Described below). These trees are: Rooted, Unrooted, Bifurating.

Rooted

A rooted phylogenetic tree is a directed tree with a unique node — the root — corresponding to the (usually imputed) most recent common ancestor of all the entities at the leaves of the tree. The root node does not have a parent node, but serves as the parent of all other nodes in the tree. These trees generally have a completely different methods

Unrooted

In a unrooted phylogenetic tree no assumption is made regarding the root node. Unrooted trees can always be generated from rooted ones by simply omitting the root. They do not require the ancestral root to be known or inferred. The unrooted tree is generally preferred as it doesn't need any kind of extra data. These algorithms will output a tree with the data the algorithms are given.An unrooted tree has no pre-determined root and therefore induces no hierarchy. Rooting an unrooted tree involves inserting a new node, which will function as the root node, between two existing nodes.

Binary

A binary, or bifurcating, tree is of course a tree in which a node may have only 0 to 2 subnodes, that is, in an unrooted tree, up to three neighbors. It is sometimes useful to allow more than 2 subnodes (**multifurcation**). A rooted bifurcating tree has exactly two descendants arising from each interior node (that is, it forms a binary tree), and an unrooted bifurcating tree takes the form of an unrooted binary tree, a free tree with exactly three neighbors at each internal node. A labeled tree has specific values assigned to its leaves, while an unlabeled tree, sometimes called a tree shape, defines a topology only.

As described in 2.1, the unnamed parent (red - squared) in the figure are the parents of taxon A, taxon B and taxon A, B, C respectively. These nodes are not leaf nodes and hence are not labeled. Also as detailed in 2.2, each coloured rectangle i.e. representing two taxons or labels and a parent node is called as a **clade**. It is also to be noted that the number of clades and the labels are related by the equation 2.1.

$$(Number\ of\ clades\ =\ Number\ of\ labels\ -\ 1) \tag{2.1}$$

2.3 Construction of Phylogenetic Trees

Phylogenetic trees can be constructed using various computational phylogenetic methods. Each method has its own advantage. Here, we majorly use the distance based approaches which requires a distance matrix that contains the distance between each label. Some of these methods are detailed in the subsections bellow. (Geer, 2002):

2.3.1 Implementations

In this section we briefly describe about the various algorithms that can be used to construct trees. We also specify all the advantages and disadvantage using each algorithms on the data families we are using.

Distance Methods

The distance based methods basically calculate the dis-similarity between two entities. But to calculate the dis-similarity, the similarity has to be calculated first. Here we majorly concentrate on word similarities where algorithms like **Cosine, Edit Distance, Jaccard, Shingling** ***etc*** can be used. For further details of the similarity algorithms used in our work is described in the sections below. Distance is often defined as the fraction of mismatches at aligned positions, with gaps either ignored or counted as mismatches (Mount and Mount, 2001). The main disadvantage of distance-matrix methods is their inability to efficiently use information about local high-variation regions that appear across multiple subtrees (Pruesse *et al.*, 2012). Below are the methods we survyed.

Neighbor-joining

Neighbour-joining Algorithm (Mount and Mount, 2001) was initially designed to analyse DNA and protein data in a tree structure. This algorithm requires distance between each pair of taxa to form the tree. We utilise this algorithm to construct phylogenetic trees. Neighbour-Joining is known to be a complex and resource hungry. The number of calculations that takes place increases exponentially as the number of labels increase.

The main importance of the neighbour-joining algorithm us that it is fast as compared to other methods like **least squares, maximum parsimony and maximum likelihood *etc.*** as detailed in Kuhner and Felsenstein (1994) which makes it to ideal to test and experiment. It is also to be noted that the output tree of neighbour-joining algorithm will always be correct as long as the distance matrix is additive (Bruno *et al.*, 2000). The basic idea is that for neighbour-joining to generate a correct tree, the distance matrix has to be additive. But, on the contrary if a distance matrix isn't additive it wouldn't mean that the generated tree is incorrect. Many other factors like negative branching, internal branching have to be taken into consideration which is out of the scope of this report.

For a distance matrix to be additive. The matrix has to satisfy the **Four point rule.**

$$D(i,j) + D(m,n) \cdot maxD(i,m) + D(j,n), D(j,m) + D(i,n) \tag{2.2}$$

The same equation can be simplified to $a_i \leqslant max(a_j, a_k)$. Which means for every value of the matrix in a(i,j) the *max* value has to be grater of equal to a_i. If this is satisfied then the given matrix *a* is said to be additive. If neighbour joining is not given a additive matrix, there will be a possibility of limb lengths being *negative* which is not possible logically since distance cannot be negative. While performing the experiments, we allow negative branching which in turn allows us to use non-additive matrix. More briefly the four point condition can be described in the following equation.

$$D(i,m) + D(j,n) = D(i,n) + D(j,m) \tag{2.3}$$

The neighbour joining algorithm starts with 2 simple node which is selected on the basis of potential neighbours. It is for a known fact that the nodes having the least distance will be the most likely neighbours. Which is also followed by many other algorithms. But its not always correct. To overcome this, neighbour joining calculates the **Q** matrix form the original distance matrix based on equation 2.4.

$$Q(i,j) = (n-2)d(i,j) - \sum_{k=1}^{n} d(i,k) - \sum_{k=1}^{n} d(j,k) \tag{2.4}$$

Based on the current distance matrix calculate the matrix Q. Now for $Q(i,j)$ a minimum element is selected (where of course $i \leqslant j$). Now, these labels (i,j) will guaranty become new neighbours. These neighbours are connected to a new internal node which will not be one of the labels. Now these three node's (two branches) limb lengths are calculated using equation 2.5.

$$\delta(f,u) = \frac{1}{2}d(f,g) + \frac{1}{2(n-2)}\left[\sum_{k=1}^{n} d(f,k) - \sum_{k=1}^{n} d(g,k)\right] \tag{2.5}$$

$$d(u,k) = \frac{1}{2}[d(f,k) + d(g,k) - d(f,g)] \tag{2.6}$$

Since a new node has been established, the limb-lengths between each label and the new node has to be calculated using equation 2.6. This basically means that a new node has been replaced with the two initial selected nodes. Hence, reducing the Q matrix by two. This process is done until there remains only 2 labels.

Neighbor joining on a set of n taxa requires $n-3$ iterations. At each step one has to build and search a Q matrix. Initially the Q matrix is size $n \times n$, then the next step it is $(n-1) \times (n-1)$, etc. Implementing this in a straightforward way leads to an algorithm with a time complexity of $O(n^3)$. This method has been used in all possible ways in the experiments. The complete **Neighbour Joining** algorithm has been implemented from scratch and open-sourced on Github[1].

