AN ABSTRACT OF THE THESIS OF

Akshat Sudhakar Surve for the degree of Master of Science in Computer Science
presented on June 12, 2009

Title: Learning Multiple Non-redundant Codebooks with Word Clustering for Document Classification.

Abstract approved:

_____

Xiaoli Fern

The problem of document classification has been widely studied in machine learning and data mining. In document classification, most of the popular algorithms are based on the bag-of-words representation. Due to the high dimensionality of the bag-of-words representation, significant research has been conducted to reduce the dimensionality via different approaches. One such approach is to learn a codebook by clustering the words. Most of the current word- clustering algorithms work by building a single codebook to encode the original dataset for classification purposes. However, this single codebook captures only a part of the information present in the data. This thesis presents two new methods and their variations to construct multiple non-redundant codebooks using multiple rounds of word clusterings in a sequential manner to improve the final classification accuracy. Results on benchmark data sets are presented to demonstrate that the proposed algorithms significantly outperform both the single codebook approach and multiple codebooks learned in a bagging-style approach.

Learning Multiple Non-redundant Codebooks with Word Clustering for Document Classification

by

Akshat Sudhakar Surve

A THESIS

submitted to

Oregon State University

in partial fulfillment of

the requirements for the

degree of

Master of Science

Presented June 12, 2009

Commencement June, 2010

Master of Science thesis of Akshat Sudhakar Surve
presented on June 12, 2009.

APPROVED:

Major Professor representing Computer Science

Director of the School of Electrical Engineering and Computer Science

---

Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Akshat Sudhakar Surve, Author

ACKNOWLEDGEMENTS

The author expresses sincere appreciation to Dr. Xiaoli Fern for her guidance, patience and financial support throughout the master's program and encouragement in the planning, implementation and analysis of this research. I would also like to thank my parents, Mr. Sudhakar Surve and Mrs. Vidya Surve for their moral and financial support and assistance during this research. Without everyone's assistance and help, it would be impossible to finish this thesis. Special thanks should be given to all the students, faculty and staff in Electrical Engineering and Computer Science community, with whom I spent really wonderful two and half years in Corvallis.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

Page

LIST OF FIGURES

LIST OF FIGURES (Continued)

Figure Page

LIST OF FIGURES (Continued)

Figure Page

LIST OF TABLES

LIST APPENDIX OF FIGURES

Figure Page

Learning Multiple Non-redundant Codebooks with Word Clustering for Document Classification

1. Introduction

Imagine going through a million documents, reading each one carefully in order to classify it and deciding its topic. Wouldn't it be great to just have the process automated and have your computer do the same job for you? In this digital age, the applications of document classification are unlimited; some of the most popular being classifying e-mails as spam or not, classifying different news articles according to their category like Sports, World, Business, etc.

1.1 Document Classification

The problem of Document Classification can be described as follows. Given set of documents $\{d_1, d_2 \ldots d_n\}$ and their associated class labels $c(d_i) \in \{c_1, c_2 \ldots c_L\}$, document classification is the problem of estimating the class label of a new unlabeled document. The first step is to build a classifier from the given set of documents with known topics. In this step, we learn the discriminative features between the topics. The resultant classifier can be used to estimate the class of a new document whose topic is unknown.

Take an example of a given collection of unlabelled paper documents containing news articles written on either one of the two topics, Sports or Computers. If any person was asked to sort every document according to its topic, he or she would go through every document by reading each one of them carefully, trying to decide its topic. The most obvious way of classifying a document i.e. deciding its topic, would be by looking at the absence or presence of certain words, or by looking at how many times a particular

word occurs in it. For example, the word Soccer will have a much greater frequency of occurrence in the documents belonging to the Sports category rather than Computers. So if a particular new document contains, say twenty occurrences of the word Soccer, we will most likely classify that document as a Sports document. The order of words is not as important as is the presence or absence or certain words in the document so as to classify it. Machine Learning has used this concept in a similar way in which any document can be represented as an unordered fixed length vector also called the Bag of Words way of representation.

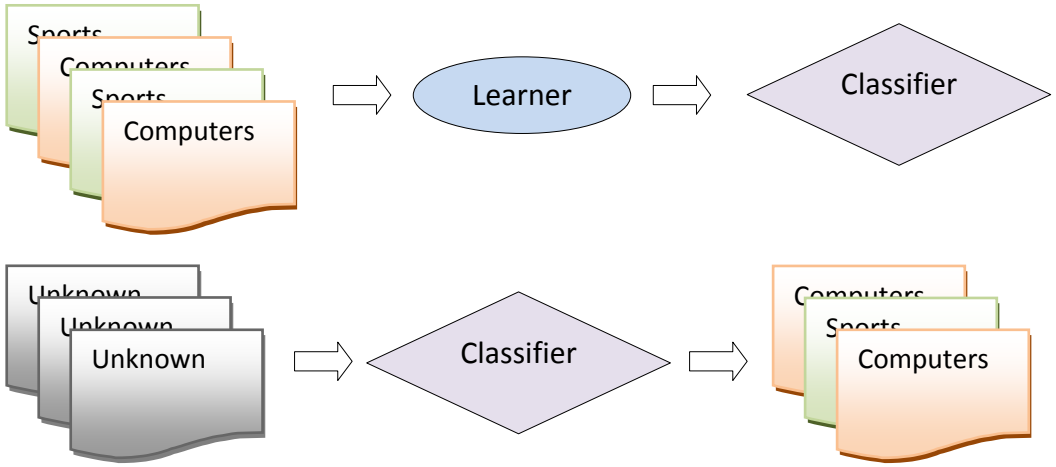

Figure 1.1 Document Classification

1.2 Bag of Words representation

Given any document or a piece of text, it can be represented as a 'Bag of Words'. A Bag of Words model is a way of representing any piece of text, such as a sentence or a whole document, as a collection of words without paying attention to the order of words or their grammatical and semantic context. This is the most common approach for representing text in Machine Learning.

Consider the following piece of text,

Through the first half of the 20th century, most of the scientific community believed dinosaurs to have been slow, unintelligent cold-blooded animals. Most research conducted since the 1970s, however, has supported the view that dinosaurs were active animals with elevated metabolisms and numerous adaptations for social interaction. The resulting transformation in the scientific understanding of dinosaurs has gradually filtered into popular consciousness..

There are different methods of representing the above piece of text as a bag of words. The first method of representing a document as a bag of words is related with the presence or the absence of the word/feature word in it, with a 0 for when it does not appear in the document and a 1 for when it appears in it (e.g. *dinosaurs* 1, *animals* 1, *balloon* 0 …). We can extend this method so that each word is represented by the number of times it occurs in the document, i.e. word counts or word frequencies (e.g. *dinosaurs* 3, *animals* 2, *balloon* 0 …). This form of representation can be further extended to include weights associated with each word. For example, the same word in different documents can have different weights based on the relative importance of that word in that particular document. One such example is the commonly used tf-idf weight, i.e., the term frequency – inverse document frequency weight (Salton et al. 1988).

There are some common problems associated with various forms of the bag-of-words models. The first of these problems is related to high dimensionality. Take the commonly used document corpus 20 Newsgroups for example; it contains approximately 20000 documents and has a vocabulary size of 111445 words (before pruning). Another commonly used benchmark data set, the ENRON dataset, contains about 17000 documents and has a vocabulary size of approximately 51000 words. This can prove to be a serious burden on some of the computationally intensive algorithms such as Support Vector Machines etc. Other issues such as memory (space) required and the amount of time taken to process the dataset is affected as well.

Some additional problems related to the bag of words form of representation are redundancy and noise. Redundancy occurs when multiple words having similar meanings (e.g. car, automobile) occur in documents. Some words can be classified as noise when they are considered irrelevant to the classification topics at hand. Take the example given above of the classification topics, Sports and Computers. Certain words like *games*, which can be related to computer games or sports can occur in both documents. Common words like *the*, *at*, *of* etc which have an equal probability to occur in both set of documents, also can be considered as noise as they do not help us learn the discriminative features. Overfitting can occur as a result which can penalize the classification results. Both redundancy and noise both can severely hurt performance and should be essentially minimized or removed altogether as they can have an adverse effect on learning.

1.3 Different Document Classification Methods

To address the problems of bag of words model of document representation, researchers have come up with several methods to classify documents by learning.

1.3.1 Document Classification by Feature Selection

Feature Selection is a technique in machine learning in which a subset of relevant features is "selected", using some predefined criterion, in order to build robust learning models. This technique is better at removing detrimental noisy features. To solve the problem of high dimensionality researchers have come up with several methods of feature selection (Yang et al. 1997, Pederson 1997, Koller 1997). Some of the popular feature selection methods include feature selection by mutual information (Zaffalon et al. 2002) and Markov Blanket based feature selection (Koller et al. 1996).

1.3.2 Document Classification by Dimensionality Reduction

Feature reduction is another technique developed for classifying documents by learning by dimensionality reduction. Latent semantic indexing (LSI) is a clever way of classifying documents (Deerwester et al. 1990) by feature reduction. It follows a strict mathematical approach without paying attention to the meanings of the words within the documents in order to learn the relationship between different documents. It is a good redundancy-removing method. The drawbacks, however, are that sometimes the results obtained are difficult to interpret. In other words, the results can be justified on a mathematical level but have ambiguous or no meaning in natural language context. LSI, however, results in lower classification accuracy than the feature clustering methods as shown by Baker et al. (1998), which will be discussed shortly. LSI led to the formation of a new technique called the probabilistic Latent Semantic Analysis (p-LSA) model (Hofmann, 1999) and later the Latent Dirichlet Allocation (LDA) model (Blei et al. 2003) which is essentially the Bayesian version of p-LSA.

1.3.3 Document Classification by Feature Clustering

This is a popular method to reduce feature dimensionality. The problem of high dimensionality associated with the bag of words form of representation is solved by the compression achieved by clustering of "similar" words into a much smaller number of word clusters, the latter being the new features (Slonim et al. 2001). The word clusters as a whole can be referred to as a codebook and each individual clusters can be referred to as a single codeword. The codebook is used to map the original bag-of-words representation to a bag-of-codewords representation of much smaller dimensionality, where each codeword is represented by the presence/absence, frequency, or weighted frequency of the words in that cluster. The tens of thousands of words occurring in the vocabulary of a large data set are allocated in much smaller number of clusters, say a couple hundred, maybe less. Due to this compression, a more efficient classification model can be formed, which is easy to work with and implement on a smaller scale, and less prone to overfitting. Existing research has shown that this method is able to maintain the high classification accuracy even after

aggressively compressing the number of features as shown in Baker et al. (1998) and Slonim et al. (2001).

1.4 Ensemble Approach for learning multiple codebooks

The effectiveness of the codebook based (word clustering based) method for text classification led us to consider the following question: a single codebook has limited representation power; can we improve the text classification accuracy by using multiple codebooks? This can be viewed as an ensemble approach to this problem. An ensemble approach builds several classifiers and aggregates the results obtained by each of them individually. In this research, we focus on ensemble approaches that are based on the idea of creating multiple codebooks; in particular, we will consider two types of ensemble approaches which are as follows.

1.4.1 Bagging-style or Parallel ensemble approach

In this method, multiple codebooks are created independently. Each of these codebooks is used to train the classifier independently to obtain a final classification result. The results from all classifiers are independent of each other. The final overall classifier is based on the outputs of the individual classifiers. A potential drawback of this method is that information captured by different codebooks may be redundant and may not be able to further improve the classification accuracy.

1.4.2 Boosting-style or Sequential ensemble approach

This technique proceeds in a sequential fashion to build the codebooks as depicted in the figure shown above. It consists of a fixed number of iterations in which each iteration, the previously built codebook and its classifier are used in some way, either directly or indirectly to build the current codebook. In other words, unlike the bagging style approach, the discriminative information captured or not captured by the

previous codebooks may be used to build the current codebook. The two main algorithms presented in this thesis follow this approach.

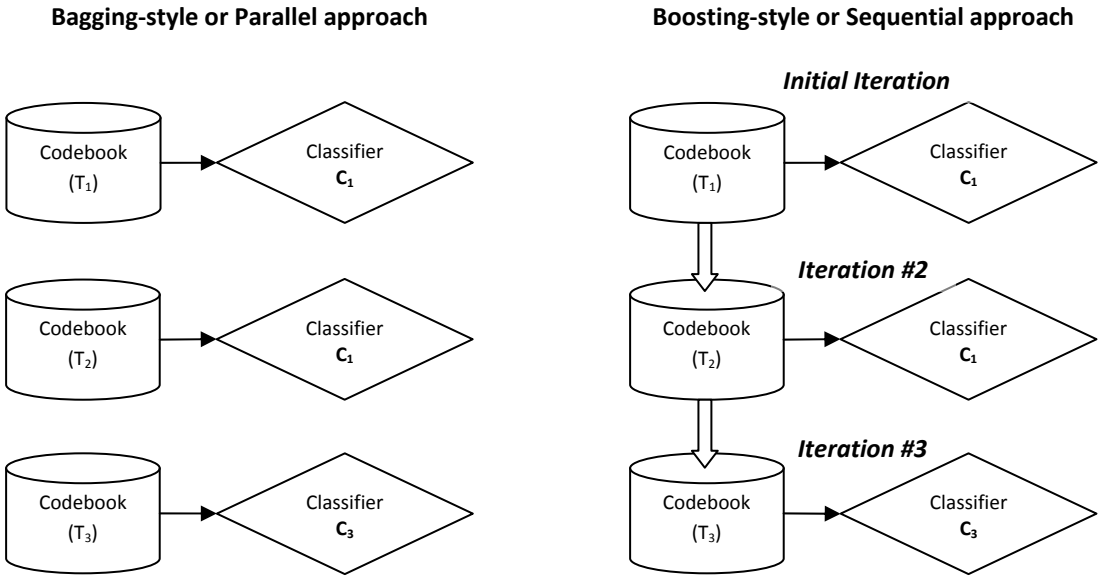

Figure 1.2 Two main types of Ensemble approaches

1.5 Proposed Methods

This thesis proposes two main methods to improve the performance for document/text classification by learning multiple codewords, each with the intention to complement previously learned codewords. These algorithms are explained in brief in the following sub-sections.

