SUPPORT VECTOR MACHINE FOR HINDI PARTS-OF-SPEECH TAGGING AND NAMED ENTITY RECOGNITION TASKS

Thesis submitted
in partial fulfillment of the requirements
for the degree of

Bachelor of Technology
in
Computer Science and Engineering

by

Shashi Narayan

under the supervision of
Prof. Sudeshna Sarkar

Department of Computer Science and Engineering
Indian Institute of Technology
Kharagpur-721302, India

Dedicated to my parents and lovely sister,

CERTIFICATE

This is to certify that the thesis entitled "Support Vector Machine for Hindi Parts-of-Speech Tagging and Named Entity Recognition Tasks" submitted by Shashi Narayan for the award of Bachelor of Technology is a bonafide research work carried out by him under my supervision and guidance. To the best of my knowledge, the results have not been submitted to any other university or institute for the award of any degree or diploma. In my opinion, this thesis is of the standard required for the award of the degree of Bachelor of Technology.

May 6th, 2009
Kharagpur

Dr. Sudeshna Sarkar
Professor
Department of Computer Science and Engineering
Indian Institute of Technology
Kharagpur-721302, India

ABSTRACT

In this dissertation, we propose a set of novel kernel functions that can be used with Support Vector Machines for tagging tasks like Parts-of-Speech (POS) tagging and Named Entity Recognition (NER). The proposed kernel function uses a weighted distance of the features. The features used for POS tagging and NER include the current word, the neighboring words and affixes. The crux of the method lies in finding an appropriate distance between the context words. Instead of using a conventional 0/1 distance between words, we use special types of word distance functions for word based features. The proposed word distance functions make finer distance computations of words based on their context similarities. The resulting kernel functions have been applied to the Hindi POS tagging and NER task and the results are quite promising.

ACKNOWLEDGEMENTS

I wish to express my sincere and deepest sense of gratitude and indebtedness to *Prof. Sudeshna Sarkar* for her ingenious help, inspiring suggestions, persistent encouragement and for being patience though all my mistakes. Her wisdom, knowledge and commitment to the highest standards inspired and motivated me. Whatever knowledge I have in NLP, it's because of her. I would like to thank her again for having confidence in me when I doubted myself.

A very special thanks goes to *Sujan Kumar Saha*, for helping me throughout the project. He was always there to meet and talk about my ideas and think through my problems.

I would also like to express my gratitude to all my beloved friends, without whom the journey of *IIT Kharapur* would not have been so delightful. It is to them that I owe my heartiest gratitude.

Finally I express my elite gratitude to my parents for their unbounded care and affection. Their love and encouragement under all odds brought me where I stand today.

Date: May 5th, 2009 Shashi Narayan

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 General

Support Vector Machines (SVM) is primarily a classifier method that performs classification tasks by constructing hyperplanes in a multidimensional space that separates cases of different class labels. SVM supports both regression and classification tasks and can handle multiple continuous and categorical variables. The foundations of SVM have been developed by Vapnik [27] and are gaining popularity due to many attractive features, and promising empirical performance. A schematic example is shown in the Figure 1.1.

Classification problem views input data as two sets of vectors in an n-dimensional

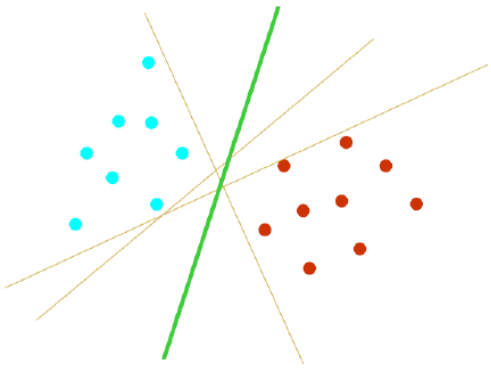

Figure 1.1: Optimal Separating Hyperplane

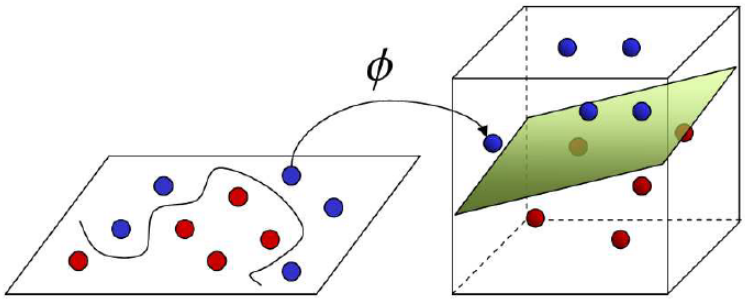

Figure 1.2: Kernel Mapping

space, SVM will construct a separating hyperplane in that space, one which maximizes the margin between the two data sets. This optimization in high-dimensional feature space uses **kernel** tricks to map this non-linear feature space to some other transformed space and then do the optimization with efficient time complexity. The transformation may be non-linear and the transformed space high dimensional; thus though the classifier is a hyperplane in the high-dimensional feature space it may be non-linear in the original input space (Figure 1.2).

Parts-of-speech (POS) tagging and Named entity recognition (NER) are some of the most important and basic parts of Natural Language Processing (NLP). These are prerequisite to information extraction, question answering, machine translation and in most other NLP applications. POS tagging is the process of assigning a part-of-speech like noun, verb, pronoun, preposition, adverb, adjective or other lexical class marker to each word in a sentence, where as NER is a sub-task of information extraction that seeks to locate and classify atomic elements in text into predefined categories such as the names of persons, organizations, locations, etc.

Use of SVM is quite common in different NLP tasks. Both the problem of POS tagging and NER can be converted to combination of multiple classification problem. We use SVM based classifier for these classification problems. To achieve the better accuracy, we have proposed multiple kernels. Two most important kernels proposed are hierarchical words clustering and string kernels. To obtain hierarchical word clustering, we have used the Brown clustering algorithm [19]. We are also comparing

the results of these kernel with other well known systems like Maximum Entropy Markov Model (MEMM) and Conditional Random Field (CRF).

1.2 Motivation

Our aim is to develop promising systems for POS tagging and NER in Indian Languages but we are limited by lack of large annotated resources. Most well known statistical models do not achieve high accuracy when the training set is small. For example, statistical taggers based on Hidden Markov Model (HMM), MEMM, CRF etc., with more than 95% word-level accuracy have been developed for English, German and other European Languages, for which large labeled data is available. But when it comes to Indian languages, performance is very low because of lack of resources. To overcome these problem we are proposing a SVM based system with a novel kernel function.

In many NLP tasks, SVM has proved its importance with respect to other important statistical techniques. We are experimenting with different types of kernels in SVM system. Hierarchical word clustering information based kernel can be used to circumvent the problem of data sparsity using additional feature of similarity. With cluster based kernels, we are also experimenting with binary feature based kernels and string kernels. In both NER and POS tagging tasks, cluster based kernels are implemented with multiple approaches to get better accuracy.

With binary feature based kernels, accuracies are very low. Addition of cluster based kernels and string kernels increases these accuracies to a promising level.

1.3 Organisation of Thesis

There are seven chapters in this dissertation. The organization of dissertation is as follows:

Chapter 2 gives the previous works related to SVM in NLP, specially in POS tagging and NER tasks. It also briefly discusses about these in Hindi POS tagging and NER.

Chapter 3 contains the details of clustering algorithm used for obtaining hierarchical word clustering.

Chapter 4 contains the details of Support Vector Machine and its kernel, an overview. It also presents full details of our purposed kernels.

Chapter 5 presents the details of the data set used for POS tagging and the results of kernels used for our POS tagging experiments.

Chapter 6 presents the details of the data set used for NER and the results of kernels used for our NER experiments.

Chapter 7 contains the final conclusion of the dissertation and the future works that can be done using these results.

Appendix A contains the proof of our kernel's validity.

Appendix B contains details of evaluation matrices used in this dissertation.

Chapter 2

Previous Works and Literature Study

This chapter will guide us through the path which lead us to our current approach used for POS tagging and NER task in Hindi language.

2.1 General Approach to POS tagging and NER Problems

POS tagging and NER play an important role in NLP applications. The history of these problems are very rich. There are mainly two broad approaches which are applied to these problems:

2.1.1 Linguistic Approach

Linguistic approach works on handcrafted rules which are written by skilled linguists. Rule based systems contains mainly lexical grammar, list of important words and set of rules. Although these type of systems has shown huge accuracy [3] [17], but measure problem with these system are that they require huge experience and grammatical knowledge of the particular language and are not transferable to other languages. They also fail within same domain with advancement of the domain.

2.1.2 Machine Learning Approach

Machine learning approach tries with the graphical representation of the problem. Some of the methods that researcher have tried in different language context are HMM [29], decision trees [11], MEMM [2], CRF [28] and SVM [13]. These methods can also be combined to produce hybrid systems. To yield a reasonable performance, these systems make use of a set of suitable features. Wrong selection of these features may add drawbacks in the system in the form of over-fitting. In POS tagging and NER tasks majority of the used features are surrounding words, suffix and prefix information. These methods can successfully incorporate these feature to learn the language model and then these models can be used for classifying new examples. One problem with these techniques is that they make use of a large amount of annotated training data to acquire high level language knowledge. A large amount of annotated training data is not possible with all languages.

Machine learning method reduces its dependency over experience and grammatical information of language. This is the great advantage of this method over linguistic approach.

2.2 SVM and Kernel Application in NLP

SVM is one of the most important machine learning approach to NLP problems. SVM is a classifier which classify by constructing an N-dimensional hyperplane that optimally separates the data into two categories. Main crux lies within its kernel trick, which makes SVM so promising and easy to handle. Kernel maps nonlinear feature space to transformed space, by which it reduces optimization time complexity [12].

Use of SVM is quite common in different NLP tasks like POS tagging [10], NER [5] and Spam recognition. Like the previous learning methods, SVM with surrounding word features, affix features has been used for different applications. SVM does not directly deal with discrete data like strings. In order to build a SVM classifier, these string features are required to be converted into numerical vectors, which is the standard input of SVM. Broadly a few approaches are used for that purpose.

One approach is to use of binary features[25],[13], where a particular feature (e.g., previous word) is converted into several (e.g., total-number of unique words in the lexicon in case of word features) binary features. Binary representation of features can cause huge number of feature to handle, that can also cause over-fitting. The other approach has been to define a family of kernel functions which are directly applicable on string vectors and these are known as "string kernel". String kernels are successfully used in various NLP tasks like text classification [1],[9], bio-medical name recognition [14],[20]. Several researches are carried out to make these kernels efficient in terms of time and space [15],[4].