UPGMA and WPGMA

UPGMA or Unweighted Pair Group method with arithmetic Mean (Sokal, 1958). UPGMA is the unweight variant version of WPGMA. The broad difference between UPGMA and WPGMA is that every distance has the same weight-age given while calculating the average.

The UPGMA constructs a denegram from the similarity matrix. The basic idea is the same as of the neighbour joining. The nearest two clusters are combined into a higher node removing and centering the initial nodes selected. Rather than being a really complex algorithm UPGMA tries to keep is simple and efficient. UPGMA is also regarded as a unreliable method (which is one of the reason NJ was preferred in the experiments) that results in rooted graphs unlike NJ. The distance between ant two clusturs is given by

[1]https://github.com/YashasviMantha/NLP

equation 2.7.

$$\frac{1}{|\mathcal{A}| \cdot |\mathcal{B}|} \sum_{x \in \mathcal{A}} \sum_{y \in \mathcal{B}} d(x, y) \tag{2.7}$$

A trivial implementation of the algorithm to construct the UPGMA tree has $O(n^3)$ time complexity, and using a heap for each cluster to keep its distances from other cluster reduces its time to $O(n^2 \log n)$. Fionn Murtagh presented some other approaches for special cases, a time algorithm by Day and Edelsbrunner (1984) for k-dimensional data that is optimal $O(n^2)$ for constant k.

Maximum Parsimony and Minimum Evolution

Maximum Parsimony is a method that try to minimize branch lengths by either minimizing distance (minimum evolution) or minimizing the number of mutations (maximum parsimony). This method is generally employed when there is mot even a slight crisp about the data, hence we will not be able to intentionally analyse the tree. These methods minimise the total number of character-state. Which indirectly means that the optimal tree will minimize the amount of homoplasy (*Parallel Evolution*). Or, the shortest possible tree that explains the data is considered best (Farris, 1970).

The minimum-evolution method tree is probably the most unreliable for our data. This methods states that, the tree with smallest sum of branch length will be likly to be the true one (Rzhetsky and Nei, 1993). Equation 2.8 is used to estimate the branch length in Minimum-evolution.

$$\hat{b} = (A^t A)^{-1} A^t \cdot d \tag{2.8}$$

The above approach is based on the law of Maximum Parsimony, which is influenced by Occam's Razor (Gauch Jr *et al.*, 2003). It minimizes the total number of character-state changes is to be preferred. Under the maximum-parsimony criterion. The principle is akin to Occam's razor, which states that—all else being equal—the simplest hypothesis that explains the data should be selected.

Maximum Likelihood

The Maximum Likelihood Estimation method observations Y are given random samples from an unknown population (Myung, 2003). The aim of this *Estimation* is to make inference about the population that is most likely to have the generated sample. Here various probability distribution functions are used (equation 2.9).

$$L(\theta; \mathbf{y}) = f(y_1; \theta) f(y_2; \theta) \dots f(y_T; \theta) = \prod_{t=1}^{T} f(y_t; \theta) \tag{2.9}$$

Here the goal is to maximize the function *f()*, hence increasing the likelihood too. This method is the research standard and is also preferred by many reviewers. But also this method is really expensive.

Many other popular and reliable methods can be used to construct trees like: *Bayesian Approaches*, *MALIGN and POY*, *Sankoff-Morel-Cedergren algorithm etc.* but these are beyond the scope of this report.

Selecting Algorithm

There are a lot of different methods for making a phylogeny. But in short maximum likelihood and Bayesian methods are the two most robust and commonly used methods. Neighbor joining is just a clustering algorithm that clusters haplotypes based on genetic distance and is not often used for publication in recent literature. NJ and UPGMA are clustering algorithms that can make quick trees but are unreliable.The really question was which algorithm had to be selected. NJ algorithm was used in all of the experiments performed. Even though it is known for unreliable and is not a research standard algorithm minor trade-offs were made. Also NJ was more suitable because we did not have any idea about the root of these data family. NJ outputted the exact tree structure we wanted.

Chapter 3

Word Embeddings

Word Embeddings treat words in a dataset as atomic unites. Already many NLP systems have already started using words as atomic units. Word Embeddings can also be called as distributed representations of words. Milkolov's paper on Word Representation (Mikolov *et al.*, 2013) describes completely about word embeddings. Word Embeddings or **Vectors**, are generated using various trained Neural models (Like NNLM, RNNLM, PNN *etc.*) are capable of capturing context of a word in a document, relation with other words.

Word2Vec is one of the most popular method by Mikolov *et al.* (2013), which is a technique to learn word embeddings using shallow neural network. Conventionally to represent a sentence **One-Hot Encoding** (Guo and Berkhahn, 2016) was used. But one hot encoding failed to represent similarity between words. One hot encoding is still used to represent categorical data, but for use to represent the similarity of some words we cant use one-hot encoding. Sentences being represented by one hot encoding are large in size and also inefficiently made. One hot encoding can be used on categorical data where the number of labels are really less in number. When dealing with words, there can be really large vocabulary. Hence creating large dataset. One hot encoding creates a different column for each label. Which also means all similar labels will also be having different columns and no way of representing similarity. For example *Good* and *Great* would have totally different columns even though they have the similar meaning. Therefore we wont able to use this encoding since we require a way of representing the similarity of words.

Our objective is to have words with similar context occupy close spatial positions. Mathematically, the cosine of the angle between such vectors should be close to 1, i.e. angle close to 0.

Cosine Similarity

Cosine Similarity measures the angle between two non-zero vectors. The range of cosine lies between $[0, 1]$. This also means that two vectors of the same orientation will have a similarity of *1*. Also, vectors (Or Words) having opposite (180 degrees) will have a similarity value of 0.

$$= \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{||\mathbf{A}||||\mathbf{B}||} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \tag{3.1}$$

The cosine of two non-zero vectors was derived using the Euclidean Dot Product. Cosine similarity can also be seen as a normalizing document length comparison. Equation 3.1 does **not** give similarity directly. Euqation 3.1 gives the *Angular Distance* form which *Angular Similarity* is calculated.

In computational linguistics word embeddings were discussed in the research area of distributional semantics. It aims to quantify and categorize semantic similarities between linguistic items based on their distributional properties in large samples of language data. The underlying idea that “a word is characterized by the company it keeps” was popularized by Firth (Baroni and Lenci, 2010).