1.5.1 Sequential Information Bottleneck with Side Information (S-IBSI)

This technique builds on an existing word clustering algorithm, namely the information bottleneck (IB) (Bekkerman et al., 2003) algorithm, and its extension information bottleneck with side information (IBSI) (Chechik et al., 2002). Both IB and IBSI are supervised clustering techniques. IB learns a clustering of the words while preserving as much information as possible about the document class labels,

whereas IBSI achieves a similar but extended to also avoid irrelevant information provided as the side information.

The basic gist of our approach is that, in each iteration of codebook learning, we consider the information contained in previous codebooks and classifiers as irrelevant side information, such that the codebook learning can focus on the information that has not yet been captured by existing codebooks and classifiers. Two variations of this general approach are proposed in this thesis. The first variation extracts the side information directly from the codebooks of the previous iteration without looking at the resulting classifiers. The second variation extracts side information from the classifier results of the previous iteration. As shown in section 4, both methods achieved significant performance improvement compared to both using a single codebook or using multiple codebooks learned in a bagging style approach which we consider as a baseline.

1.5.2 Sequential Ensemble Clustering via Feature-level Boosting Algorithm

This is another completely new algorithm which has been proposed in this thesis. It follows the same sequential approach to build codebooks. It extends the ADABOOST algorithm to the feature level, by relearning a codebook in each Adaboost iteration. At any given iteration, the low level feature vectors are modified according to the Adaboost weights associated with the documents they represent. As shown in Section 4, this algorithm performs excellently in terms of both speed and final classification accuracy.

In Chapter 2, we will review the related work; researchers have done on codebook learning. It also describes the basic algorithms which have been applied in the framework of our proposed methods. In Chapter 3, we will look in detail at the proposed methods. Chapter 4, contains the results obtained followed by a brief

discussion based on them. We conclude in Chapter 5 followed by Chapter 6 in which we will discuss the future work in brief.

2. Background and Related Work

The problem of document classification has been widely studied in machine learning and data mining. In document classification, most of the popular algorithms are based on the bag of words representation for text (Salton et al, 1983) as introduced in Section 1. There are three main problems associated with the bag of words model particularly when handling large datasets: high dimensionality, noise and redundancy. The dataset size is measured with respect to the number of documents it contains and its total vocabulary size – the total number of unique words that occur in all the documents. Datasets containing a large vocabulary of words and multiple thousand documents (e.g. 20 Newsgroups) can prove to be a burden on the classifiers incorporating computationally intensive algorithms. This is the problem of high dimensionality. Also, datasets having a large vocabulary of words have a high probability of containing multiple words with similar meanings (e.g. car, automobile). Such words are considered redundant with respect to each other and hence the problem of redundancy arises. A large vocabulary may also contain some words that are completely irrelevant to the topics at hand. For example, if the dataset contains documents on two sports topics – *Soccer* and *Hockey*, the word *fish* will be considered as "noise" as it is completely irrelevant to these two topics.

To address these issues associated with the bag of words model, codebook based methods have been developed as described in Baker et al. (1998), Dhillon et al. (2003). The basic objective of codebook learning algorithms is to "learn" a codebook, which can be used to encode the original high-dimensional dataset into a smaller version, which can be easily used for classification purposes, thus solving the problem of high dimensionality. The problem of redundancy is minimized by word clustering, since similar words end up in the same clusters. The problem of noise is minimized to some extent.

In text classification, the most successful word clustering algorithms for learning codewords to date are discriminative in nature. They aim to produce word-clusters that preserve information about the class labels. Representative examples include the Information Bottleneck (IB) approach as described in Bekkerman et al., (2003) and the Information-theoretic Divisive Clustering (IDC) approach described in Dhillon et al., (2003).

In our work, we choose to use the Information Bottleneck approach as our base learner for generating the base codewords (i.e., word clusters). Another approach that is used is the Information Bottleneck with Side Information (IBSI) as described in Chechik et al. (2002). Below we briefly highlight the crux of the IB method and how it is applied within our framework followed by the IBSI extension which has been applied to "learn" non-redundant codebooks. Lastly we describe the basics of boosting and how it has been used to develop an entirely new approach (Feature-Level Boosting) in generating non-redundant codebooks.

2.1 Information Bottleneck (IB) Algorithm

Consider two categorical random variables X and Y, whose co-occurrence distribution is made known through a set of paired i.i.d observations (X, Y). The Information contained in X about Y is the mutual information between those two variables and can be written as,

$$I(X;\ Y) = \sum_y \sum_x p(x,y)\ log\ \frac{p(x,y)}{p(x)p(y)} \tag{2.1}$$

Information Bottleneck is a compression method, in which, we try to compress the source variable X, while maintaining as much information about the relevant variable Y. The information contained in X about Y is 'squeezed' through a compact 'bottleneck' of clusters T, which is forced to represent the relevant part in X with

respect to Y. The resultant Information after compression that is contained in T about Y is given by,

$$I(T;\ Y) = \sum_y \sum_t p(t,y)\, log \frac{p(t,y)}{p(t)p(y)} \leq I(X;\ Y) \tag{2.2}$$

Given a joint probability distribution between X and an observed relevant variable Y, the goal of Information Bottleneck is to find the best tradeoff between accuracy and amount of compression when clustering a random variable X into a set of clusters T. The resultant objective function can be formulated as,

$$L = \ min_{p(t|x)}\{ I(X;T) - \ \beta I(T;Y)\} \tag{2.3}$$

where β determines the tradeoff between compressing X and extracting information regarding Y and $I(\bullet\ ;\ \bullet)$ denotes the mutual information. In document classification problems, each document taken from the training set is represented by a list of words and the number of times they occur in it along with the document's identifying class label. Here, X represents the vocabulary of words and Y, represents the class labels of the documents. The relation between X and Y is summarized by the empirical joint distribution table $P(X,Y)$. Given K, the desired number of clusters, IB learns $X' = \{wc_1, wc_2 \ \dots \ wc_K\}$, a set of K word clusters, that compactly represent X and preserve as much information as possible about the class label Y. The goal of word clustering algorithms is to find a partition T of the words, such that we maximize some score function. In this case, we use the inverse of the objective function, L (2.3).

$$score = \ L^{-1} = \ max_{p(t|x)}\{ \beta^{-1} I(X;T) - \ I(T;Y)\} \tag{2.4}$$

There are many different approaches to finding the final partition T. One of them is the divisive approach, where initially all words are assigned to a single cluster and as the algorithm proceeds, it splits the clusters at each stage. This is repeated until a

desired number of clusters, K, are formed. The goal of divisive or partitional clustering is to find a final partition T containing exactly K clusters, which form the final codebook. This codebook is then used to encode the original dataset to be used for classification purposes. Note that throughout this research we have used the hard clustering approach where each word is associated with a unique cluster. Another approach is the agglomerative approach, which is the exact opposite of the divisive approach, in which we start off with all words assigned to their own individual singleton clusters. At each stage the algorithm chooses two clusters and merges them. This process is repeated until we get the desired final number of clusters, K. Similar to the divisive approach, the goal is to find a final partition T, containing exactly K clusters, which form the final codebook. Both these approaches are not guaranteed to find the optimal partition. Also, the time and space complexity of both these techniques make it infeasible to be applied to large datasets.

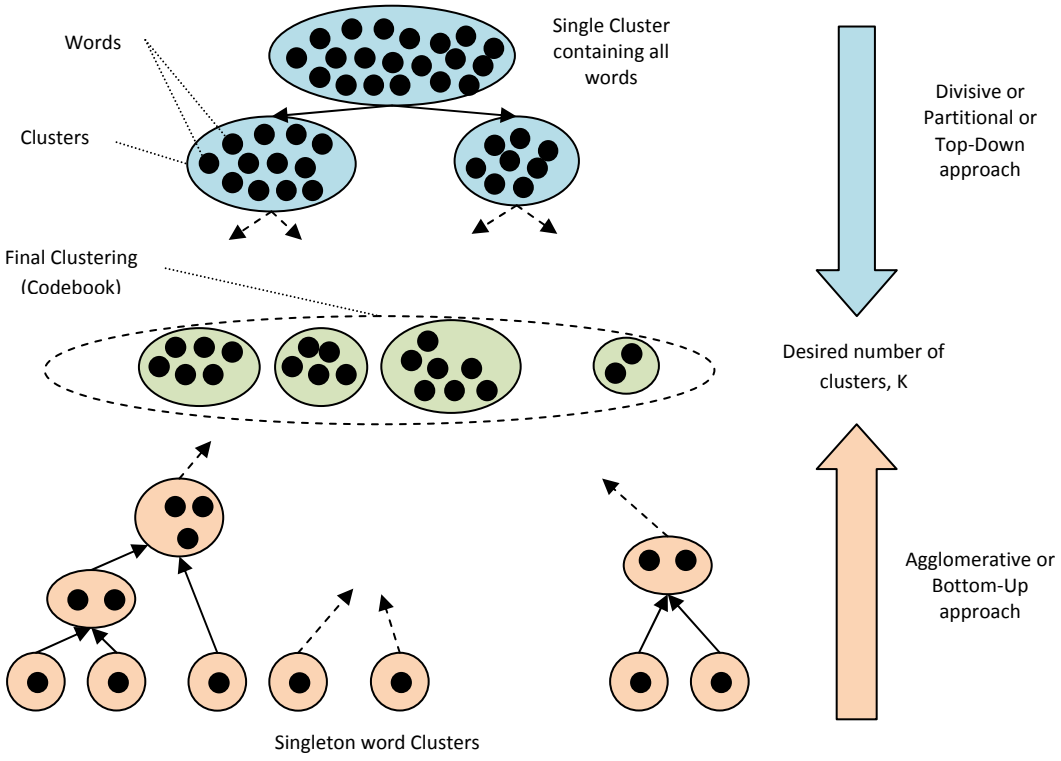

Figure 2.1 Divisive and Agglomerative approaches

In our implementation, we used the sequential Information Bottleneck (sIB) algorithm as described by Slonim et al. (2002). In this approach, we start off with exactly K clusters. Initially, all the words are randomly sampled into exactly K clusters. At each stage we extract a word (shaded in green in Figure 2.2) (selection based on a predefined method, e.g. randomly etc.) from its cluster and represent it using a new singleton cluster. We then apply an agglomeration step, to merge this singleton with a new cluster such that the score improves. This step guarantees the case in which the score either improves or remains unchanged.

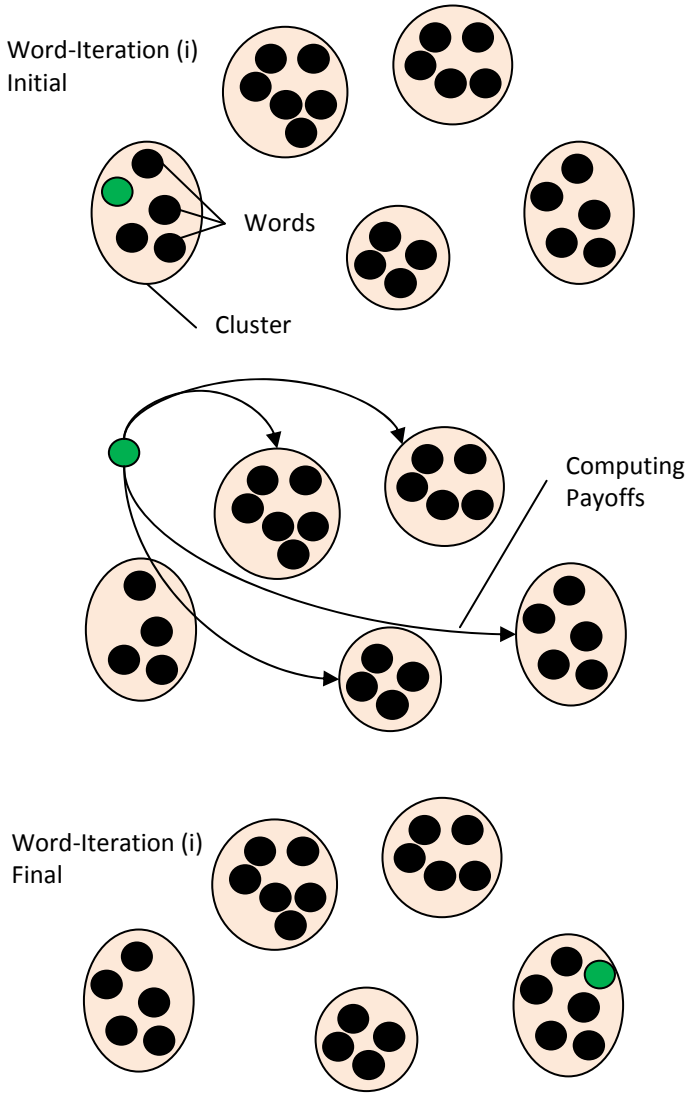

Figure 2.2 Sequential Clustering approach

2.2 Information Bottleneck with Side Information (IBSI) Algorithm

The information bottleneck method compresses the random variable X into T with the goal of maximizing the mutual information between X and Y. An extension to the IB method is the information bottleneck with side information (IBSI) method, where we augment the learning goal by introducing some additional information to inform the learning algorithm what is irrelevant.