2.2.1 Advantage of SVM Over Other Methods

Finally the advantage of using SVM over conventional statistical learning algorithms, such as decision tree, hidden markov models, maximum entropy models, can be explained with the following two aspects:

- SVMs have high generalization performance independent of dimension of feature vectors. Conventional algorithms require careful feature selection, which is usually optimized heuristically using feature extraction techniques like Principal Component Analysis (PCA), Linear Discrimination Analysis (LDA) etc, to avoid over-fitting. So, it can more effectively handle the diverse, overlapping and morphologically complex languages.

- SVMs can carry out their learning with all combinations of given features without increasing computational complexity by introducing the kernel function. Conventional algorithms cannot handle these combinations efficiently, thus, we usually select important combinations heuristically with taking the trade-off between accuracy and computational complexity into consideration.

Advantages of SVM over conventional statistical methods like HMM and MEMM are well proved in experiments of [6] [26]. Also when the proper set of feature functions are chosen via validation, then the performances of CRF and SVM are quite close [24] [18].

2.3 Indian Language POS Tagging and NER tasks

India is a multilingual country with great cultural diversities. So from its root, Indian language are very diverse and morphologically rich. Richness of these language have made the NLP task difficult and different from other European languages. The difficulty level increases with the unavailability of the annotated corpus. The algorithm that works excellent with English and most of the European languages does not reach promising level.

2.3.1 Problem with Indian Languages

Some major problem with NLP task like POS tagging, NER etc in Indian languages are as follows:

- Indian languages are resource poor language - annotated corpora, name dictionaries, good morphological analyzers etc. are not yet available in the required measure.
- Web sources for name lists are available in English, but such lists are not available in Bengali and Hindi forcing the use of transliteration.
- Although Indian languages have a very old and rich literary history, technological developments are of recent origin.
- Indian languages are relatively free order.
- Indian languages are highly inflectional language providing one of the richest and most challenging sets of linguistic and statistical features resulting in long and complex word forms.

Although language diversity has made the task difficult, but diversity itself can be incorporated within the system to circumvent the problem of lack of resources. Some recent works have proved this concept's validity. Experiments has been done with adding the information in the form of morphological features of the language, like suffix, prefixes, root words and word lengths in [22]. A reasonably good accuracy POS

tagger for Hindi has been developed using Maximum Entropy Markov Model [7]. The system uses linguistic suffix and POS categories of a word along with other contextual features. Both systems have shown significant improvement over the previous results.

Specially problem of sparsity of resources can be circumvent using semi-supervised model [8], using of both labeled training text and some amount of unlabeled text. Word cluster information derived from unannotated data can be successfully combined [21] with the models to improve the accuracy.

2.3.2 SVM in Hindi POS and NER tasks

Not much of works have been done with SVM in Hindi POS and NER tasks. Works on POS tagging [6] and NER [5] using SVM with features like contextual information of the words, lexicon, named entity recognizer and different word suffixes have got promising results. They have successfully handled unknown word problems. Also results are comparable to HMM, MEMM and CRF.

SVM has already proved its importance over conventional statistical methods like HMM, decison tree, MEMM, CRF etc., in languages like English. Although SVM has not been tried properly in Hindi, there is huge scope in this direction. Diversity of the language can be easily combined (with less complexity and without any overfitting problem) with SVM based system. Data sparsity can be handled borrowing information from unannotated data in form of clusters.

2.4 Our Proposals

The motivation behind the use of SVM framework is that it is more efficient than other conventional statistical methods to deal with the non-independent, diverse and overlapping features of the highly inflictive Hindi.

Our kernels is based upon the surrounding word features, suffix features and prefix features. To reduce the number of binary features, we are introducing new form of string kernel. To tackle the problem of sparsity of resources, we are accumulating the information from hierarchical word clustering into our kernels.

POS Tagging Task: We have experimented with simple binary feature based kernels and cluster based kernels. Cluster Based kernels are again studied with two different type of similarity measures separately; one is using cluster labels and other is using information from tree (obtained from hierarchical word clustering).

NER Task We have experimented with simple binary feature based kernels, string kernels and cluster based kernels. Here cluster Based kernels are only studied with information from hierarchical word clustering tree.

Details of functionality of kernels are further explained in Section 4.3 and Section 4.4.

Chapter 3

Clustering Algorithm

In this chapter, we describe details of clustering algorithm used for obtaining hierarchical word clustering out of raw corpora. Section 3.1 presents the clustering algorithm, we have used and Section 3.2.3 explains the idea of evaluating similarity measure between words, using the clustering information.

3.1 Word Clustering

Our aim of using word clustering is to fight the problem of data sparsity by providing one additional measure of similarity. In natural language systems, words are typically treated categorically, they are simply elements of a set. Given no additional information besides the words themselves, there is no natural and useful measure of similarity between words; laugh and play are no more similar than John. In contrast, things like real vectors and probability distributions have natural measures of similarity.

3.1.1 Similar Words in context of POS tagging and NER

For the purpose of NLP tasks like POS tagging, NER etc, exact similarity is not necessary. We can concentrate on distributional notion of similarity. It means that two words are similar if they appear in similar contexts. Both the words can be exchanged given the context, semantic similarities are not much important. For

example, words "president" and "chairman" are similar under this definition but word "play" and "football" are not similar.

3.1.2 Brown's Clustering Algorithm

To extract features from raw corpus, we have used the bottom-up agglomerative word clustering algorithm of Brown [19] to derive a hierarchical clustering of words. For the explanation of implementation part, we have followed most of the terms and concepts used by Liang in his thesis [16].

Input and Output Format

The input to the algorithm is two text files. The first file contains the list of words $w_1, ..., w_n$ to cluster and the second file (or the directory) contains the raw text, which is used to learn the n-gram model.

The output from the clustering algorithm is a binary tree (Figure 3.4), in which the leaves of the tree are the words of first file. We interpret each internal node as a cluster containing the words in that sub-tree. Thus root node of the tree is a cluster containing all the words. At subsequent level of the binary tree, we can get different set of clusters containing all the words. Size of this set increases as we move to bottom of the tree. At leaf level number of clusters is number of words present in first file, means each word itself is a cluster. Note that the algorithm generates a hard clustering; each word belongs to exactly one cluster regardless of the level considered.

Algorithm

Sequence of Brown clustering can be presented as follows:

1. **Initialization:** Starts with cluster list in which each word in its own cluster, $c_1 \leftarrow w_1$, ...,$c_n \leftarrow w_n$.

2. **Iteration:**

   (a) Stop, if only one cluster left.

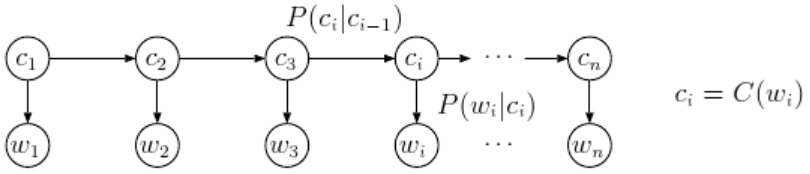

Figure 3.1: Quality of Clustering

(b) Find two clusters that maximizes the quality of the resulting clustering after merging (quality is defined next).

(c) For merging, create a node which will be parent to merged nodes. Delete merged nodes from cluster list and add this new node to cluster list

3. **Termination:** Return the hierarchical word clustering in form of binary tree.

Quality of a Clustering

We view the clustering in the context of a class-based bi-gram language model (Figure 3.1). Given a clustering C that maps each word to a cluster, the class-based language model assigns a probability to the input text $w_1, ..., w_n$, where the maximum-likelihood estimate of the model parameters are used. The quality of the clustering C to be the logarithm of this probability (Equation (3.1)) normalized by the length of the text. Further C is a deterministic mapping (Equation 3.2) and so from the model definition, quality of clustering can be represented in the from of Equation 3.3.

$$Q(C) = \frac{1}{n} log P(w_1, ..., w_n) \tag{3.1}$$

$$= \frac{1}{n} log P(w_1, ..., w_n, C(w_1), C(w_2), ..., C(w_n)) \tag{3.2}$$

$$= \frac{1}{n} log \prod_{i=1}^{n} P(C(w_i)|C(w_{i-1}))P(w_i|C(w_i)) \tag{3.3}$$

We assume that $C(w_0)$ is a special START cluster. Equation (3.1) can be converted to the function of variables the mutual information between adjacent clusters.

Let $n(w)$ be the number of times word w appears in the text and $n(w, w')$ be the number of times the bi-gram (w, w') occurs in the text. Similarly, $n(c) = \sum_{w \in c} n(w)$ to be number of times a word in cluster c appears in the text, and $n(c, c') = \sum_{w \in c, w' \in c'} n(w, w')$. Simple n is the length of the word list in first input file (Section 3.1.2).

$$Q(C) = \frac{1}{n} \sum_{i=1}^{n} logP(C(w_i)|C(w_{i-1}))P(w_i|C(w_i)) \tag{3.4}$$

$$= \sum_{w,w'} \frac{n(w,w')}{n} logP(C(w')|C(w))P(w'|C(w')) \tag{3.5}$$

$$= \sum_{w,w'} \frac{n(w,w')}{n} log \frac{n(C(w), C(w'))}{n(C(w))} \frac{n(w')}{n(C(w'))} \tag{3.6}$$

$$= \sum_{w,w'} \frac{n(w,w')}{n} log \frac{n(C(w), C(w'))n}{n(C(w))n(C(w'))} + \sum_{w,w'} \frac{n(w,w')}{n} log \frac{n(w')}{n} \tag{3.7}$$

$$= \sum_{c,c'} \frac{n(c,c')}{n} log \frac{n(c,c')n}{n(c)n(c')} + \sum_{w'} \frac{n(w')}{n} log \frac{n(w')}{n} \tag{3.8}$$

Now again the terms, $\frac{n(w)}{n}$, $\frac{n(c)}{n}$ and $\frac{n(c,c')}{n}$ can be replaced by the terms $P(w)$, $P(c)$ and $P(c.c')$ respectively. Then Equation (3.8) can be written as follows:

$$Q(C) = \sum_{c,c'} P(c,c') log \frac{P(c,c')}{P(c)P(c')} + \sum_{w} P(w) logP(w) \tag{3.9}$$

$$= I(C) - H \tag{3.10}$$

where $I(C)$ is the average mutual information between adjacent clusters and H is the entropy of the word distribution. Now point to note is that H is independent of clustering, so it is always constant for all types of clustering. So maximization of $Q(C)$ only depends upon $I(C)$ which is sum of mutual information weights between clusters.