3.1 Word2Vec

Word2Vec is a method to construct such an embedding. It can be obtained using two methods (both involving Neural Networks): Skip Gram and Common Bag Of Words (CBOW)

*CBOW Model

This method takes the context of each word as the input and tries to predict the word corresponding to the context. Consider our example: *“Have a great day”*.

Let the input to the Neural Network be the word. Notice that here we are trying to predict a target word (*day*) using a single context input word great. More specifically, we use the one hot encoding of the input word and measure the output error compared to one hot encoding of the target word (*day*). In the process of predicting the target word, we learn the vector representation of the target word.

The input or the context word is a one hot encoded vector of size V. The hidden layer contains N neurons and the output is again a V length vector with the elements being the softmax values.

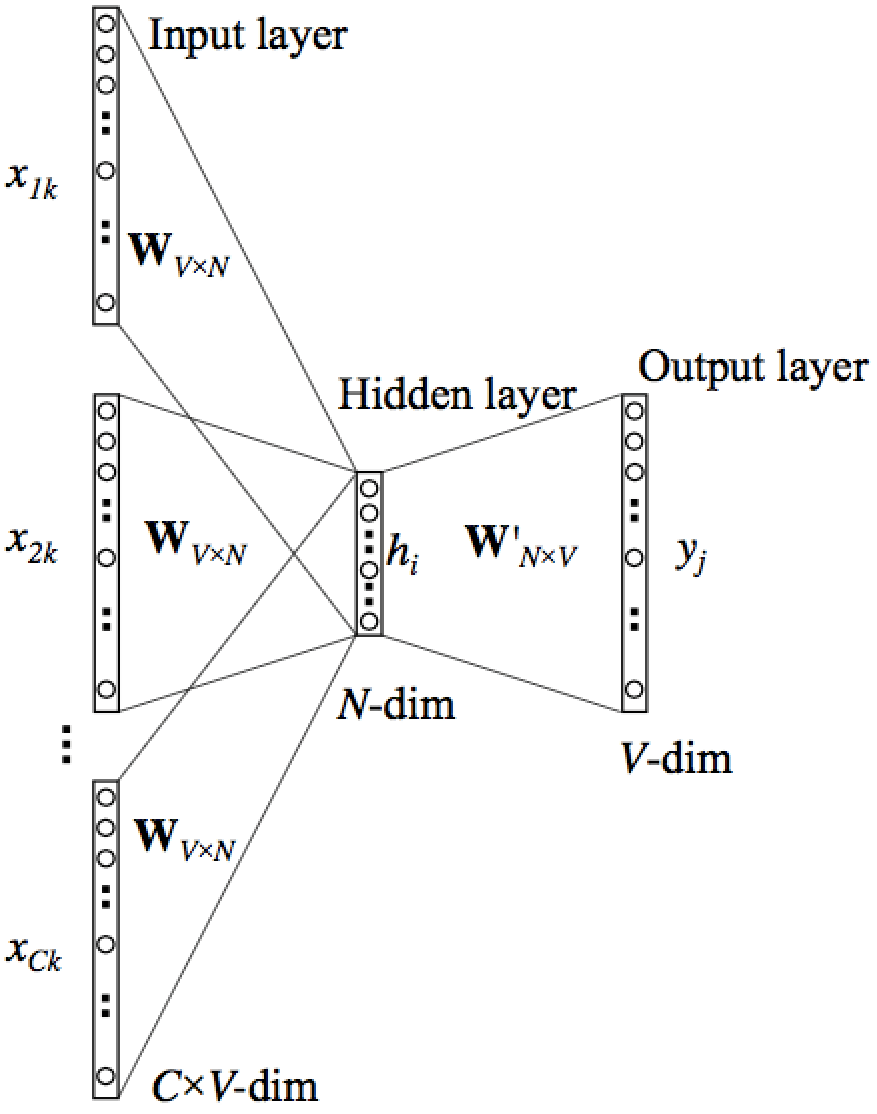


Figure 3.1: CBOW Model

We input the target word into the network. The model outputs C probability distributions. For each context position, we get C probability distributions of V probabilities, one for each word.

3.2 Cross-Lingual Cross Embedding

Our secondary goal is to learn a shared embedding space between words in all languages.

It is to be noted that while neural MT approaches implicitly learn a shared cross-lingual embedding space by optimizing for the MT objective, we will focus on models that explicitly learn cross-lingual word representations. These methods generally do so at a much lower cost than MT and can be considered to be to MT what word embedding models (word2vec, GloVe, etc.) are to language modelling.

Cross-lingual embedding models generally use four different approaches:

- Monolingual mapping: These models initially train monolingual word embeddings on large monolingual corpora. They then learn a linear mapping between monolingual representations in different languages to enable them to map unknown words from the source language to the target language.
- Pseudo-cross-lingual: These approaches create a pseudo-cross-lingual corpus by mixing contexts of different languages. They then train an off-the-shelf word embedding model on the created corpus. The intuition is that the cross-lingual contexts allow the learned representations to capture cross-lingual relations.
- Cross-lingual training: These models train their embeddings on a parallel corpus and optimize a cross-lingual constraint between embeddings of different languages that encourages embeddings of similar words to be close to each other in a shared vector space.
- Joint optimization: These approaches train their models on parallel (and optionally monolingual data). They jointly optimise a combination of monolingual and cross-lingual losses.

3.3 Implementation

In this section we detail about the work we did with word embeddings. As we have already discussed the approaches (Discussed in the subsequent chapter) we use to construct phylogenetic trees. We use crosslingual word embeddings which is the output of various combinations of monolingual word embeddings.

3.3.1 Monolingual Implementation:

Here we use the implemented library for text representation and classification, regrouping the results of Bojanowski *et al.* (2017) and Joulin *et al.* (2016) called **fastText** by *facebook*. fastText is a Python based library which uses **Word2Vec** as baseline and extending and also improving the algorithm at the same time. Instead of feeding individual words into the Neural Network, fastText breaks words into several n-grams (sub-words). The word embedding vector for a word will be the sum of all these n-grams. After training the Neural Network, we will have word embeddings for all the n-grams given the training dataset.