To better understand what 'relevant' and 'irrelevant' information is, let us consider the following example. Consider a collection of documents written on two different freshwater species of fish, Discus (*Symphysodon spp.*) and Angelfish (*Pterophyllum*), which are two very similar species of freshwater fish belonging to the *cichidae* family. Both types of fish originally come from the Amazon region; have very similar behavioral patterns, almost similar feeding requirements etc. However Discus fish are very delicate and more demanding of the two species. Their habitat parameters (water temperatures, water pH, nitrate levels etc) need to be carefully maintained in an aquarium as compared to Angelfish. Consider the case in which we need to extract all the information about the maintenance of these two species only, so as to keep them successfully in an aquarium. The relevant information in this case can be considered to be all the information related to the maintenance of these two species only, which we are interested in, for example the feeding patterns, water parameters etc, while the irrelevant information can be considered to be the information related to behavioral patterns. Irrelevant information, also called as 'Side Information', is therefore the information that is irrelevant to the topic at hand, the topic in this case being 'Maintenance of Discus and Angelfish'. Knowing the irrelevant information in advance, we can extract relevant information more accurately from the documents that we are interested in hence increasing the final classification accuracy.

Similar to IB, in this Information Theoretic formulation, let X represent the features i.e. the vocabulary of words, Y^+ represent the documents focusing on the topics we are

interested in and Y^- represent the topics that are irrelevant and considered as the side information.

The goal of IBSI is to extract structures in $P(X, Y^+)$ that do not exist in $P(X, Y^-)$. Hence we need to find a mapping of X to T in a way that maximizes its mutual information with Y^+ and minimizes the mutual information with Y^-. The resultant objective function can be formulated as,

$$L = I(X;T) - \beta[I(T;Y^+) - \gamma I(T;Y^-)] \quad (2.5)$$

where β represents the tradeoff between compression and information extraction and γ represents the tradeoff between the preservation of information about Y^+ and the loss of Information about Y^-.

This can be extended to multiple levels in which we are presented with multiple sets of documents, $\{Y^0, Y^1 \ldots Y^n\}$, in which certain sets can be considered to contain irrelevant or side information.

$$L = I(X;T) - \beta[I(T;Y^0) - \gamma_1 I(T;Y^1) - \gamma_2 I(T;Y^2) \ldots - \gamma_k I(T;Y^k)] \quad (2.6)$$

The approach described in Chechik et al. (2002) builds a single codebook. This approach makes use of the fact that the side information is known beforehand. In this research, we have applied the idea of IBSI on a single data set, by converting it into a sequential algorithm to generate multiple non-redundant codebooks so as to capture most of the information present in the data. The main question is: After generating a codebook, what should we consider to be side information in order to build another non-redundant codebook in the next iteration? Based on this, two main variations have been proposed in this thesis. In the first approach, at the end of iteration k, the clustering algorithm builds a final clustering or codebook X_k'. It can be stated that the

algorithm has “learnt” this final clustering or codebook. In order to avoid relearning or any redundancy, the information contained in this clustering can be considered as irrelevant or side information for constructing the next codebook X'_{k+1}. This is an unsupervised method of generating the next codebook. In the second approach, we have used the obtained codebook X'_k, to train a classifier. The classifier output is then used to extract the documents from the original training set which are correctly predicted by it. The information contained in these correctly predicted documents is considered as side information for building the next codebook. Since the next codebook is built from the predictions made by the classifier, this is a supervised way of building the next codebook.

2.3 Boosting

The second algorithm proposed in this thesis is based on boosting principle. We will now look at the basics of boosting in brief. The basic ideology behind Boosting is ‘Learning from one’s own mistakes’. The goal is to improve the accuracy of any given weak learning algorithm. In boosting, we first create a classifier with accuracy on the training set greater than average, and then add new component classifiers to form an ensemble whose joint decision rule has arbitrarily high accuracy on the training set. In such a case we say that the classification accuracy has been boosted.

Adaptive Boosting, also known as AdaBoost, is a popular variation of basic boosting. In AdaBoost each training pattern (document) is associated with a weight, which helps to determine the probability with which it will be chosen for the training set for an individual component classifier. If a particular pattern is correctly classified, the weight associated with it is decreased. In other words the chance that a correctly classified pattern will be reused for the subsequent classifier is reduced. Consequently if the pattern is incorrectly classified the weight associated with it is increased. In this way we can focus on the patterns that are difficult to learn in order to improve our final classification accuracy.

The basic pseudo code for the ADABOOST algorithm is shown below,

Begin

Initialize:

$D = \{x^1, y_1;\ x^2, y_2 \ldots\ x^n, y_n\}$ (Input Dataset),

$W_0 = \frac{1}{n}, i = 1 \ldots n$ (Weights)

$k = 1$

Do

Train weak learner C_k using D sampled according to $W_k(i)$

Compute E_k, the training error of C_k measured using D using $W_k(i)$

$$\alpha_k = \frac{1}{2}\ln\left[\frac{1 - E_k}{E_k}\right]$$

$$W_{k+1}(i) = \frac{W_k(i)}{Z_k} \begin{cases} e^{-\alpha_k} if\ h_k(x^i) = y_i \text{ (correctly classified)} \\ e^{+\alpha_k} if\ h_k(x^i) \neq y_i \text{ (incorrectly classified)} \end{cases}$$

$k = k + 1$

Until $k > k_{max}$

Return C_k and α_k for $k = 1 \ldots k_{max}$ (ensemble of classifiers with weights)

End

The final classification decision of a test point is based on the weighted sum of the outputs given by the component classifier,

$$g(\mathbf{x}) = \sum_{k=1}^{k_{max}} \alpha_k h_k(\mathbf{x}) \tag{2.7}$$

The method proposed in this thesis, is called Feature-Level Boosting (FLB) (described in detail in Section 3.2). The basic idea is to wrap the codebook construction process (in this case, the IB algorithm) inside a boosting procedure. Each iteration of boosting

begins by learning a codebook according to the weights assigned by the previous boosting iteration. The learnt codebook is then applied to encode the training examples which are used to train a new classifier. The predictions given by the learnt classifier are then used to compute the new weights. In the proposed approach (FLB), we modify the feature frequencies based on the weights of the documents they belong to. The "modified" training examples are supplied as input to the learning algorithm (IB) in the following iteration which generates a next non-redundant codebook.

3. Proposed Methods

Let us begin by defining some important terms. Let the vocabulary of words be represented by W. A word is the low-level feature of the document. Let the original training dataset be represented by the collection of training examples/documents $\{B_i\}_{i=1}^{N}$. The training examples are in the Bag of Words format, where each training example is represented as $B_i = \{w_1^i, w_2^i \ldots w_M^i\}$ where $w_i \in W$ (vocabulary of words). Considering there are multiple categories of labels, each document categorized by a single label l_i. Let us represent the entire set of class labels by L, such that $l_i \in L$. Since this is supervised learning we represent the original training set in the form of joint distribution of words and labels $P(W, L)$. This distribution can be thought of as a composite structure, containing several underlying structures representing the information contained in documents related to various topics. Also let the clustering be denoted by variable T.

All the methods described in Section 2, perform a single clustering to learn a single codebook – this is not enough to capture all the information present in the data. As mentioned before, the original data can be thought of as a composite structure (e.g. Structure 1 in Fig 4.1). In Fig 4.1, the composite structure 1, contains are three different structures, shaded in different shades from black to light gray (ordered by their obviousness or clarity). A single codebook that is "learnt" by the learning algorithm can represent only the most distinct structure. Hence, only the most prominent structures in the data will be "learnt", and represented in the single codebook. The graphical example of this can be described as follows.

The most distinct structure (shaded in black) is the easiest to be noticed and is 'found' by the single codebook learning algorithm represented by Codebook_1. Single codebook learning algorithms stop at this point. However the remaining two structures are still left (to be found) in the data. Using Codebook_1, it is possible to learn another

non-redundant codebook, $Codebook_2$ which represents the next structure. This process can be continued for a predefined fixed number of iterations or until all the structures have been found in the original data.

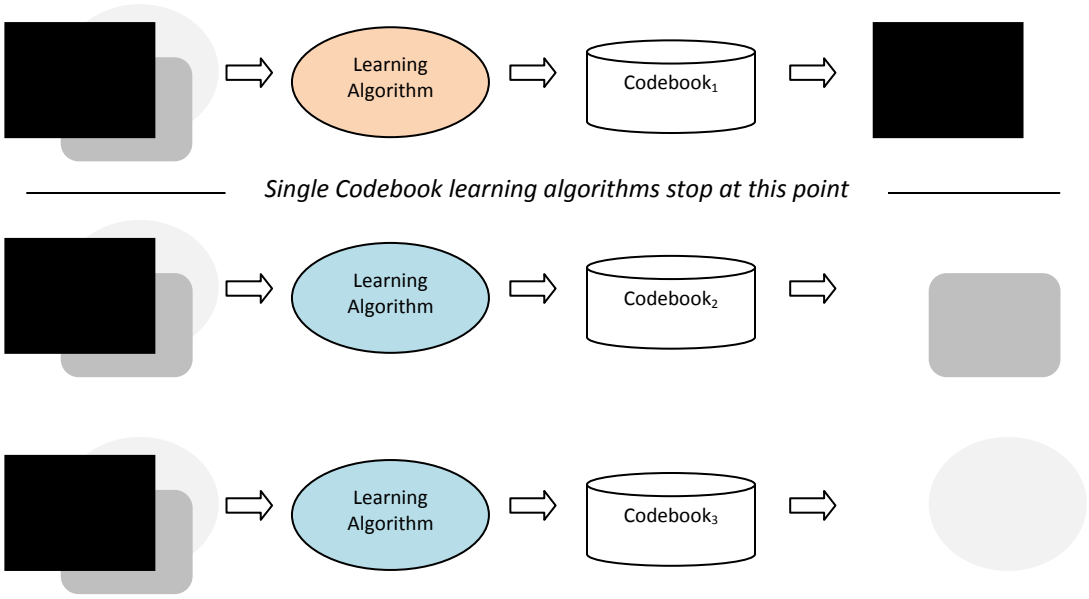

Figure 3.1 Graphical example

Our framework is inspired by the successful application of non-redundant clustering algorithms (Cui et al, 2007; Jain et al., 2008) for exploratory data analysis. Non-redundant clustering is motivated by the fact that real-world data are complex and may contain more than one interesting form of structure. The goal of non-redundant clustering is to extract multiple structures from data that are different from one another; each is then presented to the user as a possible view of the underlying structure of the data. The non-redundant clustering techniques developed by Cui et al. (2007) and Jain et al. (2008) can be directly applied to learn multiple clusterings of the low-level features and create codebooks that are non-redundant to each other. However, such an approach is fundamentally unsupervised and will not necessarily lead to more accurate classifiers. In contrast, our framework learns codebooks that are non-redundant in the sense that they complement each other in their discriminative power.

The first sub-section 3.1, describes two algorithms which are sequential variations of the original Information bottleneck with Side Information algorithm. The two variations mainly differ on what is considered as the side information for the next iteration in order to create the non-redundant codebooks. They are explained in detail in sections 3.1.1 and 3.1.2. Next, a new version of Boosting called as Feature-Level Boosting algorithm is proposed in section 3.2. This algorithm works by modifying the feature frequencies according to the weights of the documents at the end of each iteration.

3.1 Sequential Information bottleneck with Side Information (S-IBSI)

Given a set of training documents, the **aim** of this method is to improve the final classification accuracy by the application of information bottleneck with side information using a sequential approach. At each sequential iteration of the algorithm, a new non-redundant codebook is learned in such a way that it captures new information that was not captured/learned by the previous codebooks and their respective classifiers. Two main variations of this method have been proposed in this paper based on what is considered to be irrelevant or side information.

3.1.1 S-IBSI - Method 1: Side Information $= P(W, T_k)$: Information in the clustering/codebook T_k, at the end of every codebook-learning iteration.

Main Idea

The main idea behind this method is to **consider the information contained in the final clustering obtained at the end of every run as side information** for the next iteration. This is an unsupervised way of computing side information. This method proceeds in three main steps which are as follows:

- **Getting the final clustering/codebook T_k**

The main input that is provided at this step are the training examples, $\{B_i\}_{i=1}^{N}$. As mentioned before, the training examples are in the Bag of Words format where each training example is represented as $B_i = \{w_1^i, w_2^i \ldots w_M^i\}$ where $w_i \in$ W (vocabulary of words). This being Supervised Learning, the training examples are used to compute the joint probability distribution between words and class labels $P(W, L)$. Initially, for the first iteration, $k = 1$, the Information Bottleneck (IB) algorithm is used to learn the training examples. Only $P(W, L)$ is fed as input into the IB algorithm during this initial stage ($k = 1$) since there is no side information available at that moment. The subsequent iterations ($k > 1$), make use of the IBSI (Information Bottleneck with Side Information) algorithm which takes in $P(W, L)$, representing the original training examples, along with the side information arrays $\{P(W, T_1), P(W, T_2) \ldots P(W, T_{k-1})\}$, which represent the final clusterings/codebooks $\{T_1, T_2 \ldots T_{k-1}\}$ respectively, which have been computed in the previous iterations. The output of the learning algorithm at iteration 'k', is a final clustering/codebook T_k, represented using a Word-Cluster Index Key ($WCIK_k$). $WCIK_k$ is a data structure denoting which word belongs to which cluster.

- **Training the Classifier C_k.**

  The final clustering/codebook T_k is used to map the original training examples $\{B_i\}_{i=1}^{N}$ to new fixed length attribute vectors $\{F_i\}_{i=1}^{N}$, where $F_i = \{t_1^i, t_2^i \ldots t_{|T_k|}^i\}$. This mapping is done using $WCIK_k$. The mapping produces a new dataset in which clusters are features. These new attribute vectors are used to train the classifier C_k.

- **Computing the next Side Information Array, SI_k**

  Initially, the final clustering/codebook T_1, is not accurate enough to train the classifier C_1; hence we need to learn a new non-redundant codebook T_2, in the

next iteration. Since we want to find new "structures" in the data, T_1 is considered to contain irrelevant/side information for all the subsequent iterations. Hence at iteration k, T_k is used to compute the joint probability distribution between words and clusters $P(W, T_k)$, which will be used as the additional side information array in the next iteration along with the previously computed SI arrays $\{P(W, T_1), P(W, T_2) \dots P(W, T_{k-1})\}$. Each side information array is assigned a negative weight, $-1/k$. The weights are not set to -1, mainly to avoid overfitting.