Time Complexity of Brown Algorithm

Problem with this algorithm is that we have no practical methods for finding such a partition that maximize the average mutual information $I(C)$. Even given a partition, we have no efficient algorithm to prove that it does maximize $I(C)$. Fortunately we have some greedy algorithms which have shown good results with reduced time complexity.

Let's say we have m words. Initially, we assign each word to a distinct class and compute the average mutual information between adjacent classes. We then merge that pair of classes for which the loss in average mutual information is least. So till termination (Section 3.1.2), for each of $O(m)$ iterations, and for each of possible $O(m^2)$ pairs of clusters to merge, the quality of the resulting clustering is evaluated in again $O(m^2)$. So entire algorithm has time complexity of $O(m^5)$. So the naive implementation of this greedy algorithm is not feasible. We cannot seriously contemplate such a calculation except for very small values of m. A more frugal organization of the computation must take advantage of the redundancy in this straightforward calculation. Fortunately, Brown [19] presents an optimization that reduces the time from $O(m^5)$ to $O(m^3)$. As we shall see in the next section, we can make the computation of the average mutual information remaining after a merge in constant time, instead of $O(m^2)$.

Optimization

Brown [19] has shown this optimized algorithm algebraically which is difficult to follow, so we are presenting the graphical approach, used by Liang [16]. The optimized algorithm basically maintains a table containing the change in clustering quality due to each of the merges.

At any merged state, Clustering can be represented by an undirected graph with clusters as node. Two nodes (clusters) are connected by an edge if both the clusters are adjacent and this edge is given a weight which is calculated using Equation (3.11). Two clusters are adjacent if words present in those clusters are ever adjacent to each

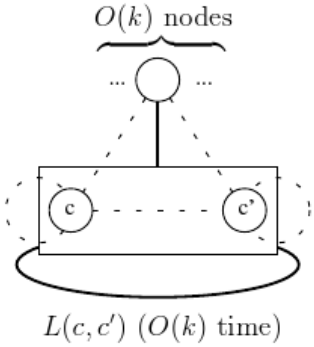

Figure 3.2: Initialization of $L(c, c')$ from scratch

other in either order in the text. A cluster can be adjacent to itself; producing self-loops in the graph. We can easily show that the sum over all edge weights is exactly equal to the mutual information $I(C)$, which is the value we want to maximize. In Equation (3.11) evaluation of $P(c)$, $P(c, c')$, $w(c, c')$ can be done very easily, each time we need.

$$w(c,c') = \begin{cases} P(c,c')log\frac{P(c,c')}{P(c)P(c')} + P(c',c)log\frac{P(c',c)}{P(c')P(c)}, & \text{if } c \neq c' \\ P(c,c)log\frac{P(c,c)}{P(c)P(c)}, & \text{if } c = c'. \end{cases} \tag{3.11}$$

We want to minimize the time complexity of finding out the required pair of cluster to merge. To reach the required purpose, we can maintains a table containing the change in clustering quality due to each of the merges. We will call this table $L(c, c')$. Thus $L(c, c')$ stores the change in total graph weight if c and c' were merged into a new node $c \cup c'$. Algebraically $L(c, c')$ can be represented with the Equation (3.12). As defined in referred equation, If C is the current clustering then let's say C' is the new clustering after merging c and c' to $c \cup c'$, then $C' = C - c, c' + c \cup c'$.

$$L(c,c') = \sum_{d \in C'} w(c \cup c', d) - \sum_{d \in C} (w(c,d) + w(c',d)) \tag{3.12}$$

So main crux lies in optimization of initialising and updating operations of $L(c, c')$.

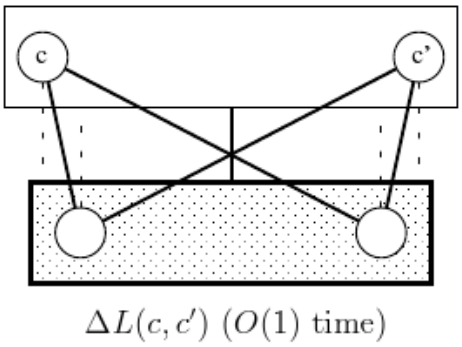

Figure 3.3: Update $L(c, c')$ after merge

Let's initially undirected graph has k nodes. Figure 3.2 shows the edges to be removed (dot lines) and added (solid lines), when c and c' are being merged. Thus merging of c and c' requires operations of $O(k)$. Initialisation of $L(c, c')$ requires this to be done for each pair of (c, c'), i.e. $O(k^2)$ times. So initialisation process of $L(c, c')$ is $O(k^3)$. Now required pair to cluster can be retrieved by searching $L(c, c')$ table in $O(k^2)$. After merging $L(c, c')$ for all the pairs (c, c') in $O(k^2)$. Figure 3.3 shows such operation when both c and c' were not merged. Note that in Figure 3.3, shaded part shows the newly merged nodes. For these set of pairs (c, c') $(O(k^2))$, update in $L(c, c')$ done in constant time. And for the rest of the pairs (c, c') $(O(k))$ where at-least one of c and c' is merged nodes, $L(c, c')$ can be updated from scratch (Figure 3.2) in $O(k)$. Thus again total time complexity of updating $L(c, c')$ table is $O(k^3)$.

Some other methods are also proposed which reduces time complexity further, but unlike the previous optimizations, the new one does not preserve functionality. Method of optimization using a fixed window size [19], does not create complete binary tree.

3.2 Our Experiment with Brown Algorithm

For our purposes, we are using $O(k^3)$ algorithm to implement our system for hierarchical word clustering. Although it is better than naive implementation but still $O(k^3)$ is very expensive. To cluster more than 5000 distinct works is not practical. So

instead of clustering all the words present in training data, we are just clustering most informative words. We are using different measures for collection of most informative words.

Our implementation is different form implementation of Liang [16]:

- In plce of using just one file for input, we are using two different files for input. One file contains the list of word to cluster. Other file contains the raw data, used to train n-gram models. If we use same file for training model also, clustering quality will reduce because of very small size. But we have huge raw corpora, which can be used for better learning model and hence better clustering.
- We are using $O(k^3)$ algorithm to implement our system for hierarchical word clustering. With more time efficient one, we won't get complete binary tree, which is required for our purpose.

3.2.1 Data Set for Clustering

For both the NER and POS tagging task, different approaches have been used for collection of most informative words. We are just clustering these most informative words. Our target is to maximize the coverage and information content. Selection process of these informative words are well explained in Chapter 5 and Chapter 6.

3.2.2 Feature Extraction from Hierarchical Cluster Tree

For better understanding, Figure 3.4 shows the result of our small experiment with Brown Clustering for twelve words. As we have explained in Section 3.1.2, the output of Brown algorithm is a binary tree with leaves as words. We can construct different set of clusters at different level (Table 3.1). So after choosing a particular level (according to the quality at that level), we can annotate each word with a class tag (Table 3.2). Also because of hard clustering and binary tree formation, each word can be assigned unique sequence of bits $(0, 1)$(Table 3.2). Note that words with same class tag have longer prefix matching in their unique sequences.

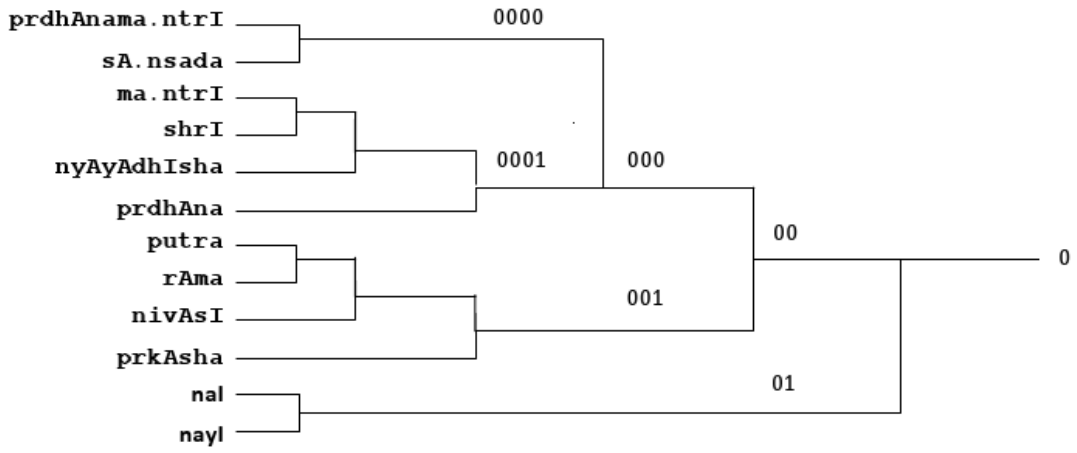

Figure 3.4: Hierarchical Cluster Tree

Figure 3.4, Table 3.1 and Table 3.2 are the results from a very small experiment, just for better understanding. Table 3.3 shows a subset of clustering taken from a complete experiment. As we can interpret, similar words are assigned to the same cluster.

3.2.3 Similarity Measure between Words

Both type of features (class tag and unique binary sequence) can be used as additional measures of similarity.

Class tag is same as POS tag annotation, so in case of MEMM and CRF, class tag can be used as values to the feature set $Class_{-2}$, $Class_{-1}$, $Class_0$, $Class_1$, $Class_2$ etc. To make it applicable with SVM, class tag can be converted to binary features like "Is-Class-I" and "Is-Class-II". It will be more clear in Chapters 5 and 6.

Binary code sequence can be directly used for the similarity measure between two words. We have seen that more similar words will have longer prefix matching in their binary sequences. This property can be used for distance d calculation between two words w_i and w_j.

Let's assume that we have words w_i and w_j with their binary sequences C_{w_i} of length p and C_{w_j} of length q respectively. Also assume that both sequences have a

Level	**No. of Clusters**	**Cluster Organization**
0 (root)	1	(prdhAnama.ntrI, sA.nsada, ma.ntrI, shrI, nyAyAdhIsha, prdhAna, putra, rAma, nivAsI, prkAsha, naI, nayI)
1	2	(prdhAnama.ntrI, sA.nsada, ma.ntrI, shrI, nyAyAdhIsha, prdhAna, putra, rAma, nivAsI, prkAsha), (naI, nayI)
2	3	(prdhAnama.ntrI, sA.nsada, ma.ntrI, shrI, nyAyAdhIsha, prdhAna), (putra, rAma, nivAsI, prkAsha), (naI, nayI)
3	4	(prdhAnama.ntrI, sA.nsada), (ma.ntrI, shrI, nyAyAdhIsha, prdhAna), (putra, rAma, nivAsI, prkAsha), (naI, nayI)
4	6	(prdhAnama.ntrI, sA.nsada), (ma.ntrI, shrI, nyAyAdhIsha), (prdhAna), (putra, rAma, nivAsI), (prkAsha), (naI, nayI)

Table 3.1: Cluster Variation with Level of Tree

r bits long prefix matching. Note that their might be some words for which binary sequences are not available, as we have not clustered all the words. Then distance function can be defined as follows:

$$d(w_i, w_j) = \begin{cases} (p-r)+(q-r), & \text{if } C_{w_i} \text{ and } C_{w_j} \text{ available} \\ 2(D+1) & \text{otherwise, where } D \text{ is depth of tree.} \end{cases} \tag{3.13}$$

Basically distance d is the minimum number of edges required to travel from w_i and w_j (dendrogram distance). If word is not present in tree, distance is twice the maximum distance possible in tree plus one. Now similarity s can be defined as follows:

$$s(w_i, w_j) = \exp^{-d(w_i, w_j)} \tag{3.14}$$

Similarity s will be high if d is small and low if d is high. s will vary from 0 to 1.