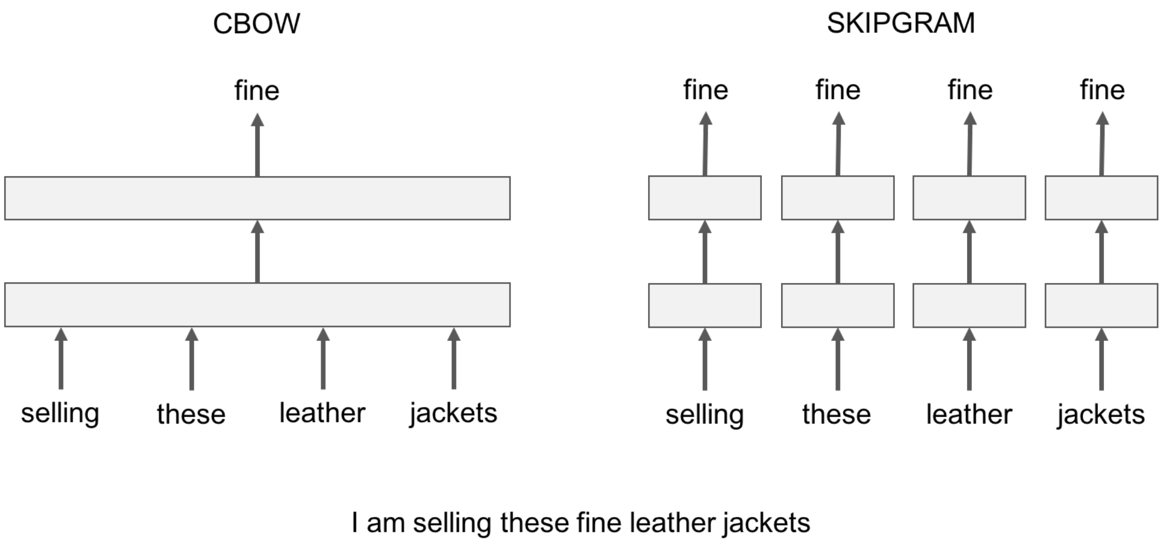


Figure 3.2: CBOW v/s Skip-Gram

fastText also employes a method to represent rare words. Here we use the *Skip - Gram* variant of fastText. The most important parameters of the model are its dimension and the range of size for the subwords. The dimension (dim) controls the size of the vectors, the larger they are the more information they can capture but requires more data to be learned. But, if they are too large, they are harder and slower to train. By default, we use 100 dimensions, but any value in the 100-300 range is as popular. Here we use a dimension of 50 for generating all the vectors for 14 Indian languages.

These are various other Parameters that can be used to tweak the vectors generated but those hyper parameters are out of the scope of this report.

3.3.2 MUSE: Multilingual Unsupervised and Supervised Embeddings

MUSE is a Python library for multilingual word embeddings. MUSE provides state-of-the-art multilingual word embeddings (fastText embeddings aligned in a common space) and large-scale high-quality bilingual dictionaries for training and evaluation. MUSE includes two methods, one supervised that uses a bilingual dictionary or identical character strings, and one unsupervised that does not use any parallel data.

Here we use the supervised (using a train bilingual dictionary, learn a mapping from the source to the target space using Procrustes alignment.) method for multilingual word embeddings. By default, *dico train* will point to our ground-truth dictionaries (downloaded above); when set to *"identical char"* it will use identical character strings between source and target languages to form a vocabulary Conneau *et al.* (2017).

3.3.3 Getting the similarity scores

We get the similarity scores by making a custom *evaluate.py* [1] script which uses 3.1 to calculate the similarity scores between each parallel lines between each pair of languages.

All the files are iterated to calculate the average similarity scores between the language pairs. Hence, giving a grand total of the similarity between the language pairs. These scores are averaged by dividing the grand total by the number of lines in a particular language pair file. These language scores become the similarity scores between that particular pair of languages.

We then calculate the distance scores between each language by subtracting the similarity scores by 1. Hence getting a distance matrix between each language. To improve efficiency of the project, we exclude the same pair language which allows us to exclude 14 same language pair files. The fact that same language pairs will have a similarity scores of 1 ideally and hence a distance score of 0. We directly add these values in the distance matrix.

Below table 3.3.3 has the distance values calculated from Cross embeddings for 14 Different Indian languages.

[1] https://github.com/facebookresearch/MUSE

	as	bn	gu	hi	kn	ko	ml	mr	ne	or	pa	sa	ta	te
as	-	0.4338	0.4573	0.5081	0.3878	0.424	0.4808	0.4697	0.4354	0.3733	0.4542	0.5122	0.5772	0.4918
bn	0.4299	-	0.3973	0.4115	0.4712	0.4667	0.4232	0.3903	0.4135	0.423	0.4175	0.5301	0.4344	0.433
gu	0.4573	0.397	-	0.4166	0.451	0.4234	0.4448	0.3624	0.3902	0.4204	0.3944	0.4995	0.4571	0.4494
hi	0.5078	0.4113	0.4166	-	0.512	0.4953	0.4778	0.4173	0.435	0.4756	0.4163	0.5562	0.4868	0.4772
kn	0.3878	0.4712	0.451	0.512	-	0.3872	0.4771	0.4671	0.4466	0.3819	0.457	0.4939	0.4778	0.4535
ko	0.424	0.4656	0.425	0.4953	0.3872	-	0.5	0.4157	0.4371	0.3951	0.4429	0.4806	0.5273	0.4867
ml	0.4808	0.4229	0.4448	0.4782	0.4781	0.5	-	0.4367	0.4546	0.4692	0.4547	0.5596	0.4196	0.4375
mr	0.4704	0.3903	0.3624	0.4173	0.4671	0.4134	0.4367	-	0.412	0.4311	0.4108	0.5052	0.4464	0.4386
ne	0.4354	0.4135	0.3902	0.4358	0.4466	0.4371	0.4546	0.412	-	0.4103	0.4035	0.5151	0.4911	0.4572
or	0.3733	0.423	0.4204	0.4721	0.3819	0.3951	0.4692	0.4311	0.4108	-	0.4336	0.4819	0.5379	0.4627
pa	0.4545	0.4177	0.3944	0.4163	0.457	0.4429	0.4547	0.4108	0.4035	0.434	-	0.5399	0.4718	0.4576
sa	0.5122	0.5301	0.4995	0.5562	0.4939	0.4806	0.5596	0.5052	0.5151	0.4819	0.5398	-	0.5683	0.5512
ta	0.5772	0.4344	0.4571	0.4868	0.4789	0.5273	0.4204	0.4464	0.4911	0.5379	0.4718	0.5659	-	0.4172
te	0.4918	0.433	0.4486	0.4772	0.4567	0.4867	0.4365	0.4386	0.459	0.4627	0.458	0.5512	0.4172	-

Table 3.1: Distance Matrix Generated, Language Codes Format: alpha-3/ISO 639-2 Code

Chapter 4

Approaches

In this section, we talk about all the technical aspects of the experiment. Here we describe the process of data cleaning and extraction, Similarity algorithms (Missed), initial survey (CNNs and FFNs), Cognates *etc.*

4.1 Data Handling

It is a known fact that, a ML model is as good as it's *data*. Cleaner the data, better is the model trained. Keeping the same thing in mind, considerable amount of man hours were spent in getting the data in a desired format.