The following diagram shows a snap-shot of the k[th] iteration/sequential run for this method.

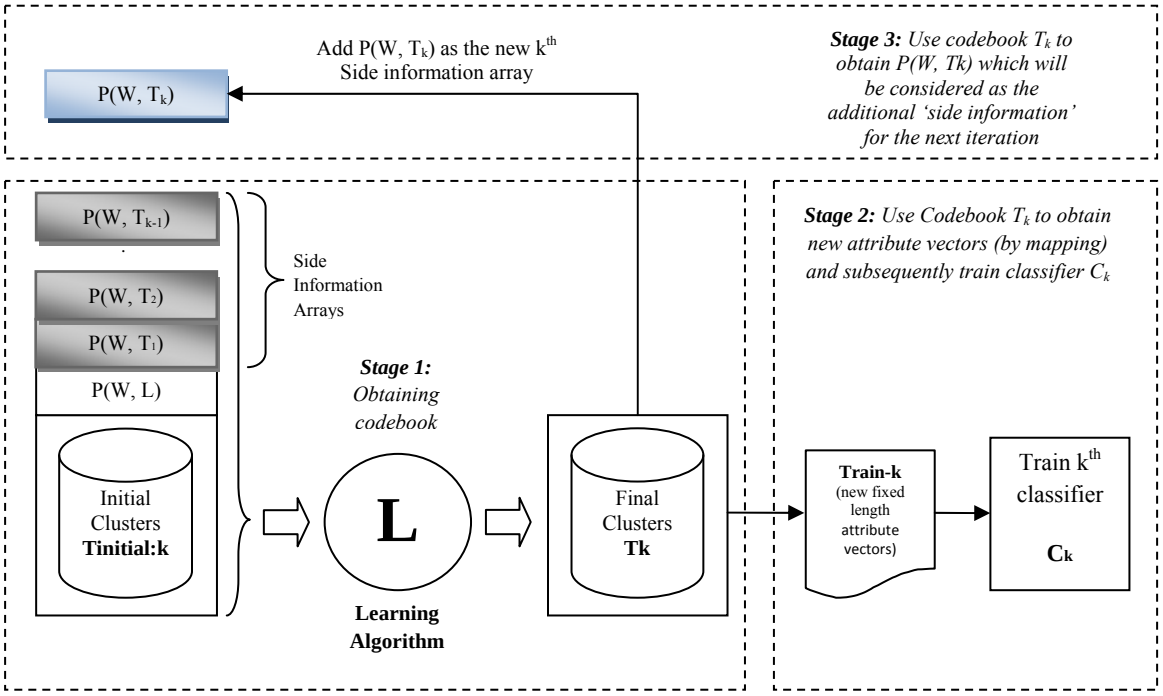

Figure 3.2: Snapshot of the k[th] sequential run in detail with S-IBSI Method 1

At iteration 'k', this can be represented by the following objective function,

$$L_k = \min(\mathrm{I}(\mathrm{W}, \mathrm{T_k}) - \beta\{\gamma_0 \mathrm{I}(\mathrm{W}, \mathrm{L}) + \gamma_1 \mathrm{I}(\mathrm{W}, \mathrm{T_1}) + \gamma_2 \mathrm{I}(\mathrm{W}, \mathrm{T_2}) + \cdots + \gamma_{k-1} \mathrm{I}(\mathrm{W}, \mathrm{T_{k-1}})\}) \quad (3.1)$$

$\{ if\ k = 1, \gamma_1 = -1; if\ k > 1;\ \gamma_i = \frac{-1}{k-1} \}$

where parameter β determines the tradeoff between compression and information extraction, while the parameter $\gamma_i$determines the tradeoff between the preservation of information about relevant variable L representing class labels of the original dataset and the loss of information about the irrelevant variable T_i, which represents one of the final clustering/codebook obtained during the previous iterations. The parameter γ_i is negative in value, since the previous codebooks are considered to contain irrelevant/side information and we wish to find new structures represented by the final codebook T_k.

This is repeated for a predefined number of iterations. The obtained codebooks $\{T_1, T_2 \dots\ T_n\}$ are mutually independent from each other and non-redundant.

Getting the final predictions

The original test set {test-w}, is in the Bag of Words format where each test example $B_i = \{w_1^i, w_2^i \dots w_M^i\}$. Like the training set, this needs to be converted into the new fixed length attribute vector format, in which the clusters themselves are features rather than words. Hence, for the kth iteration, the original test set is mapped into a new fixed-length attribute format using $WCIK_k$, which is a representation of the codebook T_k. This new modified test set $\{\text{test-c}\}_k$ is used to test the obtained classifier C_k to get a set of class label predictions P_k. All predictions $P_1, P_2 \dots\ P_n$, are obtained in this manner. The final prediction is computed by simple voting and subsequently the final classification accuracy is computed.

Drawbacks

In this method, the size of Side Information arrays depends on the number of clusters. The higher the number of clusters, the larger the Side Information arrays $P(W, T_k)$, their dimensions being $n(W) * n(T_k)$. Hence the main drawback of this method is the size scaling of the side information arrays with respect to the number of clusters and

the resultant vast time taken by the implementation of the algorithm if the number of clusters is too large (|T| > 200). Keeping this in mind, two minor variations of this method were devised to make the implementation faster. They were applied only if $n(T) \gg n(L)$. These two variations are discussed in detail in the following sections.

The main flow of this method is illustrated in Figure 3.2 and the pseudo code of this algorithm is as shown below.

Begin

Initialize:

$D = \{x^1, y_1;\ x^2, y_2 \ldots\ x^n, y_n\}$ (Input Training Set),

k_{max}, (Maximum number of non-redundant codebooks to be learned)

$k = 1$

$SI = \emptyset$, Side Information arrays

Do

If $k = 1$

Learn codebook T_1 using $P(X, Y)$ as input to IB algorithm

Else if $k > 1$

Learn codebook T_k using $P(X, Y)$ and SI as input to IBSI algorithm

Encode D using T_k to get $D_k^{'}$, the encoded training set

Train classifier C_k using $D_k^{'}$

Compute $P(X, T_k)$, the joint probability distribution between words and the clusters

$SI = SI \cup \{P(X, T_k)\}$

$k = k + 1$

Until $k > k_{max}$

Return C_k (ensemble of classifiers)

End

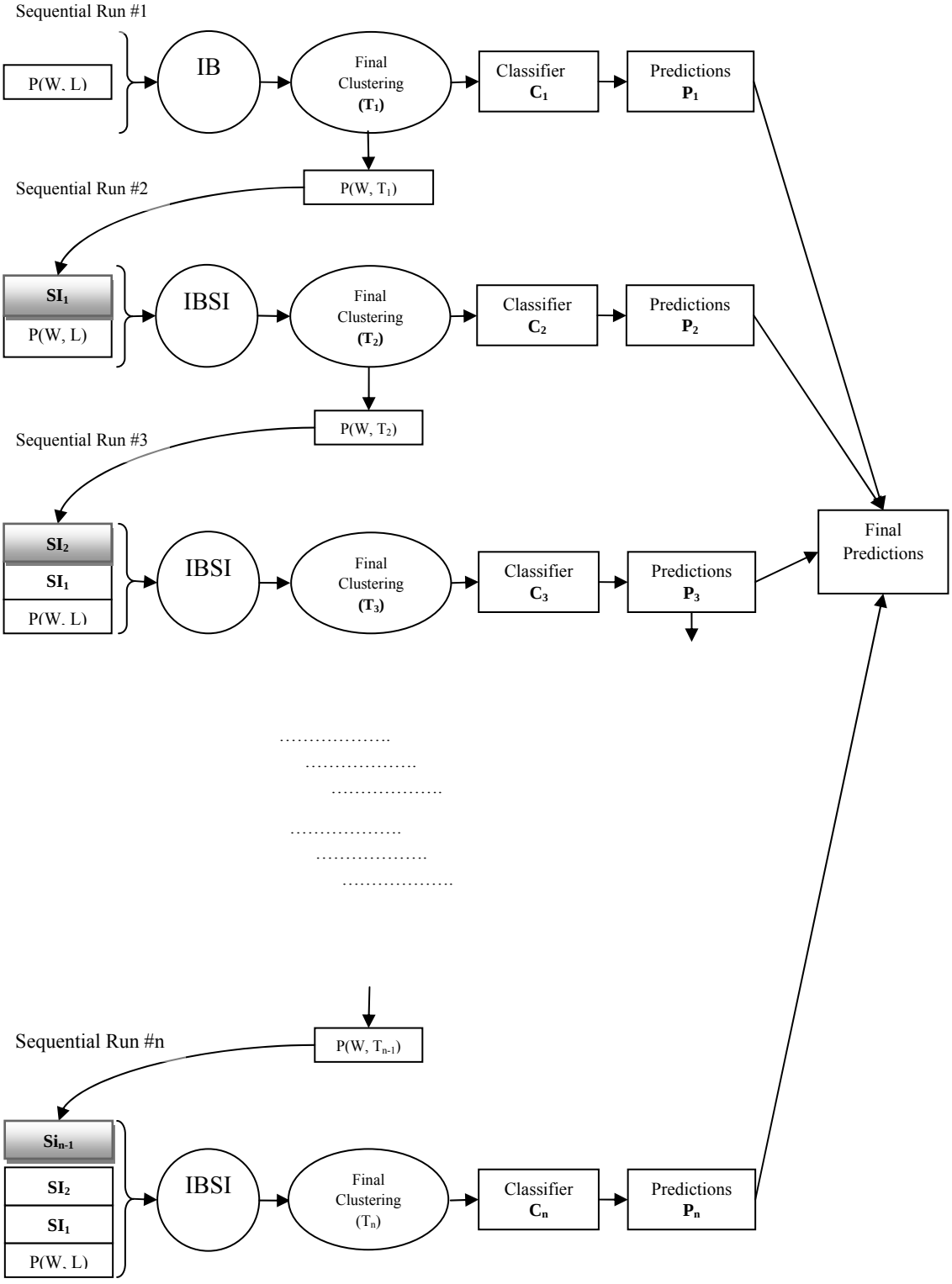

Figure 3.3: Illustration of the basic flow of S-IBSI Method 1

3.1.1.1 IBSI Method 1-a (Speedup variation 1)

These 'speedup' methods were mainly devised with the aim of reducing the computational cost of the algorithm described above, at the cost of some final classification accuracy loss. The first speedup variation, Method-1a is as follows.

At the end of the k[th] iteration, say, a final clustering T_k is obtained as result. These clusters are '**randomly'** merged together so the resultant clustering T_{SC} contains as many super-clusters as the number of class labels i.e. $n(T_{SC}) = n(L)$. This is done so, as so 'reduce' the size of the side information arrays which are represented by the joint distribution $P(W, T_k)$. The resultant Side Information arrays, represented by $P(W, T_{SC})$ have reduced dimensions of $n(W) * n(L)$. This not only saves space but makes the implementation faster as well.

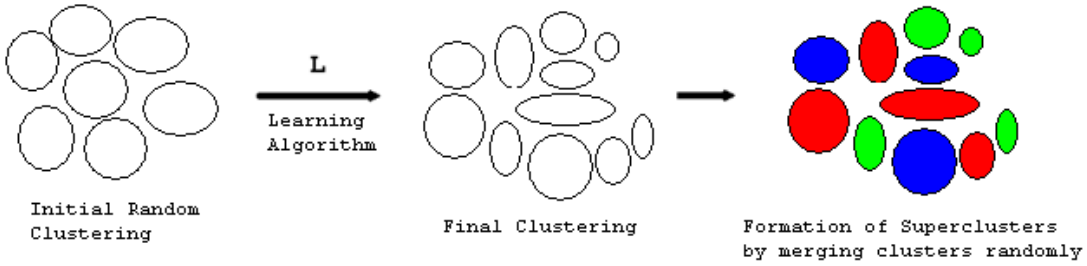

Figure 3.4 Formation of Super clusters with S-IBSI Method 1 – Variation 1

This method does not guarantee preservation of all the information after the merging since it does it randomly. It would be better if the clusters were merged 'intelligently' with some order. The following variation method-1b guarantees that.

3.1.1.2 IBSI Method 1-b (Speedup variation 2)

In this method, the final clustering T_k, represented by the joint distribution $P(L, T_k)$ is fed as input to the learning algorithm, in this case the Information Bottleneck algorithm. The number of clusters in this separate instance of learning algorithm is set to $n(L)$. Here the clusters themselves are clustered into $n(L)$ super-clusters. The

information contained in the resultant super-clusters T_{SC}, can be represented by the joint distribution $P(W, T_{SC})$. The words that were in the original cluster, say $T_{k:x}$, now are a part of the super-cluster $T_{SC:x}$ that $T_{k:x}$ is a part of.

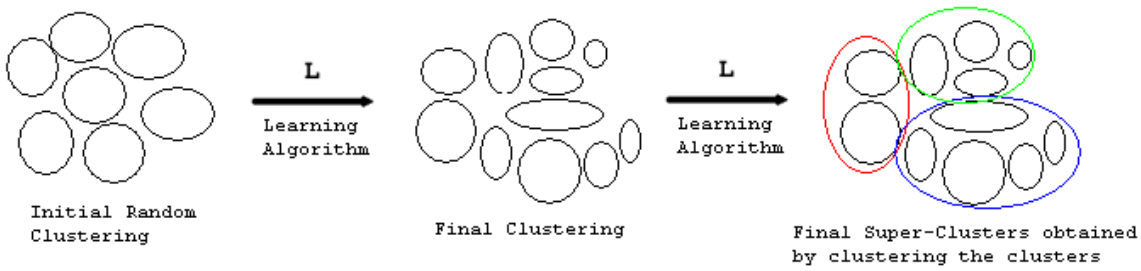

Figure 3.5 Formation of Super clusters with S-IBSI Method 1 – Variation 2

Both speed-up methods shrink the size of the Side Information arrays by "merging" the clusters in the final clustering. Although we lose the granularity of our clusters, the time and space saved is worth it.