Words	**Class Tag (Level-3)**	**Unique Binary Code**
prdhAnama.ntrI	Class-I	00000
sA.nsada	Class-I	00001
ma.ntrI	Class-II	0001000
shrI	Class-II	0001001
nyAyAdhIsha	Class-II	000101
prdhAna	Class-II	00011
putra	Class-III	001000
rAma	Class-III	001001
nivAsI	Class-III	00101
prkAsha	Class-III	0011
naI	Class-IV	010
nayI	Class-IV	011

Table 3.2: Feature Information form Brown Clustering

Class	**Words**
Class I	balki, jise, jinakA, kintu, jinhe.n, jisame.n, natIjatana, kyo.nki, vahI.n
Class II	hogA, vipakshha, ho.n, vikAsakrama, ho.nge, vipakshhI, jAye.nge, ailAna, de.nge, rakshhA, jenevA, jamAI, phiratA, sundaratA, avamUlyana, kelucharaNa, kare.nge, kIjie, mile.nge, hU.N, utsa, rahegA, rahe.ngI, rahegI
Class III	lagAyA, rakhA, dekhA, rilAi.nsa, lAyA, bhejA, paDtA, bulAyA, parikramA, pUchhA, mAnA, samajhA
Class IV	isa, isI, usa, usake, hamAre, vaijJNAniko.n, inake, unake, isake, jisake, apane, kaTAkshha, apanI, apanA

Table 3.3: Cluster Example from Large Experiment

Chapter 4

Support Vector Machines and Kernel

In this chapter, we describe the details of our kernels proposed for POS tagging and NER tasks. But before we go to that, for better catch, we will introduce SVM in brief. Section 4.1 presents concepts of support vectors and kernels and how kernel helps in non-linear space. Section **??** explains the concepts of making SVM applicable with POS tagging and NER tasks. Finally in Section 4.3 and Section 4.4, we present the kernels applied in POS tagging and NER, with complete detailed analysis.

4.1 Brief Introduction to SVM

This sections presents Support Vector Machine (SVM) [27] learning algorithm.

Let's consider the problem of separating the set of training vectors belonging to two separate classes,

$$S = \{(x^{(1)}, y^{(1)}), ..., (x^{(l)}, y^{(l)})\}, x \in \Re^n, y \in \{-1, 1\} \tag{4.1}$$

by a classifier which is defined as

$$h_{w,b}(x) = g(w^T x + b), \text{ } w \text{ and } b \text{ are parameters of } h. \tag{4.2}$$

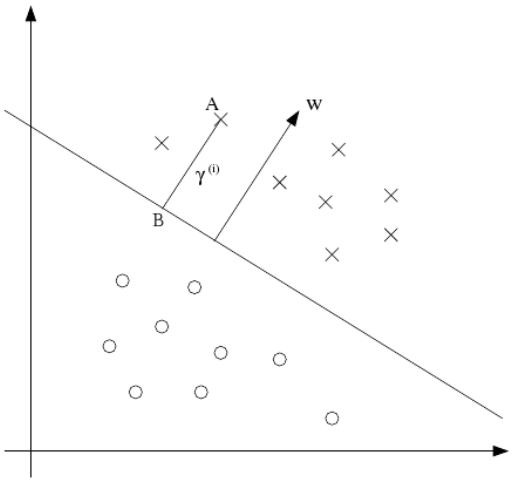

Figure 4.1: Functional Margin and Confident Examples

Here $g(z) = 1$ if $z \geq 0$ and $g(z) = 0$ otherwise.

4.1.1 Functional and Geometric Margins

For a given training example $(x^{(i)}, y^{(i)})$, the functional margin of (w, b) with respect to the training example is defined as

$$\hat{\gamma}^{(i)} = y^{(i)}(w^T x^{(i)} + b) \tag{4.3}$$

Let's visualize the functional margin concept with the help of Figure 4.1. Training examples are shown as $\times$ and o marks. $\times$ shows the positive training examples and o shows the negative training examples. The line shows the decision boundary or our classifier defined in Equation 4.2. Now for all the examples which are marked as $\times$, values of $(w^T x^{(i)} + b)$ are greater than 0 and for all the examples which are marked as o, values of $(w^T x^{(i)} + b)$ are less than 0. Examples are more confident, more it is distant from boundary line. Now let's consider functional margin (Equation 4.3).

For all correctly classified examples, $\hat{\gamma}^{(i)}$ is greater than 0 and shown by the perpendicular distance of example from classifier line. Hence, a large functional margin represents a confident and a correct prediction. One problem with this measure is it's functional dependency over w and b. We can make the functional margin arbitrarily

large without really changing anything meaningful. This lead us to define geometric margin. Geometric margin is the normalised version of functional margin.

For a given training example $(x^{(i)}, y^{(i)})$, the geometric margin of (w, b) with respect to the training example is defined as

$$\gamma^{(i)} = y^{(i)} \left(\left(\frac{w}{\| w \|} \right)^T x^{(i)} + \frac{b}{\| w \|} \right) \tag{4.4}$$

And the geometric margin of (w, b) with respect to S to be the smallest of the geometric margins on the individual training examples:

$$\gamma = min_{i=1,...,l} \left(\gamma^{(i)} \right) \tag{4.5}$$

4.1.2 The Optimal Margin Classifier

Given a training set S, target of SVM is to find a decision boundary that maximizes the geometric margin of S, since this would reflect a very confident set of predictions on the training set S and a good fit to the training data. Specifically, this will result in a classifier that separates the positive and the negative training examples with a gap equivalent to geometric margin. So the aim of SVM can be presented in the following optimization problem:

$$\begin{aligned} max_{\gamma,w,b} \quad & \gamma, \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq \gamma, i = 1, ..., l \\ & \| w \| = 1. \end{aligned} \tag{4.6}$$

After scaling $\hat{\gamma}$ to 1 and normalizing with respect to w, we can redefine our optimization problem to:

$$\begin{aligned} min_{\gamma,w,b} \quad & \frac{1}{2} \| w \|^2, \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq 1, i = 1, ..., l \end{aligned} \tag{4.7}$$

Current form of the problem is an optimization problem with a convex quadratic

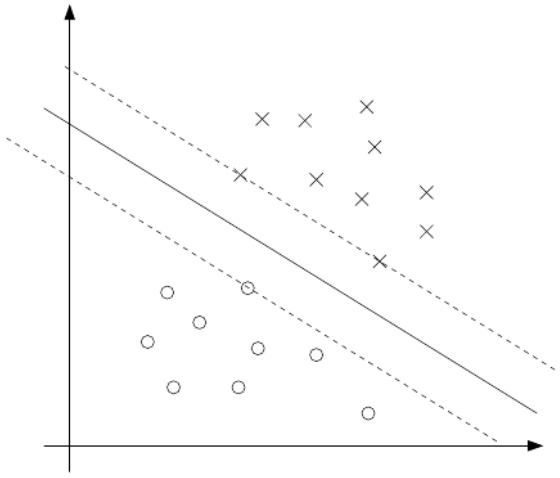

Figure 4.2: The Optimal Margin Classifier

objective and only linear constraints. Its solution gives us the optimal margin classifier (solution of parameters of Equation (4.2)).

Study of Equation 4.7 with lagrange duality (lagrange multipliers α and β) and Karush-Kuhn-Tucker (KKT) conditions in dual form, guide us to use kernel function to solve this optimization problem and to work very efficiently in high dimensional space. Final solution of Equation 4.7 in dual form can be used to write the final classifier:

$$w^T x + b = \sum_{i=1}^{l} \alpha_i y^{(i)} \langle x^{(i)}, x \rangle + b \tag{4.8}$$

One point to note that from one of KKT dual complementarity condition, we will have $\alpha_i > 0$, only for the training examples that have functional margin exactly equal to one ($\hat{\gamma} = 1$). This fact is really helpful in learning model and predicting new examples, because we need not to calculate all inner product ($\langle x^{(i)}, x \rangle$) in training examples. We just need to calculate for those $x^{(i)}$'s for which ($\hat{\gamma} = 1$) i.e. $\alpha_i > 0$. Figure 4.2 shows such point at both sides. These points are called **Support Vectors**.

Thus we have shown that the solution of optimization problem can be written only in terms of inner products between input feature vectors. We can exploit this property to apply the kernels to our classification problem. The resulting algorithm,

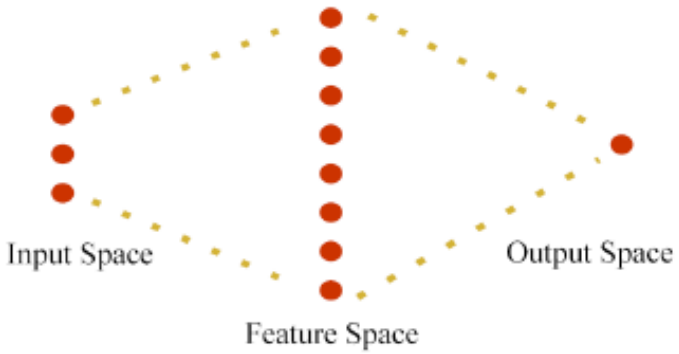

Figure 4.3: Mapping the Input Space into a High Dimensional Feature Space

support vector machines, will be able to efficiently learn in very high dimensional spaces.

4.1.3 Kernel

Kernel is a trick to create non-linear classifiers which maps non-linear space to some transformed high dimensional space where classifier can be represented as hyperplane (Figure 4.3). The transformation may be non-linear and the transformed space high dimensional.