Here, we are using 14 different Indian Languages. The data being used was WordNet Bhattacharyya (2010) and parallel corpus (as already described in earlier sections). The parallel corpus wasn't given much attention to. Where as the WordNet data was thoroughly cleaned and formatted since it is one the main components in this project.

The following operations were performed in the process of data cleaning:

4.1.1 Normalisation

Normalization refers to the creation of shifted and scaled versions of statistics, where the intention is that these normalized values allow the comparison of corresponding normalized values for different datasets in a way that eliminates the effects of certain gross influences. Some types of normalization involve only a rescaling, to arrive at values relative to some size variable.

The basic idea of normalising the given data was code compatibility. Since here we are dealing with large families of data, the most effecient way of computing them would be running a script which will run on all the data files without giving issues. We decided to rescale the data to *60,000* lines. The figure *60,000* was selected because most of the data was formatted between 40K to 100K lines. Hence taking the average between all the languages would not only lead to too much data loss, but also cover up for the languages that have relatively less data. 60K was the optimal data quantity that could be used for the further processing.

There were many instances where many language data files were missing some of the **Synsets**. For example, Assamese lacked in synsets 1,5,34 *etc.* This also meant that when processing a particular Synset ID that was available in one language but not the other would result in *Index Errors*.

The data files which contained the data that execeded 60,000 lines were reduced to the first 60,000 synsets. Where as the data files that did not contain 60,000 lines were increased to 60,000 lines. In the place of all the missing synsets, *Dummy Data* was added. The Dummy data selected is listed bellow:

$$ID - Set; NULL; NULL; NULL; NULL \tag{4.1}$$

The result of normalising data gave out 14 files that had no missing synset IDs and contained 60,000 lines each.

4.1.2 Handling Missing Data

Creating data that have no flaws is really expensive. The same goes for Multilingual data too. The data we are utilising have many cases that have missing sub-parts (Like Gloss, POS etc.). It was necessary to remove such cases. Missing sub-parts meant that there would be problems when dealing with comparison of subsequent sub-parts.

It was known that all the sub-parts was separated by a **";"**. Hence, Computationally splitting the lines with **";"** as a delimiter would result in a list of length 5. Iterating over the files at the same time checking each line in a file whether the length of the splitted list is 5 would indirectly tell us the position of missing sub-parts of a language dataset. The result of this operation gave in a uniform data files that did not have any missing sub-parts.

4.1.3 Handling Semi-Colon Misplacement

Data files being dealt with here had various misplacement of the sub-part delimiter ";". This meant that various sub-part delimiter were misplaced too. Various delimiters (like ",") were present in sub-parts which were not supposed to in the first place. This led to further complications which resulted in various mismatches.

Most commonly occurring mismatch was that, some of the words in Synset form the Gloss sub-part was added to the example sub-part of the synset. Hence adding unnecessary delimiters in the wrong sub-part. These errors had to be rectified manually because there was no possible rule that could have generalised the clean up. Fortunately the clean-up was very minimal hence was really easy to handle in manually.

4.1.4 Context Formatting and finalisation

Here we analyse the data and look for potential columns which will not be needed for computation. Such analysis resulted in the removal of *POS* (Part of speech). The tagged data will not be useful for the computation of the similarity between languages. POS may be one of the most important tagged data which is used in NLP. But for this particular use case, POS was discarded. Removal of POS would decrease the size of data to a large extend. Hence improving the efficiency of the overall system.

We also decided to concat the Gloss and Example which results in the representation of the context of the sentence. The context of a synset is really important because, the context of a particular sense could potentially add value to the Model or Data.

4.1.5 Script Conversion

Due to India's varied culture, India is a home for various languages and every language has a different script (most of them). To get the maximum efficiency computationally, we require the data in a common script. For example, Assamese is a language which had to be script converted. We selected **Hindi** as a common language and used the Indic-NLP Library (Anoop Kunchukuttan, 2013) to convert all of the scripts to a common script i.e. **Hindi**.

4.2 Distance Matrix and Tree Plotting

A brief introduction was given about distance matrix in the previous sections. In this section we detail the methods we used in creating the distance matrix and how we draw the phylogenetic tree. As already described, distance matrix was calculated using 2 completely approaches. The distance matrix of word embedding has already been detailed. The final data that resulted in all the data cleaning procedure described above was used in the orthographic similarity approach. Whereas a complete different corpus was used, which contained large number of sentences in each line. From the cross-lingual embeddings Normalised Cosine similarity was used to calculate the similarity between distances. Then distance was calculated form the similarity using the inverse formula.

A variety of approaches can be used to calculate the similarity. All the algorithms have already been implemented to the fullest effectiancy by the Python implementation of the *Java - Library: STRSIM*. In this library, Levenshtein edit distance, LCS distance and their sibblings are computed using the dynamic programming method, which has a cost O(m.n). For Levenshtein distance, the algorithm is sometimes called Wagner-Fischer algorithm (”The string-to-string correction problem”, 1974). The original algorithm uses a matrix of size m x n to store the Levenshtein distance between string prefixes.

4.2.1 Levenshtein Distance

Levenshtein have been extensively been used in the experiments that include the calculation of similarity between orthographic methods. Mathematically, the Levenshtein distance between two strings, a and b (of length a and b respectively). The Levenshtein distance is a string metric for measuring the difference between two sequences.

$$\mathrm{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0 \\ \min \begin{cases} \mathrm{lev}_{a,b}(i-1,j)+1 \\ \mathrm{lev}_{a,b}(i,j-1)+1 \\ \text{otherwise.} \\ \mathrm{lev}_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)} \end{cases} & \end{cases} \tag{4.2}$$

Informally, the Levenshtein distance between two words is the minimum number of single-character edits (i.e. insertions, deletions, or substitutions) required to change one word into the other.