3.1.2 S-IBSI: Method 2: Side Information = $P_k(W, L)$: Information contained in the correctly predicted training documents at the end of every codebook-learning iteration.

The main idea behind this method is to consider the information contained in the subset of training set, i.e. the documents "correctly predicted" by the obtained classifier, as irrelevant/side information. Unlike the previous method (described in 3.1.1) this is a supervised way of learning the next codebook. The idea behind this technique is similar to the concept of boosting. This method proceeds in three main steps of which the first two are almost identical to the ones used in Method 1. The steps are as follows:

- **Getting the final clustering/codebook T_k**

The main input that is provided at this step are the training examples $\{B_i\}_{i=1}^{N}$. A random initial clustering $T_{initial_k}$, which is obtained by randomly assigning words to a predefined number of clusters, is and also fed into the learning algorithm at the start of this step. Initially, for the first iteration, $k = 1$, the Information Bottleneck (IB) algorithm is used to learn the training examples. Only $P(W, L)$ is fed as input into the IB algorithm during this initial stage ($k = 1$) since there is no side information available at that moment. The subsequent iterations ($k > 1$), make use of the IBSI (Information Bottleneck with Side Information) algorithm which takes in $P(W, L)$, representing the original training examples, along with the side information arrays $\{P_1(W, L), P_2(W, L) \dots P_{k-1}(W, L)\}$: which represent the joint probability distribution between word and clusters of the training set subsets which contain the correctly predicted documents at the end of each iteration. The output of the learning algorithm at iteration k, is a final clustering/codebook T_k.

- **Training the classifier C_k.**

  The final clustering/codebook T_k is used to map the initial training examples $\{B_i\}_{i=1}^{N}$ to new fixed length attribute vectors $\{F_i\}_{i=1}^{N}$, where $F_i = \{t_1^i, t_2^i \dots t_{|T_k|}^i\}$. This mapping is done using $WCIK_k$, which has been explained in the previous section. These new attribute vectors are used to train the classifier C_k. The trained classifier C_k is then tested with the encoded training set $Train_k$ to get a set of predictions of the training examples, L_{Train_k} (to see how well the classifier has been trained). This technique is very similar to that of Boosting. These predictions will be used in the next step to compute the next side information array.

- **Computing the next Side Information Array, SI_k**

Using predictions L_{Train_k}, the correctly predicted documents are extracted from the original training set (features: words) to form a new subset of training examples $Train\_correct_k$. We use this to compute a new joint probability distribution between words and class labels $P_k(W, L)$. This distribution will be used as the additional side information array in the next iteration along with the previously computed SI arrays $\{P_1(W, L), P_2(W, L) \dots P_k(W, L)\}$. Each side information array is assigned a negative weight = -1, in this method.

The main advantage of this method is that the time taken is very less compared to all previous methods, mainly because all the side information arrays are of a fixed dimensions, $n(W) * n(L)$. This saves a lot of implementation time.

At iteration 'k', this can be represented by the following objective function,

$$L_k = \min(\mathrm{I}(\mathrm{W}, \mathrm{T_k}) - \beta\{\gamma_0 \mathrm{I}(\mathrm{W}, \mathrm{L}) + \gamma_1 \mathrm{I}(\mathrm{W}, \mathrm{L_1}) + \gamma_2 \mathrm{I}(\mathrm{W}, \mathrm{L_2}) + \cdots + \gamma_{\mathrm{k}-1} \mathrm{I}(\mathrm{W}, \mathrm{L_{k-1}})\}) \quad (3.2)$$

$\{\gamma_i = -1\}$

As mentioned before, the parameter β determines the tradeoff between compression and information extraction. However in this method the parameter $\gamma_i$determines the tradeoff between the preservation of information about **relevant** variable L representing class labels of the *original* dataset and the loss of information about the **irrelevant** variable L_i, which represents the set of class labels of the correctly predicted training documents at iteration i. The parameter γ_i is negative in value, since the correctly predicted documents as a whole are considered to contain irrelevant/side information and we wish to find 'new' structures on the current iteration, k. The Figure 3.6 shows a snap-shot of the k^{th} iteration/sequential run for this method.

This is repeated for a predefined number of iterations. The obtained codebooks $\{T_1, T_2 \dots T_n\}$ are mutually independent from each other and non-redundant. The final predictions are obtained in the same manner as the previous variation.

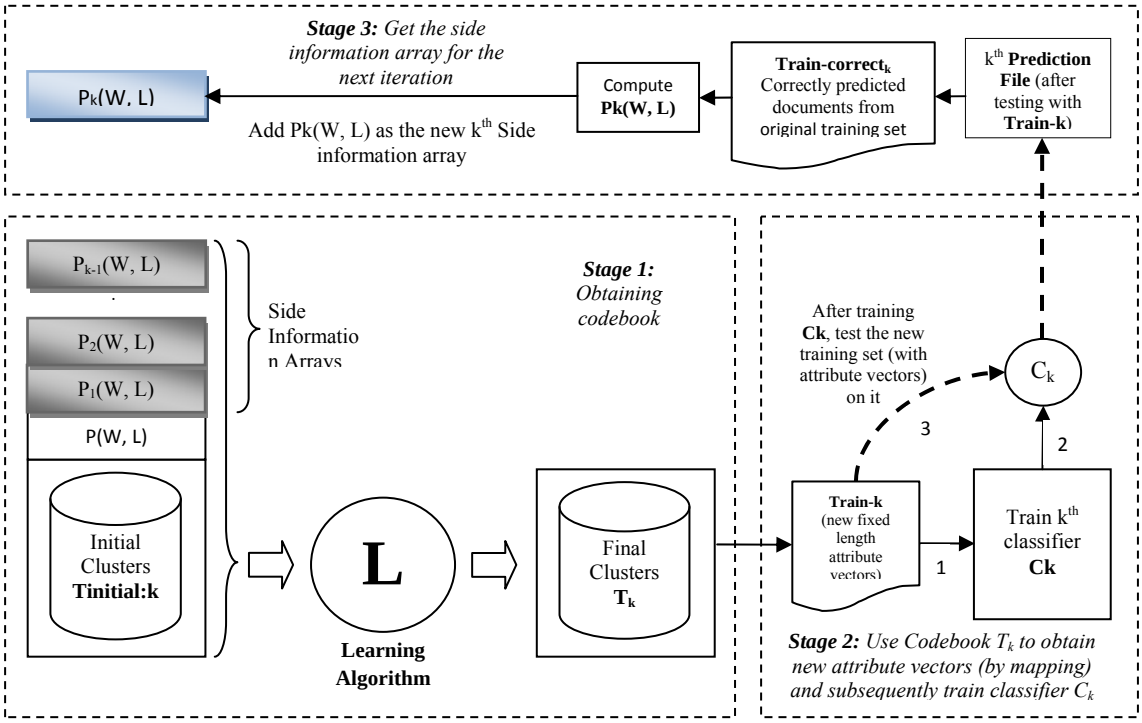

Figure 3.6: Snapshot of the k[th] sequential run in detail with S-IBSI Method 2

Below we can see the pseudo code of this algorithm and the main flow of this method is illustrated in Figure 3.4.

Begin

Initialize:

$D = \{x^1, y_1;\ x^2, y_2 \ldots\ x^n, y_n\}$ (Input Training Set),

k_{max}, (Maximum number of non-redundant codebooks to be learned)

$k = 1$

$SI = \emptyset$, Side Information arrays

Do

If $k = 1$

Learn codebook T_1 using $P(X, Y)$ as input to IB algorithm

Else if $k > 1$

Learn codebook T_k using $P(X, Y)$ and SI as input to IBSI algorithm

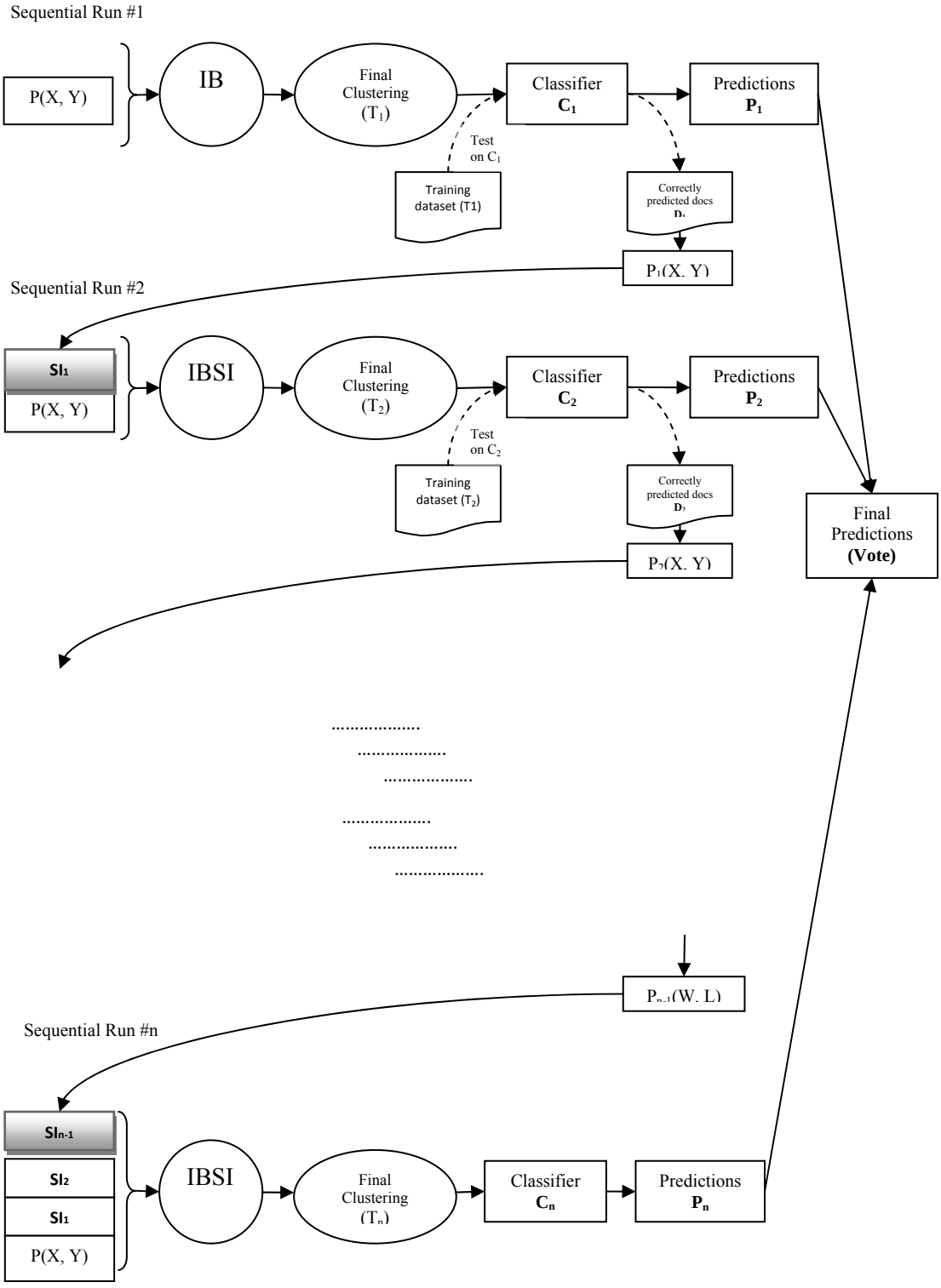

Figure 3.7: Illustration of the basic flow of S-IBSI Method 2

Encode D using T_k to get $D_k^{'}$, the encoded training set

Train classifier C_k using $D_k^{'}$

Test the trained classifier's performance using $D_k^{'}$

Extract the correctly predicted documents from D to get a subset of train $D_k^{''}$

Compute $P_k(X, Y)$, the joint probability distribution between words and class labels from $D_k^{''}$

$SI = SI \cup \{P_k(X, Y)\}$

$k = k + 1$

Until $k > k_{max}$

Return C_k (ensemble of classifiers)

End

3.2 Sequential ensemble clustering via Feature-Level Boosting (FLB)

Description of ADABOOST

In normal Boosting, each pattern/document is associated with a weight which determines its probability with which it will be chosen for the training set for an individual component classifier. The higher the weight, the greater the probability with which it will be chosen and vice versa. If a particular document is correctly classified, the weight associated with it is decreased. In other words the chance that a correctly classified pattern will be reused for the subsequent classifier is reduced. Consequently if the pattern is incorrectly classified the weight associated with it is increased. In this way we can focus on the patterns that are difficult to learn in order to improve our final classification accuracy.

Main Idea

Each training example/document is represented in the bag of words format, represented as $B_i = \{w_1^i, w_2^i \ldots w_M^i\}$ where $w_i \in$ W (vocabulary of words). In every document, a word is represented by its frequency of occurrence, which is the number

of times it appears in that document. At the end of each iteration, for each document, feature-level boosting works by modifying these frequencies according to the instance-weight associated with that document, to produce a new set of training examples modified at "feature-level". This new set of training examples is used to generate a new non-redundant codebook at every iteration. The steps are as follows:

- **Getting the final clustering/codebook T_k**

  The main input provided to the learning algorithm in this step, is the set of training examples $\{B_i^k\}_{i=1}^N$. This being Supervised Learning, the training examples are used to compute the joint probability distribution between words and class labels $P_k(W, L)$. A random clustering $T_{initial_k}$, is obtained, and fed into the learning algorithm as well. The Information Bottleneck (IB) algorithm is used to learn the training examples. The output of the learning algorithm at iteration *k* is a final clustering/codebook T_k. The final clustering is represented using a Word-Cluster Index Key ($WCIK_k$). $WCIK_k$ shows which word belongs to which cluster and will be used in the next step.