Let's assume x and z are two attribute in non-linear space and ϕ is a function which maps attribute form non-linear space to feature space. Now rather than applying SVM using the original input attributes x, we may instead choose to learn using some features $\phi(x)$. This can done simply by replacing x in Equation (4.8) by $\phi(x)$. Since the algorithm can be written entirely in terms of the inner products $\langle x, z\rangle$, this means that we would replace all those inner products with $\langle \phi(x), \phi(z)\rangle$.

Now from the definition of kernel, an inner product in feature space has an equivalent kernel in input space, provided certain conditions hold, like K is a symmetric positive definite function (Appendix A).

$$K(x, z) = \langle \phi(x), \phi(z)\rangle \tag{4.9}$$

Using Equation (4.9), the resulting algorithm is formally similar, except that

every dot product is replaced by a non-linear kernel function $K(x, z)$. This allows the algorithm to fit the maximum-margin hyperplane in the transformed feature space using feature mapping function ϕ.

We can easily compute $K(x, z)$ by finding $\phi(x)$ and $\phi(z)$ and taking their inner product. By using efficient calculation of $K(x, z)$, we can even get reed of these calculation.

$$\langle \phi(x), \phi(z) \rangle = K(\langle x, z \rangle) \tag{4.10}$$

Equation (4.10) says that inner product in feature space is directly the function of inner product in input space. In those cases, SVM can learn in high dimensional feature space given by ϕ without finding $\phi(x)$ and $\phi(z)$ explicitly.

Some of the important kernel used in ML are polynomial kernel, Gaussian kernel, string kernel etc.

4.1.4 SVM Implementation: SVM^{light}

For our experiments, we have used SVM^{light}[1]. SVM^{light} is an implementation of Support Vector Machines (SVMs) in C. Main features of this system is that we can integrate our own custom kernel very easily. Because of steepest feasible descent and caching of kernel evaluations, SVM^{light} is real fast. It can easily handle thousands of support vectors and several hundred-thousands of training examples.

At first, system learns from training file using customized kernel function and creates a **model** file. Model file basically learn all the support vectors. This model file is used for classifying new examples. After testing is complete, it produces a prediction file which contains the confidence value of each example for that classification.

[1] SVM^{light}: Support Vector Machine by Thorsten Joachims, Cornell University, http://svmlight.joachims.org

4.2 Binarizing the Classification Problem

Part-of-speech tagging a word or name entity recognition of a word in context is a multiple-class classification problem. Since SVM are binary classifiers, a binarization of the problem needed to be performed before we apply them. We have experimented with three different types of binarization scheme (dictionary scheme). Let's assume that there are m classes.

4.2.1 No Dictionary Used

This is the most common approach to these problems. We have applied a simple one-per-class binarization, i.e., a SVM is trained for every class (m times) in order to distinguish between examples of this class and all the rest. So the learning process creates a total of m model files. When assigning a class to a word, the most confident class according to the predictions of all m binary SVMs (model file) is selected.

4.2.2 Dictionary Used for Testing

In this approach, we again use the same approach for training as previous. SVMs are trained for every class (m times) in order to distinguish between examples of this class and all the rest. But when assigning a class to a word, we don't check for confidence value with the predictions of all m binary SVMs. A dictionary is extracted from the training corpus with all possible classes for each word. We use this dictionary to reduce the domain of search for confidence value from m to set of classes possible for this word. But if word is unknown then search domain is set of all classes (m).

4.2.3 Dictionary Used for Training and Testing

This approach is highly expensive. It also creates huge number of model files. Main idea behind this method is that if we are considering a word w with possible class set $[c_1, c_2, ..., c_n]$, and we are finding confidence value for class c_1 then it should consider only the rest of the classes of this set as negative classes, not all. In this way, we avoid the generation of excessive (and irrelevant) negative examples.

In this approach a dictionary is extracted from the training corpus with all possible classes for each word. We form a super-set of possible sets of classes. For each of set of classes of super-set, we create model files considering above approach. For assigning a class to a word, we again use dictionary to find possible set of classes, and so possible set of model files and the confidence value.

Note that unless it is mentioned, we are using the first approach.

4.3 Proposed Kernels for POS Tagging

This section presents all the kernels proposed for POS tagging task. We have used two different kernels based on feature information retrieved from hierarchical clustering (Table 3.2). The purpose of the use of word clustering is to circumvent the problem of data sparsity by incorporating one additional measure of similarity. Kernel based on class tag information presented in Section 4.3.1 and kernel based on code sequences presented in Section 4.3.2. Corresponding results can be found in Chapter 5.

4.3.1 Binary Feature Based Kernel

Feature Space

This kernel incorporate class tag (Table 3.2, column 1 and 2) information with the existing feature set like surrounding words and previous POS tags (Figure 4.4). The overall feature space (F) can be represented as combination of different (say, m) **feature groups**. For example in the current kernel, with window of length 3, we have features like word feature group ("previous-word", "current-word", "next-word"), POS tag feature group ("previous-tag") and class feature group ("previous-class", "current-class", "next-class").

Kernel Computation

These features can not be used directly in SVM like we use in MEMM or CRF. As name clarifies, this kernel is binary feature based kernel. All features are coded to binary valued features. For example a particular feature (e.g., previous-word) is

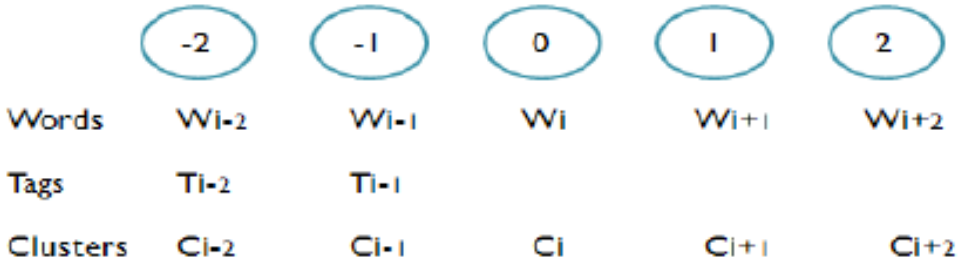

Figure 4.4: Feature Space with Window Size 5

converted into several (e.g., total number of unique words in the lexicon in case of word features) binary features. For a word, all those features will have value 1 which is present in the context. For the rest, value will be 0.

For example, in a context "Ram-NN-C_1 drinks-VB-C_2 water-NN-C_3", "Ram", "drinks" and "water" are three words with NN, VB and NN POS tags and C_1, C_2 and C_3 class tags respectively. Then for word "drinks", binary valued features with value 1 are "previous-word-Ram", "current-word-drinks", "next-word-water", "previous-tag-NN", "previous-class-C_1", "current-class-C_2", "next-class-C_3". Here window length is 3

Present representation does not need any specific kernel evaluation. Simple kernel of SVM^{light}, dot product in feature space, suffices.

4.3.2 Hierarchical Cluster Based Kernel

Feature Space

This kernel incorporate cluster information in form of binary bit sequences (Table 3.2, column 1 and 3). As we have clustered only the words using the clustering approach, the hierarchical cluster based kernel deals with the word feature group only. The words occurring in the word window are represented by the bit sequences obtained from the cluster. Thus the element of word feature group "previous-word", "current-word", "next-word" etc. represent different bit sequences.

Kernel Computation

The similarity between two same feature of two different words is computed using Equation (3.14). Finally the individual similarity of features are combined in a weighted fashion to calculate similarity between words.

Current word is the most important feature among other features of the word feature group, and this is given a weight of 1. The weights of the other positions are taken to vary inversely with the square of the distance to the current word. Thus +1 and -1 positions combined receive a weight of 1/2, and +2 and -2 positions get a total weight of 1/4 (required if window of length 5).

4.4 Proposed Kernels for NER

This section presents all the kernels proposed for NER tagging task. We have used four different kernels. Kernel based on binary feature space is presented in Section 4.4.1, string kernel is presented in Section 4.4.2, hierarchical cluster based kernel is presented in Section 4.4.3 and composite kernels is presented in Section 4.4.4. Corresponding results can be found in Chapter 6.

4.4.1 Binary Feature Based Kernel

Feature Space

The feature space (F) of this kernel consists of word feature group (F_{word}), suffix feature group (F_{suf}) and prefix feature group (F_{pre}). We are not using any cluster feature group. Results of this kernel is taken as baseline results.

(F_{word}) is set of features ("previous-word", "current-word", "next-word") (if window of length 3), (F_{suf}) is set of l features if we consider suffixes upto length l and similarly (F_{pre}) is set of l features if we consider prefixes upto length l.

Kernel Computation

Process is similar to the one explained in Section 4.3.1. All features of F are coded to binary valued features. Features of (F_{word}) can be coded in same manner explained in Section 4.3.1. Features of (F_{suf}) and (F_{pre}) themselves can be used for binary valued features.

For example, in a context "Ram drinks water", "Ram", "drinks" and "water" are three words. Now for the word "drinks", binary valued features with value 1 are "previous-word-Ram", "current-word-drinks", "next-word-water", "s", "ks", "nks", "d", "dr" and "dri". Here window length is 3 and affix length considered is also 3.

Again binary representation does not need any specific kernel evaluation. Simple kernel of SVM^{light}, dot product in feature space, suffices.

4.4.2 String Kernel

Feature Space

Feature space of string kernel is same to the feature space used for binary feature based kernel (Section 4.4.1). The feature space (F) of this kernel again consists of 3 different feature groups (F_{word}), (F_{suf}) and (F_{pre}).

Kernel Computation

For string kernel, we don't use binary representation of features. The individual features are transformed into a $c+1$ dimensional vector where c is the number of named entity (NE) classes. These vectors are based on class information based statistics derived from the training data. For each feature group, a sub-kernel is used. Finally the sub-kernels are combined in a weighted fashion to obtain the final string kernel using:

$$K_{str}(x, y) = \sum_{j=1}^{m} \lambda_j K_j(x, y) \tag{4.11}$$

where K_j denotes the j-th sub-kernel corresponding to the j-th feature group

and λ_j refers to its corresponding weight which is chosen as the relative importance of the feature group in the feature space. Since kernel function set is closed under normalization, polynomial expansion and linear combination of two kernels also give rise to a proper kernel [23].

Feature Vector Representation: The sub-kernels use a numerical vector representation of the individual features. The vectors are of dimension $c+1$ corresponding to c NE classes and one for the not-name class. These vectors use class information based statistics derived from the training corpus. The vector representation is discussed here in detail in the context of word features (F_{word}).