	as	bn	gu	hi	kn	ko	ml	mr	ne	or	pa	sa	ta	te
as	-	0.7885	0.853	0.8252	0.8697	0.8577	0.8952	0.8436	0.8161	0.7987	0.8637	0.8519	0.928	0.8973
bn	0.7885	-	0.7398	0.7314	0.8695	0.7917	0.8448	0.7627	0.7863	0.7351	0.8473	0.7627	0.8937	0.8788
gu	0.853	0.7398	-	0.7197	0.8693	0.779	0.8482	0.75	0.7945	0.7725	0.8452	0.7724	0.9011	0.8738
hi	0.8267	0.7322	0.7207	-	0.8654	0.7666	0.8379	0.7455	0.773	0.7364	0.7993	0.7706	0.8891	0.869
kn	0.8697	0.8695	0.8693	0.8643	-	0.8617	0.8616	0.8599	0.8618	0.8634	0.9006	0.8693	0.8873	0.8493
ko	0.8577	0.7917	0.779	0.7656	0.8616	-	0.8492	0.7493	0.834	0.778	0.8595	0.811	0.8971	0.8728
ml	0.8952	0.8448	0.8482	0.837	0.8616	0.8492	-	0.8475	0.8728	0.837	0.9169	0.8371	0.8124	0.8499
mr	0.8456	0.7641	0.7516	0.7457	0.8616	0.7509	0.849	-	0.8105	0.7753	0.8386	0.7829	0.9018	0.8711
ne	0.8181	0.788	0.7962	0.7723	0.8635	0.8355	0.8744	0.809	-	0.8014	0.8392	0.8178	0.9118	0.8793
or	0.7987	0.7351	0.7725	0.7356	0.8633	0.778	0.837	0.774	0.7997	-	0.8468	0.783	0.8844	0.8716
pa	0.8637	0.8473	0.8452	0.7976	0.9005	0.8595	0.9169	0.8364	0.8372	0.8468	-	0.8863	0.9373	0.9004
sa	0.852	0.7628	0.7725	0.7697	0.8694	0.8111	0.8373	0.7815	0.8156	0.7831	0.8865	-	0.8881	0.8703
ta	0.9334	0.8959	0.9034	0.8903	0.8908	0.8998	0.8147	0.9029	0.9148	0.8868	0.9409	0.8914	-	0.8743
te	0.8973	0.8788	0.8738	0.8676	0.8493	0.8728	0.8499	0.8692	0.8775	0.8716	0.9004	0.8702	0.8716	-

Table 4.1: Distance Matrix Generated through orthographic methods. Language Code Format: alpha-3/ISO 639-2 Code

Table 4.2.1 describes the resultant distance matrix which was calculated using the orthographic approach detailed above section. It needs to be pointed out that same language pair combinations have a value of *zero*. This is in the ideal condition, computationally it has been found out that same language pairs have a distance value which range between (0.13 - 0.29).

4.3 Improving Phylogenetic Trees:

As described in Rama *et al.* (2018) where they implement automatic cognate detection to construct phylogenetic trees. They have compared various methods that have been used to identify cognates. They have already found that trees constructed from cognate detected data is more close to the expert annotated trees (which also means trees have been improved). We propose that inducing cognate data along with the WordNet data can further improve the construction of trees.

We also experiment with context of the WordNet data. We concat the gloss and example before getting it into a common script. Concating the gloss and example of every synset in the WordNet families enables us to get the concrete context of each synsets. Now the same experiments can be performed that were performed on just the words. Hence resulting in the two similarity scores. These similarity were given equal weight age i.e. the average of both the similarities were taken. Trees that have been constructed by the embeddings methods were introduced the context data. The resultant trees did show some improvement, but did not perform as expected.

By performing further investigation, we have found out that the quality of cross lingual word embedding can be further improved. Adding cognate data to the orthographic approach could also be an idea to further improve phylogenetic trees. We can also use other tree construction algorithms like cluster analysis to improve phylogenetic trees.

4.4 Challenges Faced

One of the major challenges that we faced while performing the experimenting was compatibility. There were many tools that wanted some prerequisite dependencies. A very old tool called PHYLIP only runs on DOS based operating systems. PHYLIP was used to plot the trees using the distance matrix generated. There are other alternatives which can be used for plotting trees, but none of them are as reliable as PHYLIP. PHYLIP has various hyper parametes which can be used to achieve the optimal tree. Another tool called *GraphViz* which is a python based library that requires a very old version of C++ distribution (whose support has been discontinued) GraphViz can be utilised to generate newick formated strings that inturn can be used to plot more formatted trees.

Another challenge that we encountered was the lack of bilingual dictionaries for MUSE. Since we are working on only Indian Languages, MUSE requires bilingual dictionaries for generating cross-lingual word embeddings. This problem was solved by retrieving all the lexically equal words which are script converted from the WordNet. There were also times where we could only find hand full number of lexically equal words. We decided to repeat the same words until we met the requirement of MUSE which was around 1.5K words.

Cleaning the data was another big challenge. We had 16 files which included various Indian languages. But unfortunately, Manipuri was in Romanian Format, because of which we were not able to process them. Converting Romanian script to Manipuri script was also not an option because we will have to code a custom convertor which is possible, but would take a lot of time which is totally not worth it. So, we took the decision of removing the whole dataset.

Chapter 5

Results

In this sections we discuss about the constructed trees that were obtained computationally through different approaches. As discussed in the above section that, the quality of word embeddings given for Indian Languages by MUSE framework which is used here happens to be really low. At the current scenario trees plotted using the orthographic similarity approach performs better than that of the cross lingual word embedding approach.

MUSE has been known to perform really well in producing word embeddings for European languages. For getting embeddings for Indian Languages. We had to also input bilingual dictionaries for all the language pairs which is about 182 combinations. We constructed there bilingual dictionaries using lexical equality methods detailed in the above sections. These dictionaries are used to train and evaluate the MUSE model. Poor quality of these dictionaries can also result in the poor quality of the embeddings.

Some of the dictionaries resulted in a files that contained very less number of unique words. These files had to be brought to a quantity that could be inputted into the algorithm. The quality of these dictionaries could be improved to improve the overall algorithm itself. Also using different approaches to generate cross lingual embeddings can also be improve the quality of word embeddings.

We can also observe that inducing context data also improves the trees by a mark. Improving the word embeddings and also inducing the context data would result in a much better tree. Cognate were also detected using the orthographic similarity approach what a threshold of **0.85**. This cognate data can be induced after improving word embeddings

It is to be noted that table 5.1 details the number of cognates that have been identified from a script converted dataset.

Language Code	Number of Cognates Identified	Language Code	Number of Cognates Identified
as	500K	ml	18K
bn	100K	mr	100K
gu	100K	ne	63K
hi	200K	or	100K
kn	50K	pa	40K
ko	73K	sa	60K
te	40K	ta	20K

Table 5.1: Cognate Identifications, Language Format: alpha-3/ISO 639-2 Code

These cognate data will be eventually embedded in the WordNet data and further computation of phylogenetic trees will be done. This would give us a distance matrix from which we can use various plotting algorithms to construct a phylogenetic tree. This data can also be further be experimented with context embeddings. But the first step is to improve the embeddings algorithms for Indian Languages.