- **Training the Classifier C_k.**

  The final clustering/codebook T_k is used to map the initial training examples $\{B_i^0\}_{i=1}^N$ to new fixed length attribute vectors, $\{F_i^k\}_{i=1}^N$, where $F_i^k = \{t_1^i, t_2^i, \ldots, t_{|T_k|}^i\}$. This mapping is done using $WCIK_k$, which has been explained in the previous section 3.1.1. These vectors form a new training set *train-full*$_k$, whose features are clusters (composite codewords) themselves. This step uses the instance/document weights, W_k, to sample the documents from 'new' train, *train-full*$_k$ (which contains ALL fixed-length attribute vectors). Quasi-pseudo random sampling is used to sample the documents. The resultant sampled set, *train-sample*$_k$ is used to train classifier C_k. After training C_k, *train-full*$_k$ is tested on it to get a set of predictions L_{Train_k}, which will be used in the next step.

- **Computing the next Side Information Array, SI_k**

  Using predictions, L_{Train_k}, the weighted training error E_k is computed and subsequently α_1 is computed using the following formula.

$$\alpha_k = \frac{1}{2} ln \left[\frac{1-E_k}{E_k}\right] \quad (3.3)$$

  The weights are updated for all the documents using the following rule,

$$W_{k+1}(i) \leftarrow \frac{W_k(i)}{Z_k} * \begin{cases} e^{-\alpha_k} & (\text{if the i}^{\text{th}} \text{ document is } correctly \text{ classified}) \\ e^{+\alpha_k} & (\text{if the i}^{\text{th}} \text{ document is incorrectly classified}) \end{cases} \quad (3.4)$$

  After all the new weights have been computed, a modified version of the original training set, train-FLB_k is constructed as follows. In the training set, each document is represented as a vector in bag-of-words form, i.e. list of frequency of words and its identifying class label. For each document, each feature frequency **from the original training set,** $\{B_i^0\}_{i=1}^N$ is multiplied by the corresponding weight associated with that document. The new feature-level modified dataset, is used as input for the next run to train the classifier. Looking at this broadly, in the case the document D_i was incorrectly predicted by the resultant classifier, the frequency of the document, D_i itself will be increased in the new training set and vice versa. This new 'feature-level modified' set, is is fed into the learning algorithm as input, on the next iteration to obtain a new non-redundant codebook.

Figure 3.3 shows a snap-shot of the k^{th} sequential run for this method.

Getting the final predictions…

The original test set {test-w}, is too in the Bag of Words format where each test example $B_i = \{w_1^i, w_2^i \ldots w_M^i\}$. For the k^{th} iteration the original test set is mapped into a new fixed-length attribute format using $WCIK_k$, which is a representation of the

codebook T_k. This new modified test set $\{\text{test-c}\}_k$ is used to test the obtained classifier C_k to get a set of class label predictions P_k. All predictions $\{P_1, P_2 \ldots P_n\}$, are obtained in this manner. The final classification decision of a test point is based on the weighted sum of the outputs given by the component classifiers.

$$g(\mathbf{x}) = \sum_{k=1}^{k_{max}} \alpha_k h_k(\mathbf{x}) \tag{3.5}$$

The main flow of this method is illustrated in Figure 3.9. The main advantage of this method from an implementation point of view is that it is very time and space efficient. The time taken by FLB is very less compared to all previous methods and the results obtained are the best.

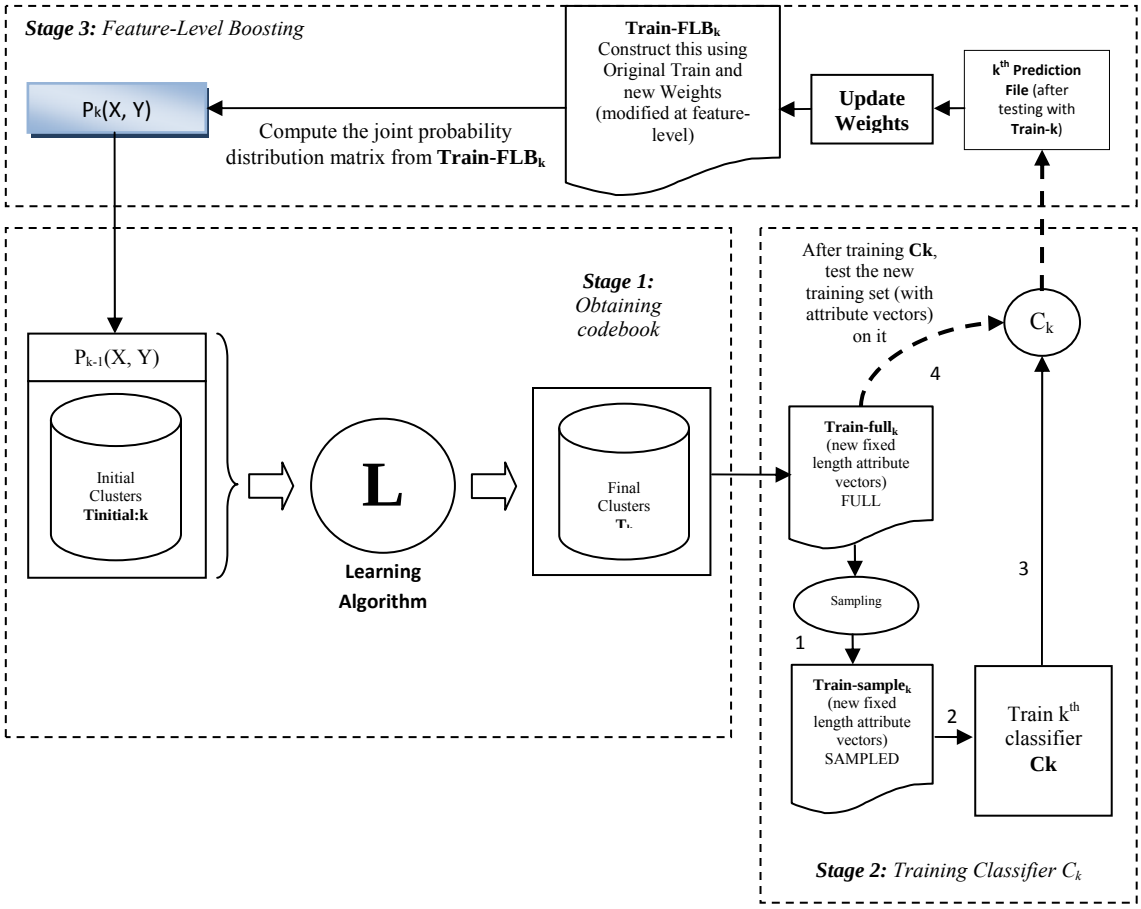

Figure 3.8: Snapshot of the kth sequential run in detail with FLB

Below we can see the basic pseudo-code of the FLB algorithm.

Begin

Initialize:

$D_1 = \{x^1, y_1;\ x^2, y_2 \ldots\ x^n, y_n\}$ (Input Training Set),

k_{max}, (Maximum number of non-redundant codebooks to be learned)

$W_0 = \frac{1}{n}\ , i = 1 \ldots n$ (Weights)

$k = 1$

Do

Learn codebook T_k using D_k

Encode D_k using T_k to get $D_k^{'}$, the encoded dataset

Sample $D_k^{'}$, using W_k to get $D_k^{''}$, the sampled version of the encoded training set

Train classifier C_k using $D_k^{''}$

Compute E_k, the training error of C_k measured using $D_k^{'}$

$$\alpha_k = \frac{1}{2}\ln\left[\frac{1 - E_k}{E_k}\right]$$

$$W_{k+1}(i) = \frac{W_k(i)}{Z_k} \begin{cases} e^{-\alpha_k}\, if\ h_k(x^i) = y_i \text{ (correctly classified)} \\ e^{+\alpha_k}\, if\ h_k(x^i) \neq y_i \text{ (incorrectly classified)} \end{cases}$$

Modify D_1, at the feature-level using $W_{k+1}(i)$, to get D_{k+1}

$k = k + 1$

Until $k > k_{max}$

Return C_k and α_k for $k = 1 \ldots k_{max}$ (ensemble of classifiers with weights)

End

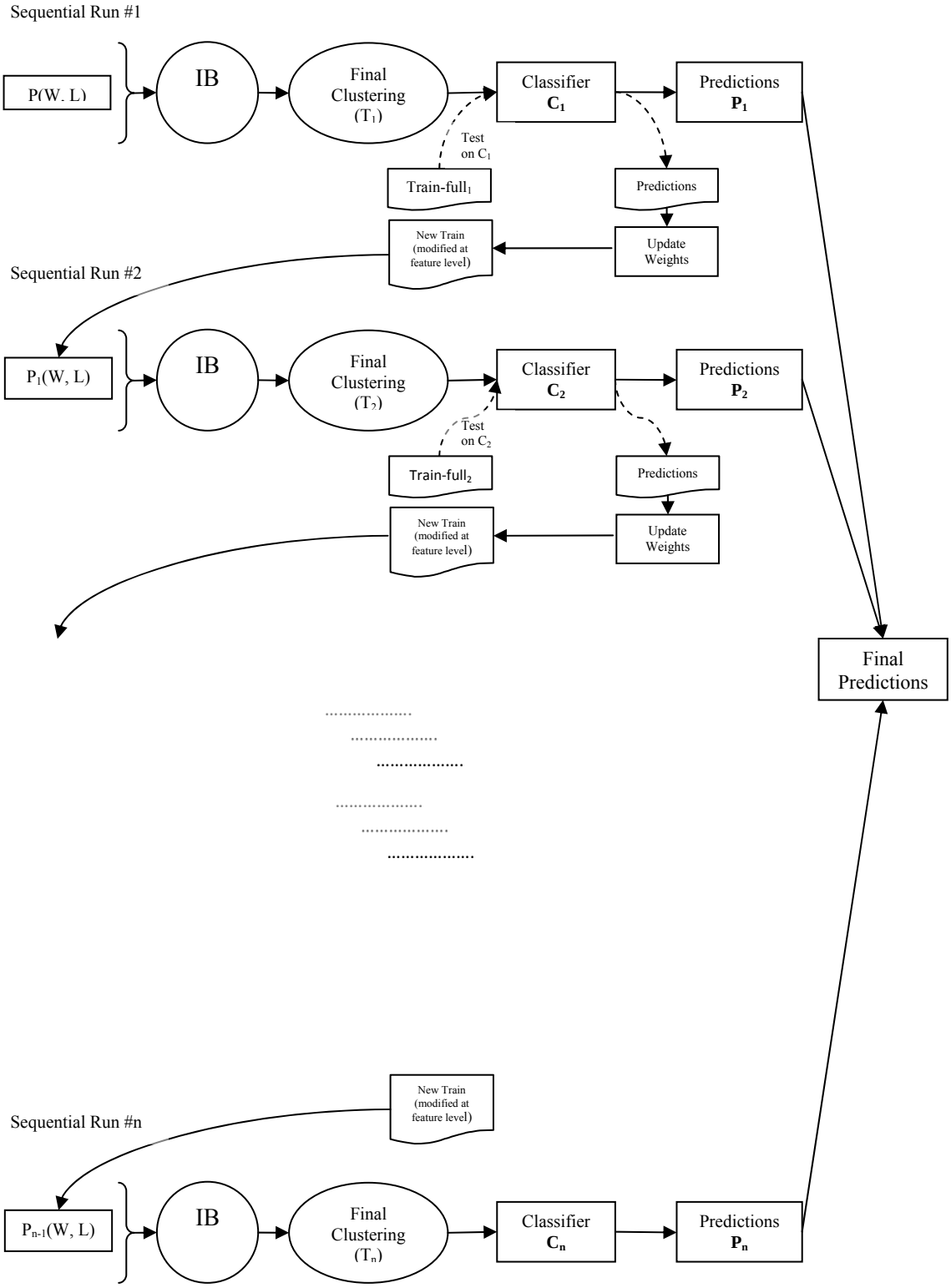

Figure 3.9: Illustration of the basic flow of FLB

4. Experiments

4.1 Experimental Setup

Section 4.1.1 describes the different datasets that were used for our experiments. Next, in section 4.1.2, we will look at the preprocessing steps required to convert the raw text files into a common easily process-able format required by the clustering programs. The tools used to conduct the experiments, i.e. software packages etc are mentioned in brief in 4.1.3.

4.1.1 Datasets

The experiments conducted here make use of the most popular text datasets namely the 20 Newsgroups, ENRON and WebKB datasets which are described briefly as follows.

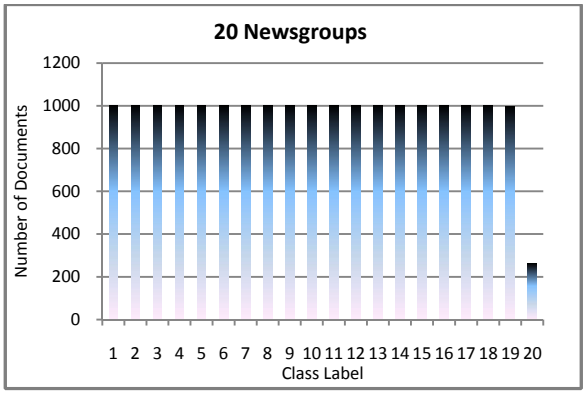

Figure 4.1: Original 20 Newsgroups Class-Document Distribution

20 Newsgroups

The 20 Newsgroups is a classic benchmark dataset popularly used in text applications of machine learning techniques, such as text classification and text clustering. The raw dataset consists of approximately 20000 messages taken from 20 different Usenet newsgroup topics. The documents/messages are organized into 20 different newsgroups, partitioned according to the subject matter. Some of the newsgroups are very closely related to each other, such as topics based on computers (e.g. comp.sys.ibm.pc.hardware and comp.sys.mac.hardware), while others are highly unrelated (e.g rec.autos and alt.atheism). The distribution of documents in each topic

is shown in the figure below. As we can see the documents are evenly distributed across the class topics. For the experiments, we chose 10 random classes and extracted 500 random documents from each of them. This final dataset referred to as *NG10* was used for our experiments.