Word features are used in form of a context window of length $p+q+1$, containing p previous words and q next words. Different position specific vectors are defined corresponding to the $p+q+1$ different positions. Now for each word (w) in the corpus we define a class specific weight, $Wt_C(w)$, which is assigned as the corresponding component of the $(c+1)$-dimensional vector. For a particular position, $Wt_C(w)$ is defined as,

$$Wt_C(w) = \frac{\text{Occurrence of } w \text{ in position } pos \text{ of a NE of class } C}{\text{Total occurrence of } w \text{ in corpus}} \tag{4.12}$$

where *pos* denotes a particular position of the $p+q+1$ window. Feature vectors are defined for other feature groups similarly.

Computing the Sub-kernels: Now the sub-kernels for the feature groups are computed. First, for each feature in a feature group we assign a relative importance. For example, for word feature group current word is given a weight of 1. The weights of the other positions are taken to vary inversely with the square of the distance to the current word. Thus $+1$ and -1 positions combinedly receive a weight of $1/2$, and $+2$ and -2 positions get a total weight of $1/4$. For the other feature groups similar distribution is assigned. For affix features, we currently consider only the affixes of the current word. Here we assign length specific weight distribution; higher length (like, $l = 4, 5$) affixes share more weightage than the smaller affixes ($l = 1, 2$).

Now the sub-kernels between the instances are computed. For two particular

feature values, X_k and Y_k corresponding to k-th feature group, we compute,

$$K_{val(k)} = \begin{cases} 1, & if\ X_k = Y_k; \\ \langle X_k, Y_k \rangle, & otherwise. \end{cases} \tag{4.13}$$

Here $\langle X_k, Y_k \rangle$ is the inner product between the vectors corresponding to X_k and Y_k. Once all the $K_{val(k)}$ values for a particular feature group are obtained, these are combined using the corresponding weight distribution discussed above to get the final sub-kernel $K_j(x, y)$ for the feature group. Now we combine all sub-kernels using Equation 4.11 to get the final kernel, string kernel $K_{str}(x, y)$.

In Table 4.1, we have explained the computation of the string kernel with an example. We are evaluating the string kernel for words "kvAlAlampura" and "bilAsapura". In the example, we have considered word features (window 3) and suffix feature (of length 5, 4 and 3). Correcsponding word features and suffix features are shown in the table. Vectors rows show the feature vector representation of corresponding feature. $K_{val(k)}$ is the kernel value for each instance of a feature group (Equation 4.13). Now we combine these $K_{val(k)}$'s of a feature group using weight distribution. This gives kernel value K_{group} for each feature group. These are combined using Equation 4.11 to get the final kernel K_{str} (λ_j's are taken 1).

	Word features {-1 0 +1}	**Suffix features {5 4 3}**
Instance-1 Vector-1	me kvAlAlampura shahara [.03 .04 .04 .89] [0 1 0 0] [0 .43 0 .57]	mpura pura ura [0 1 0 0] [0 .9 .09 .01] [.02 .08 .01 .89]
Instance-2 Vector-2	me bilAsapura jile [.03 .04 .04 .89] [0 .8 .2 0] [0 .63 0 .38]	apura pura ura [0 .91 .09 0] [0 .9 .09 .01] [.02 .08 .01 .89]
$K_{val(k)}$	1 0.80 0.49	0.91 1 1
K_{group}	1.17	1.16
K_{str}	2.33	

Table 4.1: String kernel computation

4.4.3 Hierarchical Cluster Based Kernel

Feature Space

Feature space of this kernel is same as Section 4.3.2. The words occurring in the word window are represented by the bit sequences obtained from the cluster. Thus the element of word feature group F_{word} represent different bit sequences forming new feature group $F_{cluster}$. Note that, in this case we have different cluster information at different location of the word window. For example, if window length is 5 then there are 5 hierarchical clustering (one for each location). So same word at different location of window can have different bit sequence. This method is used to cover most informative words (Section 6.2.3).

Kernel Computation

The similarity between two same feature of two different words is computed using Equation (3.14). Finally the individual similarity of features are combined in a weighted fashion to calculate similarity between words (explained in Section 4.3.2).

4.4.4 Composite Kernel

Feature Space

Feature space of this kernel is union of feature spaces of string kernel (Section 4.4.2) and hierarchical cluster based kernel (Section 4.4.3). Thus feature space F of this kernel consists of F_{word}, F_{suf}, F_{pre} and $F_{cluster}$.

Kernel Computation

Process of kernel computation is as follows:

1. Calculate the sub-kernel (F_{word}), (F_{suf}) and (F_{pre}), as explained in Section 4.4.2).
2. Calculate the sub-kernel $F_{cluster}$, as explained in Section 4.4.3.
3. Combine all these 4 sub-kernel using Equation (4.11).

Chapter 5

Experiments on Parts of Speech Tagging

This chapter deals with the details of the data set used for POS tagging and the results of the different kernels (Section 4.3) applied.

5.1 What are POS Tags?

27 POS Classes	JJ, NN, PSP, QW, VF, SYM, PRP, VNN, RP, JJP, CC, RB, NNPC, NNP, VNF, QC,VAUX, NNC, NEG, NST, DEM, QF, QO, INJ, INTF, ECH, UNK

Table 5.1: Pasts-of-Speech Tag Set

Part-of-speech (POS) of a word explains not what the word is, but how the word is used. In fact, the same word can be a noun in one sentence and a verb or adjective in the next. For our POS tagging task, we have used 27 POS classes (Table 5.1).

5.2 Details of Data Used

5.2.1 Unannotated Corpora: Dainik Jagran ($\mathcal{D}_{unannotated}$)

This data is used in Brown clustering algorithm (Section 3.1.2). The file is used as a raw file (second input file) to learn uni-gram and bi-gram model for the words present in first input file.

- Total number of words: 20 millions
- Total number of distinct words: $230,296$

5.2.2 Annotated Corpora ($\mathcal{D}_{annotated}$)

This data is used in POS tagging experiments. Both training and testing files are randomly generated from this corpora. Each word of this file is annotated with one of the 27 tags (Table 5.1).

- Total number of sentences: $2,176$
- Total number of words: $50,199$
- Total number of distinct words: $7,891$
- Number of POS classes used for annotation: 27

5.2.3 Development of Word List To Cluster

To use $\mathcal{D}_{annotated}$ in POS tagging experiment, we need to cluster all of its distinct 7891 words. But because of expensive time complexity of Brown algorithm, we could not cluster all 7891 words.

So finally, in place of clustering all 7891 words, we collect N the most informative words, and use this word list to cluster (first input file to Brown Algorithm). We are using **frequency** as an information measure to our experiment. Note that frequency measure also maximise the coverage. In this dissertation, we have experimented with two types of clustering information. First one is using top 1000 most frequent words

and second one is using top 2000 most frequent words. Rest of the words of $\mathcal{D}_{annotated}$ which are not covered by clustered word, are labeled as "unknown-class". We are also comparing the results in both of the cases and we prove that larger the clustered words better is the result.

Top 1000 **Most Informative words** ($\mathcal{L}_{1000}$)

- Minimum frequency of words: 6
- Number of words covered of $\mathcal{D}_{annotated}$: $38,027$
- Percentage coverage: 75.75%

Top 2000 **Most Informative words** ($\mathcal{L}_{2000}$)

- Minimum frequency of words: 3
- Number of words covered of $\mathcal{D}_{annotated}$: $42,333$
- Percentage coverage: 84.33%

5.2.4 Development of Training and Test File

Train and test data, both are extracted from annotated corpora $\mathcal{D}_{annotated}$ randomly. 600 sentences are selected randomly out of $2,671$ for test data preparation and rest of the $2,071$ sentences taken for training data preparation. Statistics of training and test data are as follows:

Training Data: 2071 Sentences ($\mathcal{D}_{train}$)

- Total number of words: $38,857$
- Number of words with "unknown-class" tag:
  - $9,439$ when clustered with $\mathcal{L}_{1000}$
  - $6,086$ when clustered with $\mathcal{L}_{2000}$

Test Data: 600 Sentences ($\mathcal{D}_{test}$)

- Total number of words: $11,342$
- Number of words with "unknown-class" tag:
  - $2,733$ when clustered with $\mathcal{L}_{1000}$
  - $1,780$ when clustered with $\mathcal{L}_{2000}$

Unless it is specifically mentioned in the experiment, these two data set for training and testing are used.

5.3 SVM: Binary Feature Based Kernel

This section shows the results of kernel explained in Section 4.3.1. Table 5.2 shows the results of experiment done with window length 3 and Table 5.3 shows the results of experiment done with window length 5.

Classifier	**Known Words**	**Unknown Words**	**Overall**
Feature: Surrounding words and Tags (Baseline)			
SVM	90.35	54.02	87.17
MaxEnt	78.39	33.16	73.81
CRF	87.87	57.47	84.79
Feature: Surrounding words, cluster labels and Tags			
SVM	91.34	56.19	87.79
MaxEnt	88.74	56.44	85.48
CRF	88.27	59.36	85.34

Table 5.2: POS Tagging: Binary feature based kernel [Window-3]

From the result, we see that addition of cluster label enhance the system's performance. Either it is MEMM or CRF or SVM, system's performance increases by a significant margin. In case of unknown word this jump is huge because although word is new but contextual information may be similar. Also in all case results of SVM is superior with others.

Classifier	**Known Words**	**Unknown Words**	**Overall**
Feature: Surrounding words and Tags (Baseline)			
SVM	91.58	55.07	87.90
MaxEnt	73.95	35.48	70.06
CRF	86.04	57.21	83.13
Feature: Surrounding words, cluster labels and Tags			
SVM	91.71	56.62	87.90
MaxEnt	84.20	55.06	81.26
CRF	86.43	60.13	83.77

Table 5.3: POS Tagging: Binary feature based kernel [Window-5]

5.4 SVM: Hierarchical Cluster Based Kernel

This section shows the results of kernel explained in Section 4.3.2. Note that Table 5.4, 5.5 and 5.6 shows the results of a binary classification, "NN" group (["NN", "NNPC", "NNP", "NNC"]) vs Rest, task. It does not show the results of POS tagging. This is used to faster proof of our method's validity. Window length is taken 5, unless mentioned.

Table 5.4 compares the results of two classifier based on cluster information derived from either of $\mathcal{L}_{1000}$ or $\mathcal{L}_{2000}$. Results proves our statement that if we would have all the distinct words of $\mathcal{D}_{train}$ clustered, system would perform much better.