Table 5.1 gives the statistics about sum all the cognates found in a particular language with all the particular languages.

Below are the resultant trees illustrated. Figure 5.1 refers to the tree generated from orthographic data. We see that the orthographic data fails to analyse some languages like Assamese. Figure 5.2 illustrates the resultant tree form the cross lingual embeddings data that has been generated. Ideally we hypothesize that tree generated from cross lingual word embeddings would perform better than the baseline orthographic similarity approach.

In figure 5.3 we can see that by introducing the context data in cross lingual word embeddings we improve the tree indirectly. We think improving the the quality of word embeddings will also improve the tree too.

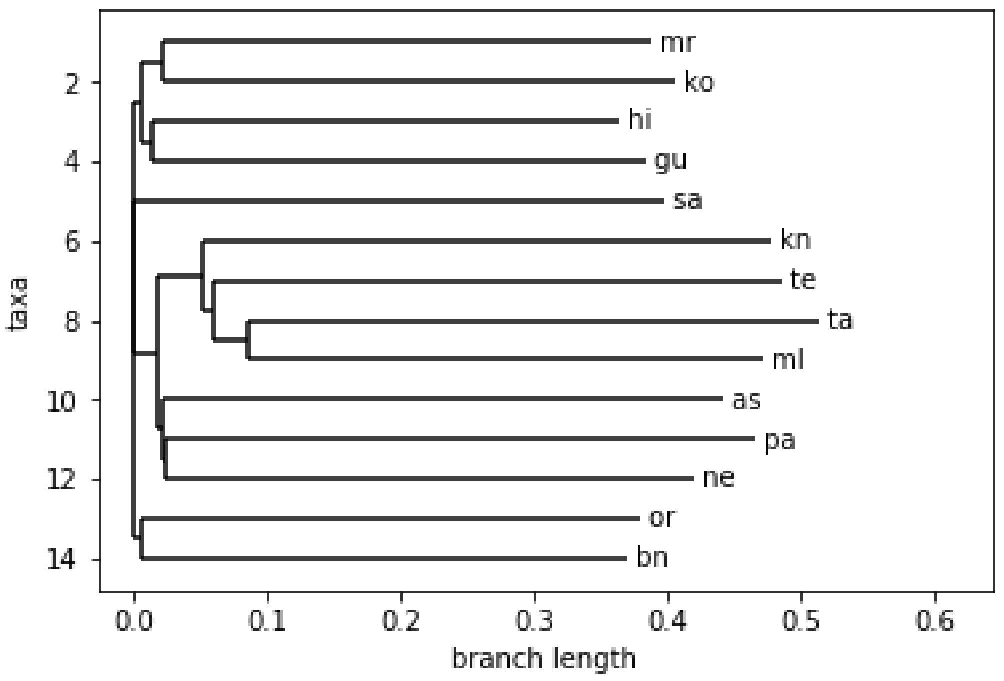


Figure 5.1: Phylogenetic tree constructed from Orthographic data

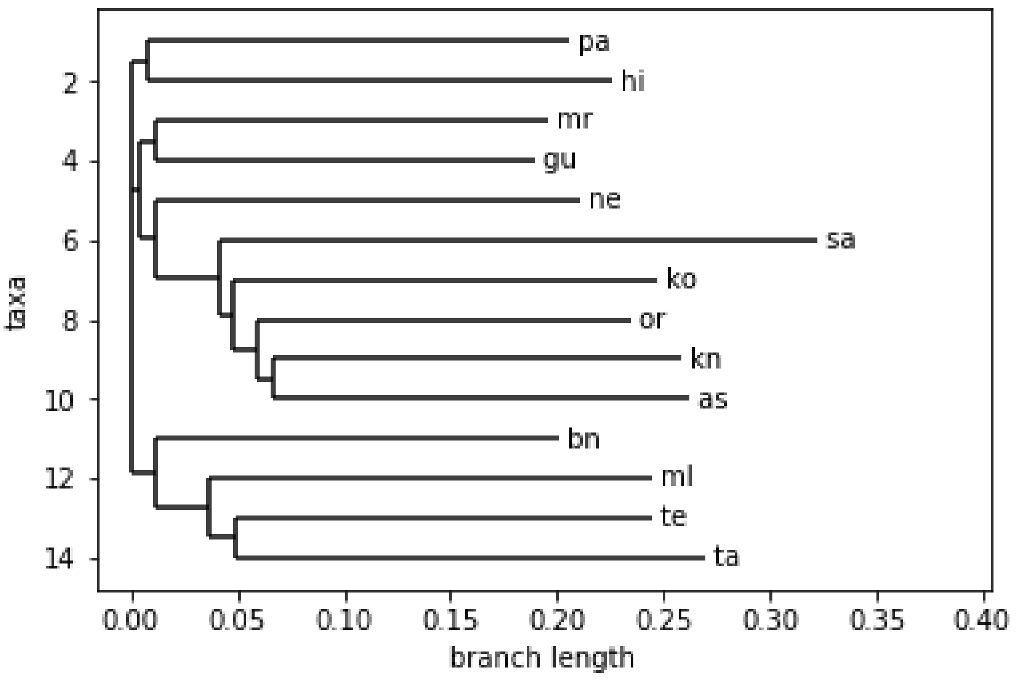


Figure 5.2: Phylogenetic tree constructed from Cross Embeddings data

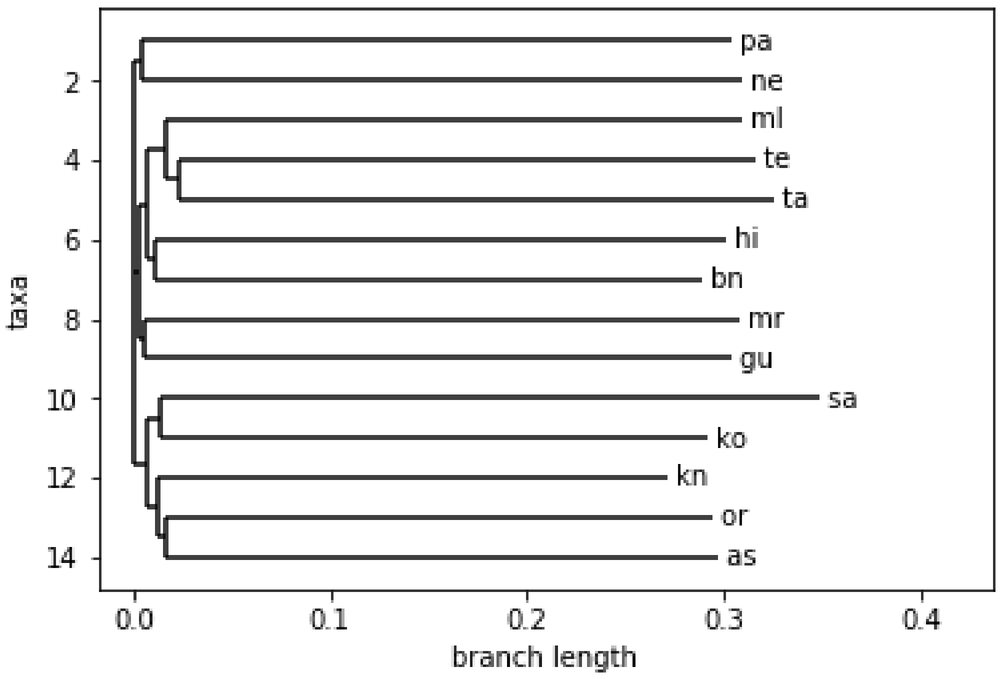


Figure 5.3: Tree constructed from Embeddings with context data

Chapter 6

Conclusion and Future Work

We start of with the data and it's structure that was used to perform the experiments. Then we detail about the types of phylogenetic tree that are present and also detail about the types we use in the period of our work. We detailed various methodologies like Parsimony Method, Statistical methods like Maximum Likelihood, Neighbour Joining etc. and also brief about the algorithms and heuristics used in tree construction, and phylogenetic analysis.

We also detail the about word embeddings and their uses along with the methods for generating them. We then calculate the distance matrix for the analysis of the phylogenetic trees for results. We also survey the potential methods that can be used to improve these resultant trees.

In the future we aim to generate word embeddings with higher quality which would in turn result in improved phylogenetic trees. We also aim to use methodologies to improve phylogenetic trees. Our goal, eventually, is to build a gold phylogenetic tree that illustrates the correct evolution of all the Indian Languages.

References

Anoop Kunchukuttan, 2013, "Indic nlp library," `https://github.com/anoopkunchukuttan/indic_nlp_library`

Baroni, Marco, and Alessandro Lenci, 2010, "Distributional memory: A general framework for corpus-based semantics," *Computational Linguistics* **36**, 673–721

Bhattacharyya, Pushpak, 2010, "Indowordnet," in *In Proc. of LREC-10* (Citeseer)

Bojanowski, Piotr, Edouard Grave, Armand Joulin, and Tomas Mikolov, 2017, "Enriching word vectors with subword information," *Transactions of the Association for Computational Linguistics* **5**, 135–146

Bruno, William J, Nicholas D Socci, and Aaron L Halpern, 2000, "Weighted neighbor joining: a likelihood-based approach to distance-based phylogeny reconstruction," *Molecular biology and evolution* **17**, 189–197

Conneau, Alexis, Guillaume Lample, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou, 2017, "Word translation without parallel data," *arXiv preprint arXiv:1710.04087*