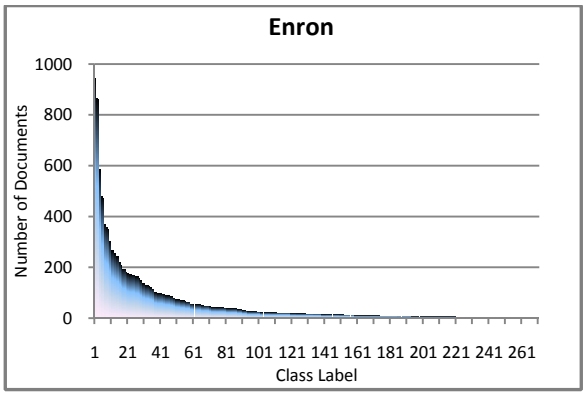

Figure 4.2 Original Enron Document-Class Distributions

Enron

ENRON dataset is the second dataset which has been used in our experiments. This is a descendant of the dataset harvested from Enron's mail server after it was used as evidence against the company and was subsequently put into the public domain. Originally, there were no topic labels available for the Enron dataset and it was therefore appropriate only for unsupervised methods. Recently, a set of annotations for the dataset has been released and it may now be used for supervised methods. The data is a collection of messages associated with a total of seven users. The different types of messages in combination with the users associated with them are considered to be the final class topics for our experiments. For example, in user *Williams* folder, the subfolder *settlements* contains all e-mails related to Williams (email recipient/author) and settlements (type of message). *Williams.Settlements* is taken as the class topic in the final dataset. All class topics are extracted in this manner. The final raw dataset contains a total of 15413 documents distributed across 271 different class topics and a large vocabulary size of 51384 words. The distribution of documents across class

topics is not uniform, the largest topic having 942 documents while the smallest topic having only a single document in it. The distribution of documents in each topic can be seen in figure 4.3. Due to the large number of classes contained in the ENRON dataset and the uneven distribution of documents across class topics, only ten largest topics were selected from the dataset for preprocessing. The final dataset referred to as *Enron10* was used for our experiments.

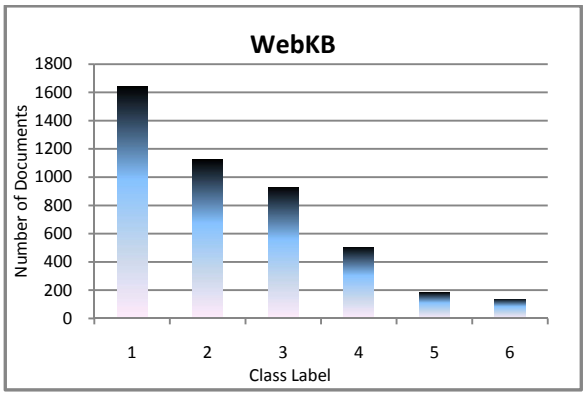

Figure 4.3 Original WebKB Document-Class Distributions

WebKB

The CMU text learning group collected www-pages from computer science departments of various universities for its Worldwide Knowledge Base (WebKB) project. The whole dataset contains a total of 4518 documents classified into six categories (student, faculty, course, project, staff and department) having a total vocabulary size of over 20000 words. For each of these class topics, the pages were collected from four different universities. The distribution of the documents from this dataset can be seen in figure 4.3. For WebKB, only those class topics were selected which had more than 200 documents associated with them (student, faculty, course and project) for creating the final set of documents used for the experiments.

Classic3

Classic3 is a small dataset that was primarily used during the coding, debugging and testing of the code written during the research. It is a small dataset consisting of only three topics and a total of about 3000 documents, the documents being evenly spread out among the topics. This dataset has a small vocabulary size of approximately 5000 words. Due to the small size of the dataset it can be handled very easily by the clustering programs. However, this dataset is very simple to learn compared to the others and was therefore not used in the main experiments.

4.1.2 Preprocessing

This subsection briefly describes the steps involved in the preprocessing of data, which is the conversion of raw data files into a more easily and effectively processes-able format that will be used by the clustering programs. The steps involved are as follows.

Step 1: Raw text files → Bag of Words format

Mallet, a popular text mining software package, was used to preprocess all the datasets which have been used for the experiments. The way it works is explained in brief as follows. It first reads each training document, noting down each word and builds a dictionary, i.e. vocabulary of words. It stems the words such that words like ask, asking, asked are all represented by 'ask' alone. Stopwords like *the*, *and,* etc which have a high uniform frequency of occurrence are considered irrelevant and are removed. The final dictionary is used to count the occurrence of each word (present in the dictionary) in each document to produce a final version of the dataset which is the bag of words form of representation. The output of Mallet is in the Comma Separated Values (CSV) format. However this representation is not used by the clustering programs used in this research. Additional code had to be written to convert this format into Attribute Relation File Format (ARFF) that is used by the Weka machine learning package.

ARFF (Attribute Relation File Format)

We use the ARFF format throughout this research to represent the documents in the datasets. This file format is used as a standard input format for the different clustering algorithms. The following shows the basic template of an ARFF file.

@relation title
@attribute attribute_name$_1$ attribute_type$_1$
…
@attribute attribute_name$_k$ attribute_type$_k$
@attribute class {class_label$_1$, … , class_label$_m$}
@body
{attribute_ index$_1$ count$_1^1$, attribute_index$_2$ count$_2^1$, … , Total attributes Class Label$_1$} ...*#Document 1*
{attribute_ index$_1$ count$_1^2$, attribute_index$_2$ count$_2^2$, … , Total attributes Class Label$_2$} ...*#Document 2*

CSV (Comma Separated Values) file format

Another file format is also used, namely the CSV file format. As the name suggests, this format describes a document by the attribute index and frequency separated by commas. The following shows the basic template of an CSV file format.

Class Label$_1$ attribute_ index$_1$ count$_1^1$, attribute_index$_2$ count$_2^1$, … , attribute_index$_2$ count$_n^1$ …*#Document 1*
Class Label$_2$ attribute_ index$_1$ count$_1^2$, attribute_index$_2$ count$_2^2$, … , attribute_index$_2$ count$_n^2$ …*#Document 2*

Step 2: Pruning of words and documents

To maintain a degree of uniformity we chose words that appeared at least once in a minimum of three different documents. The dataset was further filtered to select documents which contained a minimum of five words in them. This final dataset was used as input to the clustering algorithm. The numbers stated above in section 4.1.1

(number of documents, number of words) are from this stage. The final characteristics of the datasets are shown below in table 4.1.

Dataset	#Class topics	#Documents	#Words
NG10	10	5000	24246
Enron10	10	6188	24812
WebKB	4	4197	16512

Table 4.1 Basic characteristics of the datasets used in the experiments

Two commonly used text classification algorithms were used in the experiments, which are described briefly as follows.

- Naïve Bayes Multinomial (will be referred to as NBM) classifier is a supervised learning classifier popularly used for text classification. It is simple, fast and provides very good results on text data. We use the Weka 3.5.8 implementation in our experiments.

- Support Vector Machines (will be referred to as SVM) is another popular classifier commonly used for text classification. We used the libSVM package (Chang et al., 2009) in our experiments. We experimentally chose the Radial Basis Function (RBF) over the Linear Function mainly due to less time taken for training the classifier and its classification performance. The best cost parameter c, and gamma value g, were found using the *grid* script in the libSVM package.

4.2 Experimental Results

In this section we will first discuss the different baselines we have considered to compare the performance of our proposed methods with, in section 4.2.1. We also

discuss the important parameters used by our IBSI algorithms and the computational complexities associated with each proposed method in section 4.3.2. In section 4.2.3, we will look at the performance curves obtained with a fixed codebook size. Finally in section 4.2.4, we will compare the performance with different codebook sizes.

4.2.1 Baselines

We will first look at the baselines with which we will compare the results obtained by our proposed methods. There are two main types of baselines which are as follows.

- The first baseline is *Single*, in which only a single codebook is learned to represent the data. This codebook is built by applying the IB algorithm to the original empirical joint distribution P(W, L). We examine three different codebook sizes: the first is 100 (this is the standard size of the individual codebooks used in the initial experiments section, referred to as S100), the second is 1000 (which is the total number of code words used by our first ten non-redundant codebooks, referred to as S1000), and the third is 2000 (which is the total number of code words used by all the twenty non-redundant codebooks, referred to as S2000). Results were obtained for single codebooks of size 100, 1000 and 2000 for each datasets. This was done by running for a single iteration and setting the size of clusters to 100, 1000 and 2000 respectively.

- The second baseline is called *Random*, in which ten/twenty bagged samples of the original training corpus are created and each is used to estimate an empirical joint distribution $P_t(X, Y)$ and learn a codebook using the IB algorithm. We have used the FLB algorithm for this baseline. At each iteration, random weights are assigned to each document which will create the illusion of bagging. Note that all training examples are used to learn the classifiers and the bagged samples are only used to produce the codebooks.

4.2.2 Parameters, Complexities etc

IBSI-Method 1 (IBSI-1) is the method described in Section 3.1.1, in which side information is derived from the final clustering (codebook) of the previous iteration. Hence, the size of Side Information arrays (joint probability of words and clusters, $P(W,T)$) is directly proportional to the number of clusters. In our experiments, the number of clusters (100) was far larger than the number of class topics (<=10) and as a result, this method needed a lot of disk space to store its data structures. Mainly due to the large size of the side information arrays, IBSI-1 takes in excess of five hours for a single run of twenty iterations. The IBSI algorithm is very sensitive to the weights, γ_i associated with each side information array. The objective function of IBSI-1, at the k^{th} iteration, is shown below followed by a table which shows the values of weights that were used at each iteration.

$$L_k = \min(\mathrm{I}(\mathrm{W},\mathrm{T_k}) - \beta\{\gamma_0\mathrm{I}(\mathrm{W},\mathrm{L}) + \gamma_1\mathrm{I}(\mathrm{W},\mathrm{T_1}) + \gamma_2\mathrm{I}(\mathrm{W},\mathrm{T_2}) + \cdots + \gamma_{\mathrm{k-1}}\mathrm{I}(\mathrm{W},\mathrm{T_{k-1}})\}) \quad (4.1)$$

β – Compression Vs Information extraction; γ_i – Preservation information about W Vs Loss of Information about T_i

Iteration	$\gamma_i$values
1	$\gamma_0 = 1.$
2	$\gamma_0 = 1;\ \gamma_1 = -1.$
3	$\gamma_0 = 1;\ \gamma_1 = -0.5;\ \gamma_2 = -0.5.$

Table 4.2 Weights, $\gamma_i$at each iteration for IBSI-1

IBSI-Method 2 (IBSI-2) is the method described in Section 3.1.2, in which side information is derived from the training documents correctly predicted by the classifier of the previous iteration. The side information arrays in this case are the joint probability arrays of the words and class labels, $P(W,L)$) . IBSI-2 needs lesser space, since the side information arrays are all of equal size and independent to the number of clusters. IBSI-2 takes a much shorter time of about thirty minutes for a single run. Like IBSI-1, which method is sensitive to the weights as well. After a lot of trial and

error, the weights in the table shown below were found to be the best for each iteration. The objective function of IBSI-2, at the k^{th} iteration, is shown below followed by a table which shows the values of weights that were used at each iteration.

$$L_k = \min(\mathrm{I}(\mathrm{W}, \mathrm{T_k}) - \beta\{\gamma_0 \mathrm{I}(\mathrm{W}, \mathrm{L}) + \gamma_1 \mathrm{I}(\mathrm{W}, \mathrm{L_1}) + \gamma_2 \mathrm{I}(\mathrm{W}, \mathrm{L_2}) + \cdots + \gamma_{k-1} \mathrm{I}(\mathrm{W}, \mathrm{L_{k-1}})\}) \quad (4.2)$$

β – Compression Vs Information extraction; γ_i – Preservation information about W Vs Loss of Information about T_i

Iteration	γ_i values
1	$\gamma_0 = 1$
2	$\gamma_0 = 2;\ \gamma_1 = -1$.
3	$\gamma_0 = 3;\ \gamma_1 = -1;\ \gamma_2 = -1$.

Table 4.3 Weights, γ_i at each iteration for IBSI-2

Feature-Level Boosting (FLB) is the modified boosting approach that is proposed in Section 3.2, in which the feature frequencies are modified according to the boosting weight of the documents. As there are no additional structures like side information arrays involved with this method, it is most space efficient of the three algorithms, where it needs very few data structures to work on. This method is also the fastest of all three methods. It takes approximately only five to ten minutes for a single run, depending on the size of the dataset.

4.2.3 Experimental results for individual codebook size fixed to 100

For the following set of experiments we will fix our individual codebook size to 100. For the proposed methods, every run comprised of 20 iterations, every iteration producing a final clustering of 100 clusters, which can be viewed as a codebook containing 100 code words. Hence 20 such codebooks of size 100 are created, one at

each sequential iteration, each codebook depending on the codebooks created during the previous iterations.

The first comparison with the baselines is done at iteration 10, where we compare the combined classification accuracy from 10 codebooks of size 100 (total size = 10*100) with the baselines, single codebook of size 1000(S1000) and Random (R10). The second comparison is done at iteration 20, where we compare the combined classification accuracy of 20 codebooks of size 100 (total size = 20*100) with a single codebook of size 2000 (S2000) and Random (R20).

For each dataset, each graph shows the performance curves of the three proposed algorithms and the *Random* baseline curve. The results were obtained using two classifiers: Naïve Bayes Multinomial (NBM) and Support Vector Machines (SVM). Finally in the tables, we will summarize the performance accuracies of the proposed methods with the baselines.

For all experiments, we randomly sample two thirds of the documents from the original dataset for training and the remaining examples for testing. The testing examples are not used in any part of the training process. Each of the reported numbers shown in the results is averaged across five random runs. The standard deviations are also shown in the parentheses.