List Used for Cluster Information Derivation	**Precision**	**Recall**	**Accuracy**
$\mathcal{L}_{1000}$	87.56	86.11	92.77
$\mathcal{L}_{2000}$	89.77	89.45	94.26

Table 5.4: "NN" group vs others: System's performance over the use of clustered information

Table 5.5 shows the performance of our system (SVM) with varying size of training data. We fix the test data ($\mathcal{D}_{test}$) and gradually increase the training data form smaller size to $\mathcal{D}_{train}$. As per expectation, performance of system improves with the increase in the size of training data. The quantity of increment decreases as we reach to

saturation level.

Training Data Size	$\mathcal{L}_{1000}$			$\mathcal{L}_{2000}$		
	Precision	**Recall**	**Accuracy**	**Precision**	**Recall**	**Accuracy**
11,490	86.55	79.99	91.02	88.09	81.33	91.79
18,684	88.42	80.82	91.77	89.52	84.35	92.94
28,337	87.33	85.44	92.54	89.78	87.86	93.87
33,686	88.10	84.45	92.54	90.07	88.15	94.03
38,857	87.56	86.11	92.77	89.77	89.45	94.26

Table 5.5: "NN" group vs others: System's performance over varying size of training data

Table 5.6 shows the comparative performance of our system (SVM based) with respect to MEMM and CRF for a binary classification experiment ("NN" group vs Rest). Our system based upon hierarchical cluster based kernel performs better than other systems like MEMM and CRF (surrounding words and tags).

Classifier	**Precision**	**Recall**	**Accuracy**
SVM (cluster)	89.77	89.45	94.26
MaxEnt (Word and Tag)	89.21	78.23	91.36
CRF (Word and Tag)	89.91	82.38	92.57

Table 5.6: "NN" group vs others: System's performance compared to MEMM and CRF

Table 5.7 shows the results of classification experiment (one Vs other) for every class of POS.

Table 5.8 shows the final result of POS tagging experiment using different types of dictionary scheme (Section 4.2).

	$\mathcal{D}_{train}$		$\mathcal{D}_{test}$		Performance		
Class	(+)	(-)	(+)	(-)	**Precision**	**Recall**	**Accuracy**
CC	1627	37230	484	10858	92.64	93.60	99.41
DEM	449	38408	147	11195	86.55	70.07	99.47
ECH	1	38856	0	11342	nan	nan	100
INJ	26	38831	9	11333	100	44.44	99,96
INTF	56	38801	17	11325	77.88	41,18	99.89
JJ	1609	37248	451	10891	89.47	41.46	97.48
JJP	468	38389	126	11216	20.00	2.38	98.81
NEG	423	38434	127	11215	97.67	99.21	99.96
NN	8456	30401	2470	8872	87.08	81.86	93.41
NNC	567	38290	160	11182	99.36	33.12	99.04
NNP	1307	37550	400	10942	91.83	47.75	98.01
NNPC	367	38490	108	11234	88.10	34.26	99.33
NST	406	38451	100	11242	91.92	91.00	99.85
PRP	2888	35969	811	10531	92.22	93.59	98.98
PSP	5895	32962	1701	9641	97.58	97.00	99.19
QC	701	38156	241	11101	95.52	79.67	99.49
QF	415	38442	115	11227	86.78	91.30	99.77
QO	159	38698	39	11303	96.55	71.79	99.89
QW	162	38695	48	11294	97.87	95.83	99.97
RB	320	38537	86	11256	71.88	26.74	99.37
RP	937	37920	270	11072	98.18	80.00	99.49
SYM	3857	35000	1131	10211	100	99.2	99.92
VAUX	3034	35823	904	10438	89.43	86.06	98.08
VF	3454	35403	1003	10339	81.19	63.71	95.49
VNF	681	38176	225	11117	94.12	35.56	98.68
VNN	592	38265	167	11175	89.29	59.88	99.30

Table 5.7: Classification Results: One vs Rest

Dictionary Scheme	**Accuracy**
No-Dictionary (Section 4.2.1)	85.83
Dictioanry for Testing (Section 4.2.2)	87.15
Dictionary for Training and Testing (Section 4.2.3)	87.38

Table 5.8: POS Tagging: Final Result

Chapter 6

Experiments on Named Entity Recognition

This chapter deals with the details of the data set used for NER and the results of the different kernels (Section 4.4) applied.

6.1 What are NER Classes?

13 NE Classes	L [LU, LB, LC, LE], P [PU, PB, PC, PE], O [OU, OB, OC, OE] and NN

Table 6.1: NER Classes [U: Unique, B: Begin, C: Continue, E: End]

Named entities (NE) are predefined categories such as the names of persons, organizations, locations, expressions of times, quantities, monetary values, percentages, etc. For our NER task, we have 3 major NE person (P), location (L) and organisation (O). One extra class is for "Not-Noun" (NN). Finally these four are distributed over 13 different classes (Table 6.1).

For our experiment, we have combined U (unique) and B (begin) to B. And C (continue) and E (end) to C. So finally we have 7 NE classes, LB, LC, PB, PC,

OB, *OC* and *NN*.

6.2 Details of Data Used

6.2.1 Unannotated Corpora: Dainik Jagran ($\mathcal{D}_{unannotated}$)

This data is used in Brown clustering algorithm (Section 3.1.2). The file is used as a raw file (second input file) to learn uni-gram and bi-gram model for the words present in first input file.

- Total number of words: 12.6 millions
- Total number of distinct words: $130,675$

6.2.2 Annotated Corpora: Training ($\mathcal{D}_{train}$) and Test ($\mathcal{D}_{test}$) Data

For NER task, Annotated data are directly in the form of training and test data. Each word of these files is annotated with one of the 13 NE classes (Table 6.1). Statistics of $\mathcal{D}_{train}$ and $\mathcal{D}_{test}$ are as follows:
Training Data ($\mathcal{D}_{train}$)

- Total number of sentences: $9,982$
- Total number of words: $204,948$
- Total number of distinct words: $17,192$

Test Data ($\mathcal{D}_{test}$)

- Total number of sentences: 481
- Total number of words: $10,008$
- Total number of distinct words: $2,741$

6.2.3 Development of Word List To Cluster

To use hierarchical based kernel, we need to cluster all of $\mathcal{D}_{train}$'s distinct $17,192$ words. But because of expensive time complexity of Brown algorithm, we could not cluster all $17,192$ words. So finally, in place of clustering all $17,192$ words, we collect N the most informative words, and use this word list to cluster (first input file to Brown Algorithm).

In NER task, **frequency**, as an information measure will not suffice, because we have many top frequent words from "NN" class. So we have used different way to tackle this problem. In place of having single hierarchical clustering information for all the words, we have l different hierarchical clustering information for all the words, where l is window length. So we have hierarchical clustering for each location of window. Now a word can have different clustering information, if appears at different places.

This method has been used because it provides better way to cover informative words (depending on its location in window and its effect on current word). A word is informative in its location if it is frequent and current word of the window has very less probability to be "NN". Using this information we can form word list for each location.

In this dissertation, we have experimented with two types of clustering information. First one is using top 1000 most informative words at each window location and second one is using top 2000 most informative words at each window location. Rest of the words which are not covered by clustering at a window location , are labeled as "unknown-class". We are also comparing the results in both of the cases and we prove that larger the clustered words, better is the result.

6.3 SVM: Binary Feature Based Kernel (Baseline)

This section shows the results of kernel explained in Section 4.4.1. Table 6.2 presents the results from binary kernel with surrounding word features. Table 6.3 presents the results from binary kernel with surrounding word features and affix features. These

are the baselines for our proposed kernels.

	Named Entity Classes			
Performance	Person (P)	Loc (L)	Org (O)	Total
F-measure	64.259	68.569	60.042	64.290

Table 6.2: NER: Binary feature (current and surrounding words) based kernel [Window-3]

	Named Entity Classes			
Performance	Person (P)	Loc (L)	Org (O)	Total
Precision	79.505	68.569	96.429	88.230
Recall	66.125	74.583	62.914	68.158
F-measure	73.099	81.469	76.147	76.905

Table 6.3: NER: Binary feature (Words and Affix) based kernel [Window-5]

6.4 SVM: String Kernel

This section shows the results of kernel explained in Section 4.4.2. In Table 6.4, only the feature vector representation of surrounding words are considered, where as in Table 6.5, the feature vector representation of suffix and prefixes are also considered. Addition of suffix features increases the accuracy by a significant margin. Note that the maximum accuracy increases to 79.30 after tuning the system.

	Named Entity Classes			
Performance	Person (P)	Loc (L)	Org (O)	Total
Precision	82.391	90.0	85.156	85.294
Recall	59.079	72.083	70.199	65.395
F-measure	68.814	80.051	76.957	74.031

Table 6.4: NER: String Kernel (surrounding words) [Window-5]

	Named Entity Classes			
Performance	Person (P)	Loc (L)	Org (O)	Total
Precision	82.038	80.242	79.870	81.032
Recall	71.816	79.583	74.834	74.868
F-measure	76.587	79.911	77.270	77.828

Table 6.5: NER: String Kernel (surrounding words and affixes) [Window-5]

6.5 SVM: Hierarchical Cluster Based Kernel

This section shows the results of kernel explained in Section 4.4.3. Kernel only use word features in the form of clustering information (binary sequences). In Table 6.6, for each position of the word window, the 1000 most informative word has been selected. Table 6.7 similarly considers 2000 most informative word at each position of the word window.

	Named Entity Classes			
Performance	Person (P)	Loc (L)	Org (O)	Total
Precision	79.310	89.308	86.792	83.267
Recall	44.444	55.417	23.841	43.816
F-measure	56.966	68.394	37.407	57.418

Table 6.6: NER: Hierarchical cluster based kernel [1000 most informative words]

	Named Entity Classes			
Performance	Person (P)	Loc (L)	Org (O)	Total
Precision	81.882	87.778	87.037	84.696
Recall	47.967	62.500	47.020	52.368
F-measure	60.496	73.013	61.056	64.720

Table 6.7: NER: Hierarchical cluster based kernel [2000 most informative words]

6.6 SVM: Composite Kernel

This section shows the results of kernel explained in Section 4.4.4. Kernel combines word features, affix features and cluster features. Note that for this experiment, we have considered top 2000 most informative words at each position of window.