Day, William HE, and Herbert Edelsbrunner, 1984, "Efficient algorithms for agglomerative hierarchical clustering methods," *Journal of classification* **1**, 7–24

Farris, James S, 1970, "Methods for computing wagner trees," *Systematic Biology* **19**, 83–92

Felsenstein, Joseph, 1993, *PHYLIP (phylogeny inference package), version 3.5 c* (Joseph Felsenstein.)

Gauch Jr, Hugh G, Hugh G Gauch, and Hugh G Gauch Jr, 2003, *Scientific method in practice* (Cambridge University Press)

Geer, Alpi Bhagwat Chattopadhyay Gaedeke Lyon Minie Morris Ohles Osterbur Tennant, Messersmith, 2002, "Ncbi advanced workshop for bioinformatics information specialists [online],"

Guo, Cheng, and Felix Berkhahn, 2016, "Entity embeddings of categorical variables," *arXiv preprint arXiv:1604.06737*

Joulin, Armand, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov, 2016, "Bag of tricks for efficient text classification," *arXiv preprint arXiv:1607.01759*

Kuhner, Mary K, and Joseph Felsenstein, 1994, "A simulation comparison of phylogeny algorithms under equal and unequal evolutionary rates.." *Molecular biology and evolution* **11**, 459–468

Mikolov, Tomas, Kai Chen, Greg Corrado, and Jeffrey Dean, 2013, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*

Mount, David W, and David W Mount, 2001, *Bioinformatics: sequence and genome analysis*, Vol. 2 (Cold spring harbor laboratory press New York:)

Myung, In Jae, 2003, "Tutorial on maximum likelihood estimation," *Journal of mathematical Psychology* **47**, 90–100

Nerbonne, John, and Wilbert Heeringa, 1997, "Measuring dialect distance phonetically," in *Computational Phonology: Third Meeting of the ACL Special Interest Group in Computational Phonology*

Pruesse, Elmar, Jörg Peplies, and Frank Oliver Glöckner, 2012, "Sina: accurate high-throughput multiple sequence alignment of ribosomal rna genes," *Bioinformatics* **28**, 1823–1829

Rama, Taraka, Johann-Mattis List, Johannes Wahle, and Gerhard Jäger, 2018, "Are automatic methods for cognate detection good enough for phylogenetic reconstruction in historical linguistics?." *arXiv preprint arXiv:1804.05416*

Rzhetsky, Andre, and Masatoshi Nei, 1993, "Theoretical foundation of the minimum-evolution method of phylogenetic inference.." *Molecular biology and evolution* **10**, 1073–1095

Saitou, Naruya, and Masatoshi Nei, 1987, "The neighbor-joining method: a new method for reconstructing phylogenetic trees.." *Molecular biology and evolution* **4**, 406–425

Sokal, Robert R, 1958, "A statistical method for evaluating systematic relationship," *University of Kansas science bulletin* **28**, 1409–1438

Sokal, Robert R, 1961, "Distance as a measure of taxonomic similarity," *Systematic Zoology* **10**, 70–79

Thompson, Julie D, Desmond G Higgins, and Toby J Gibson, 1994, "Clustal w: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice," *Nucleic acids research* **22**, 4673–4680

Van Dongen, Stijn, and Anton J Enright, 2012, "Metric distances derived from cosine similarity and pearson and spearman correlations," *arXiv preprint arXiv:1208.3145*