	Named Entity Classes			
Performance	Person (P)	Loc (L)	Org (O)	Total
Precision	83.380	82.403	74.342	81.233
Recall	71.816	77.500	72.848	73.816
F-measure	77.167	79.877	73.587	77.347

Table 6.8: NER: Composite kernel [Surrounding words and cluster feature]

	Named Entity Classes			
Performance	Person (P)	Loc (L)	Org (O)	Total
Precision	83.836	79.600	80.392	81.771
Recall	74.526	80.000	74.834	76.316
F-measure	79.967	79.799	77.514	79.931

Table 6.9: NER: Composite kernel [Surrounding words, affixes and cluster feature]

6.7 Result Analysis

Table 6.10 summarizes the experimental results. It also compares the corresponding best result to the results of MEMM and CRF. Using binary representation we have achieved a f-value of 64.88 when only word features are used. Addition of suffix and prefix features increases the f-value to 76.91. Using the string kernel, we have achieved a f-value of 74.03 when only word features are used. When these are the only features used, a SVM with our kernel function significantly outperforms other well-known approaches. With the addition of suffix and prefix features we were able to increase the f-value to 79.20. This accuracy is obtained when the ratio among the weightage of the word, suffix and prefix features is chosen as 4:1:1. When the ratio is taken as

Classifier	**Precision**	**Recall**	**F-val**
Feature: current and surrounding words			
SVM: Binary Feature	83.87	52.90	64.88
SVM: String Kernel	85.29	65.40	74.03
SVM: Cluster Kernel	84.70	52.37	64.72
SVM: Composite Kernel	81.23	73.82	77.35
MaxEnt	66.24	52.57	58.62
CRF	75.11	54.61	63.24
Feature: Word, Suffix, Prefix			
SVM: Binary Feature	88.23	68.16	76.91
SVM: String Kernel	87.44	72.37	79.20
SVM: Composite Kernel	88.16	73.11	79.93
MaxEnt	87.11	67.63	76.15
CRF	90.41	73.16	80.87

Table 6.10: NER: Final Results - The accuracy of different classifiers

2:1:1 the accuracy reduces to 77.84, which proves that the selection of suitable weight of the feature groups (i.e., the λ values in Equation 4.11) is important.

Because of the high time complexity of Brown clustering and unavailability of time, in our experiments we have clustered only 2000 words for each position in word window. As only 2000 words are clustered (whereas total number of distinct words in the training corpus is 17K), for majority of the training samples the bit sequences are not available. In spite of the poor coverage of clustering, using the hierarchical word clustering based kernel we have achieved a f-value of 64.72. If all the 17K words were clustered then the kernel is expected to perform well. But the results are yet to be verified. To show the impact of cluster size on the classifier accuracy we run experiment using 1000 words clustering. These 1000 words are the most informative words based on the used selection technique. Using these clusters the f-value of the system is reduced to 57.41.

The composite kernel based classifier achieves a f-value of 77.35 using only word features and 79.93 using word and affix features. From the results it is observed that the proposed composite kernel based classifier outperforms the binary feature representation based classifier. For only word features the improvement is very high

(64.88 to 77.35); and when affix features are added then also significant performance improvement (76.91 to 79.93) is achieved. Now we compare the performance of the SVM classifier using the proposed kernel with MaxEnt and CRF[1] classifiers that use same feature sets. When only the word features are used, the proposed kernel performs much better than both the MaxEnt and CRF classifiers. With affix features also the proposed kernel is better than MaxEnt and competitive with CRF.

[1]The tools used are, http://maxent.sourceforge.net/ and http://crfpp.sourceforge.net/ respectively

Chapter 7

Conclusion and Future Works

7.1 Conclusion

In this dissertation, we have proposed novel family of kernel functions. Word features and affix based features have been used. Each feature is given a different weight. The kernel function computes the weighted distance between a pair of instances. In NER task, the distance between the word features is computed by making use of some word distance function which are obtained by processing the entire corpus.

We have experimented with two different types of word distance functions. The first one defines a feature vector (NER) corresponding to each word based on its occurrence in the proximity of named entities. The second one makes use of a clustering algorithm to define the distances. The results look quite promising. We will also like to compare this approach to other string based kernels used in literature.

7.2 Future Work

Algorithms proposed in this dissertation are at their very rudimentary stage and there are many possible improvements that can be implemented. Some of the future works that can extend the dimension of our works are:

- First of all, we have seen that larger the cluster information better is the result.

So we would like to try our systems with all words clustered and verify our proposals at large scale.

- Currently we have just used word features and affix features. We can combine other features like word length, to tune our system.
- We would like to check our methods on broad data like Bio-medical NER system, so that we can compare the results with broader dimension.

Appendix A

Valid Kernels

Let's say $\mathcal{K}$ be a kernel corresponding to some feature mapping ϕ. Now, consider some finite set of m points $[x^{(1)}, ..., x^{(m)}]$, and let a square, m-by-m matrix $\mathcal{K}$ be defined so that its (i, j)-entry is given by $\mathcal{K}_{ij} = \mathcal{K}(x^{(i)}, x^{(j)})$. This matrix is called the *Kernel matrix*. Note that we used $\mathcal{K}$ to denote both the kernel function $\mathcal{K}(x, z)$ and the kernel matrix $\mathcal{K}$, due to their obvious close relationship.

A.1 Mercer Theorem

Let $\mathcal{K} : \mathcal{R}^n \times \mathcal{R}^n \mapsto \mathcal{R}$ be given. Then for $\mathcal{K}$ to be a valid (Mercer) kernel, it is necessary and suficient that for any $[x^{(1)}, ..., x^{(m)}]$, $(m < liminf)$, the corresponding kernel matrix is symmetric positive semi-definite.

Appendix B

Evaluation Metrics

Performance of text classification is measured using the following two way contigency table:

	Classified +	*Classified -*
Correct +	a	b
Correct -	c	d

Table B.1: Two way contingency table

Then we have following metrics, which are usd in this dissertation:

$$Precision(\mathcal{P}) : \frac{a}{a+c} \text{ when } (a+c) \neq 0 \tag{B.1}$$

$$Recall(\mathcal{R}) : \frac{a}{a+b} \text{ when } (a+b) \neq 0 \tag{B.2}$$

$$Accuracy(\mathcal{A}) : \frac{a+d}{a+b+c+d} \text{ where } (a+b+c+d) > 0 \tag{B.3}$$

$$F - measure(\mathcal{F}) : \frac{2\mathcal{P}\mathcal{R}}{\mathcal{P}+\mathcal{R}} \tag{B.4}$$

Bibliography

[1] Fortuna B. String kernels. *Proceedings of the 7th International multi-conference Information Society*, 2004.

[2] Agichtein E. Borthwick A., Sterling J. and Grishman R. Exploiting diverse knowledge sources via maximum entropy in named entity recognition. In *In Proceedings of the Sixth Workshop on Very Large Corpora*, pages 152–160, 1998.

[3] Eric Brill. A simple rule-based part of speech tagger. In *Proceedings of the Third Conference on Applied Natural Language Processing*, pages 152–155, Trento, Italy, March 1992. Association for Computational Linguistics.

[4] Teo CH and Vishwanathan SVN. Fast and space efficient string kernels using suffix arrays. *In Proc. of the 23rd International Conference on Machine Learning*, pages 929–936, 2006.

[5] Asif Ekbal and Sivaji Bandyopadhyay. Named entity recognition using support vector machine: A language independent approach. *International Journal of Electrical and Electronics Engineering*, 2007.

[6] Asif Ekbal and Sivaji Bandyopadhyay. Part of speech tagging in bengali using support vector machine. *International Journal of Electrical and Electronics Engineering*, pages 106–111, 2008.

[7] A. Dalal et. al. Building feature rich pos tagger for morphologically rich languages: Experience in hindi. *ICON*, 2007.

[8] D. Cutting et. al. A practical part-of-speech tagger. *In Proc. of the 3rd Conference on Applied NLP*, pages 133–140, 1992.

[9] Lodhi H. et al. Text classification using string kernels. *Journal of Machine Learning Research*, pages 419–444, 2002.

[10] Jesus Gimenez and Lluis Marquez. Svmtool: A general pos tagger generator based on support vector machines. *irec*, 2004.

[11] Hideki Isozaki. Japanese named entity recognition based on a simple rule generator and decision tree learning. In *Association for Computational Linguistics*, pages 306–313, India, January 2001.

[12] T. Joachims. Making large-scale svm learning practical. *MIT Press*, 1999.

[13] Takeuchi K. and Collier N. Use of support vector machines in extended named entity recognition. In *In: Proceedings of the sixth Conference on Natural Language Learning*, 2002.

[14] Eleazar E Leslie C and Noble WS. The spectrum kernel: a string kernel for svm protein classification. *In Proc. of Pacific Symposium on Biocomputing.*, 2002.

[15] Kuang R. Leslie C. Fast string kernels using inexact matching for protein sequences. *Journal of Machine Learning Research*, pages 1435–1455, 2004.

[16] Percy Liang. *Semi-Supervised Learning for Natural Language*. MASSACHUSETTS INSTITUTE OF TECHNOLOGY, 2005.

[17] D. McDonald. Internal and external evidence in the identification and semantic categorization of proper names. In *Corpus Processing for Lexical Acquisition*, pages 21–39, 1996.

[18] Nam Nguyen and Yunsong Guo. Comparisons of sequence labeling algorithms and extensions. *Proceedings of the 24th international conference on Machine learning table of content*, pages 681–688, 2007.

[19] Brown PF and Pietra VJD. Class-based n-gram models of natural language. *Computational Linguistics*, 18(4):467–479, 1992.

[20] Lane T. Qiu S. The rna string kernel for sirna efficacy prediction. *In Proc. of the 7th IEEE International Conference on Bioinformatics*, 2007.

[21] Mitra P Saha SK and Sarkar S. Word clustering and word selection based feature reduction for maxent based hindi ner. *ACL-08:HLT*, pages 488–495, 2008.

[22] Sudeshna Sarkar Sandipan Dandapat and Anupam Basu. Automatic part-of-speech tagging for bengali: An approach for morphologically rich languages in a poor resource scenario. *ACL*, 2007.

[23] Schlkopf and Smola. Learning with kernels: Support vector machines, regularization, optimization and beyond. *MIT Press*, pages 407–423, 2001.

[24] S. Sundararajan. Crf versus svm-struct for sequence labeling. *Yahoo Research Technical Report*, 2007.

[25] Kudo T and Matsumoto Y. Chunking with support vector machines. *In Proc. of NAACL-2001*, pages 192–199, 2001.

[26] Taku Kudoh Tetsuji Nakagawa and Yuji Matsumoto. Unknown word guessing and part-of-speech tagging using support vector machines. *In Proceedings of the Sixth Natural Language Processing Pacific Rim Symposium*, 2001.

[27] Vladimir Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, 1995.

[28] Li Wei and McCallum Andrew. Rapid development of hindi named entity recognition using conditional random fields and feature induction. *In: ACM Transactions on Computational Logic*, 2004.

[29] G.D. Zhou and J. Su. Named entity recognition using an hmm-based chunk tagger. In *Association for Computational Linguistics*, pages 473–480, 2001.