In the following pages, we will see the results obtained with the three datasets. Figures 4.5 and 4.6 show the performance curves obtained with the NG10 dataset followed by table 4.4 which shows the comparison of accuracies at key points (with 10 and 20 codebooks) in order to compare then with the baselines. Similarly figures 4.7 and 4.8 show the performance curves obtained with the Enron10 dataset accompanied by table 4.5. Figures 4.9 and 4.10 show the performance curves obtained with the WebKB dataset accompanied by table 4.6.

4.2.3.1 Newsgroups (NG10) results

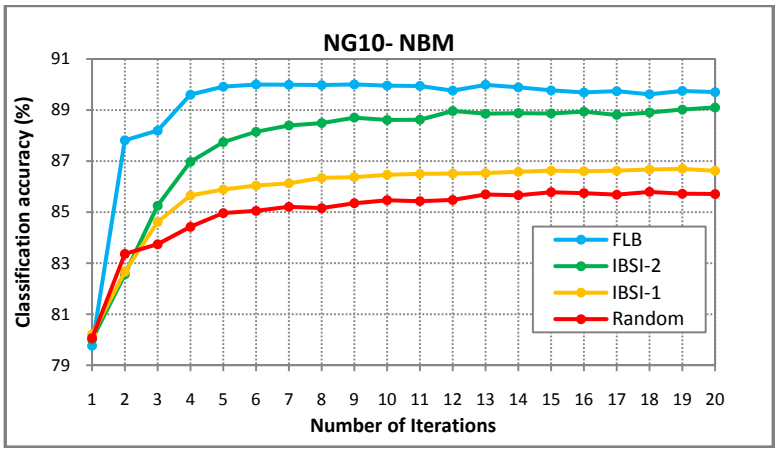

Figure 4.4 Performance of algorithms with NG10 dataset using NBM Classifier

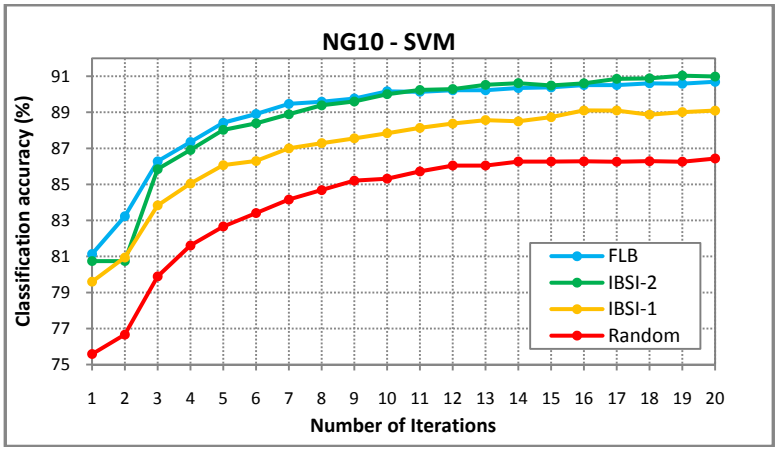

Figure 4.5 Performance of algorithms with NG10 dataset using SVM Classifier

Dataset: 10NG				
Method	NBM		SVM	
	10*100	20*100	10*100	20*100
Single (S1000, S2000)	84.3187 (0.97)	85.4189 (0.44)	82.1536 (0.34)	82.7682 (0.27)
Random (R10, R20)	85.4659 (0.46)	85.7090 (0.64)	85.3178 (0.36)	86.4343 (0.42)
IBSI – Method 1	86.4640 (0.46)	86.6025 (0.76)	87.8394 (0.96)	89.0956 (0.32)
IBSI – Method 2	88.6104 (0.66)	89.1006 (0.75)	90.0038 (0.46)	90.9874 (0.26)
FLB	89.9543 (0.75)	89.7027 (0.66)	90.1649 (0.44)	90.6945 (0.40)

Table 4.4 Classification accuracies (%) of algorithms on NG10 dataset

4.2.3.2 Enron (Enron10) results

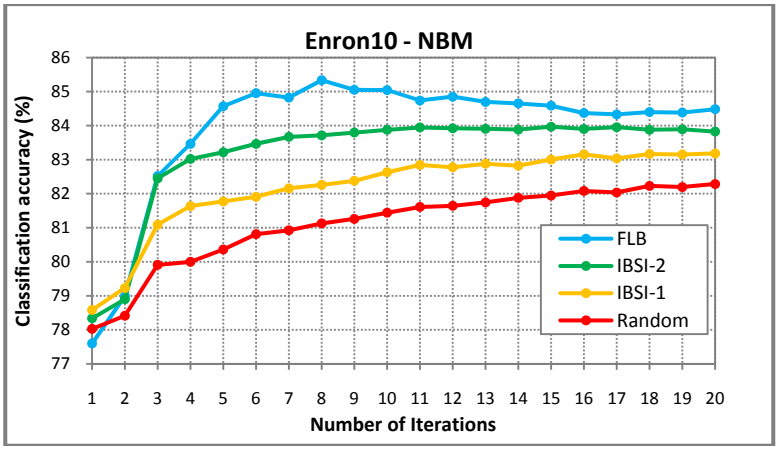

Figure 4.6 Performance of algorithms with Enron10 dataset using NBM Classifier

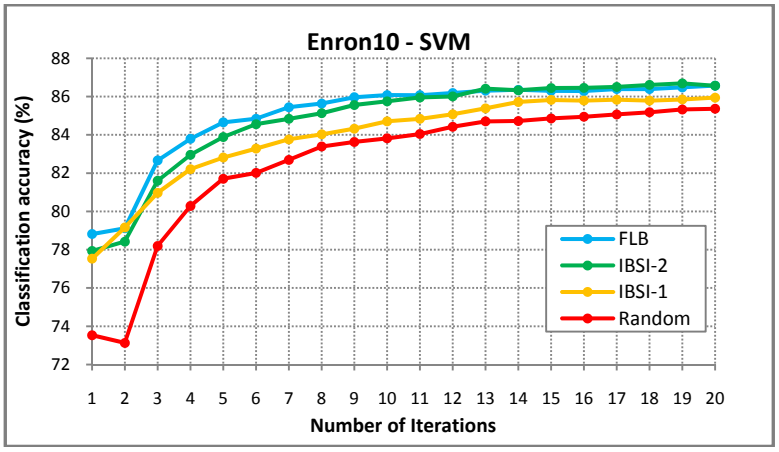

Figure 4.7 Performance of algorithms with Enron10 dataset using SVM Classifier

Dataset: Enron10				
Method	NBM		SVM	
	10*100	20*100	10*100	20*100
Single (S1000, S2000)	80.9022 (0.61)	81.9701 (0.37)	78.1522 (0.17)	77.0463 (0.71)
Random	81.4425 (0.72)	82.2853 (0.43)	83.8136 (0.32)	85.3600 (0.16)
IBSI – Method 1	82.6315 (0.18)	83.1827 (0.31)	84.7043 (0.72)	85.9422 (0.32)
IBSI – Method 2	83.8766 (1.25)	83.8256 (1.33)	85.7497 (0.51)	86.5654 (0.26)
FLB	85.0441 (0.38)	84.4829 (0.65)	86.0767 (0.24)	86.5711 (0.63)

Table 4.5 Classification accuracies (%) of algorithms on Enron10 dataset

4.2.3.3 WebKB results

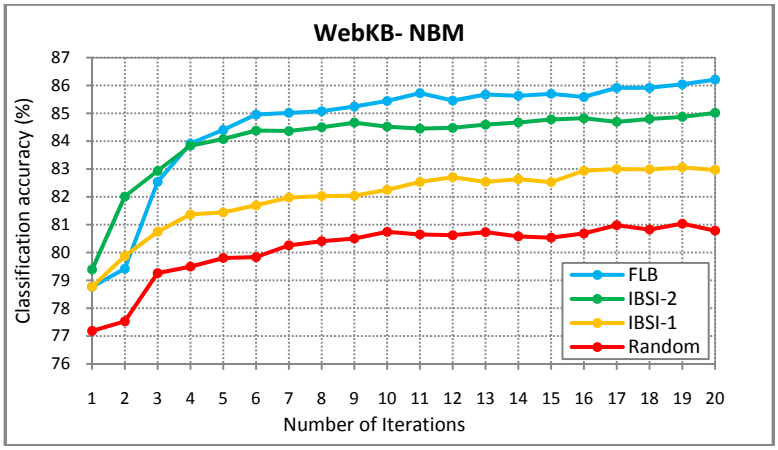

Figure 4.8 Performance of algorithms with WebKB dataset using NBM Classifier

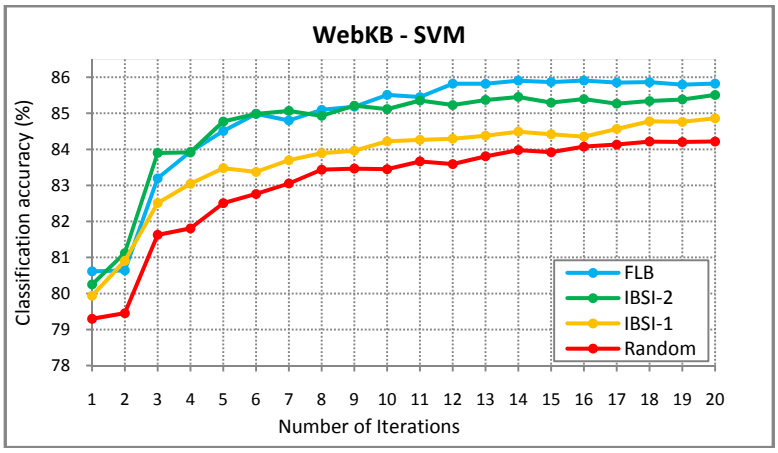

Figure 4.9 Performance of algorithms with WebKB dataset using SVM Classifier

Dataset: WebKB				
Method	NBM		SVM	
	10*100	20*100	10*100	20*100
Single (S1000, S2000)	79.2003 (0.30)	77.9427 (0.48)	78.3231 (0.21)	77.7381 (0.33)
Random	80.7431 (0.54)	80.7859 (0.36)	83.4493 (0.72)	84.2179 (0.67)
IBSI – Method 1	82.2498 (0.62)	82.9678 (0.79)	84.2219 (0.40)	84.8611 (0.28)
IBSI – Method 2	84.5195 (0.65)	85.0129 (0.50)	85.1151 (0.75)	85.5093 (0.61)
FLB	85.4417 (1.14)	86.2082 (0.82)	85.5121 (0.59)	85.8246 (0.59)

Table 4.6 Classification accuracies (%) of algorithms on WebKB dataset

Discussion

All the three proposed methods show a good deal of improvement over the baselines. FLB is clearly the best of the three methods proposed in this thesis. The improvement in classification accuracy is clearly better than both the baselines. This improvement can be due to the fact that FLB is more flexible as it as there are different levels of classification since the individual feature frequencies are themselves modified according to the weights associated with the documents. IBSI-2 comes very close to FLB in some cases. It was all also able to improve over both *Single* and *Random*. The curves are almost identical for NG10 and Enron10 datasets (with SVM classifier). IBSI-1 also shows some improvement. It clearly does not perform as well as the other two algorithms but the results are better than the baselines or at least comparable in some cases.

In IBSI-1, the side information is derived from the final clustering of the previous iteration. This is an unsupervised technique of extracting side information for generating the next non-redundant codebook. On the other hand both IBSI-2 and FLB make use of the classifier to identify the correctly predicted documents from the training set on each iteration. Hence both IBSI-2 and FLB make use of supervised techniques for extracting discriminative information for generating the next non-redundant codebook. From the results, we can clearly observe that IBSI-2 and FLB produce a better improvement than IBSI-1. IBSI-2 was able to effectively capture additional discriminative information than IBSI-1. In other words more information is derived from the classifiers than the information directly derived from the clustering solution. Hence we can conclude that the supervised techniques are more effective at extracting side information, in order to generate the codebook at the next iteration.

From the performance curves shown above we can see that the algorithm tends to converge quicker using NBM classifier. However, the improvement using SVM classifier is better than NBM.

4.2.4 Experimental results for different codebook sizes

In this section we will analyze the comparison of performance of multiple codebook sizes. After observing these results and seeing the efficiency and performance of the FLB algorithm, we will now focus on it to compare different individual codebook sizes. The total codebook size is fixed to 2000 code words. We will examine seven different individual codebook sizes of 20, 50, 100, 200, 500, 1000 and 2000 code words to generate multiple non-redundant codebooks such that the total codebook size is 2000. This method has been labeled as *Multiple* in the results shown below. For example, for individual codebook size of 20 code words, we will have to generate 100 non-redundant codebooks (total codebook size is 100 * 20). The final combined classification accuracy of all the 100 codebooks can be referred to as M20. Similarly, the final classification accuracy of 40 codebooks having individual codebook size of 50 code words can be referred to as M50 and so on. From the results in the previous subsection, SVM curves look more stable compared to NBM and since all the datasets perform better on an average with the SVM classifier, the results in this g section were obtained with SVM classifier only. The results obtained for *Multiple* code book sizes can also be compared with the initial accuracies obtained with a *single* codebook, referred to as *Single*. e.g. the classification accuracy obtained from a single codebook with individual codebook size of 20 code words can be referred to as S20. *Single* results are shown here primarily with the aim of proving that it does not influence the improvement of smaller codebook sizes with respect to the baseline (in this case S2000). We will now look at the results obtained on all three datasets.

4.2.4.1 Performance on NG10 dataset

The final classification accuracies obtained with NG10 dataset are shown in Figure 4.11. In the figure, the blue bars represent the final classification accuracies of multiple codebooks. The red bars represent the single codebook results obtained with the respective codebook size. From the results obtained, we can clearly observe that the performance in classification accuracy peaks when the individual codebook size is

set to 100 code words. Setting the codebook size to 20 or 50, did not improve the final classification accuracy over that of M100. It was however better than the rest. The lower accuracy may be due to the fact that the clusters are too small and hence lesser discriminative information is being captured at each iteration due to the small size of clusters. The accuracies obtained with single codebooks (S20, S50…) appear to attain a maximum at S1000.

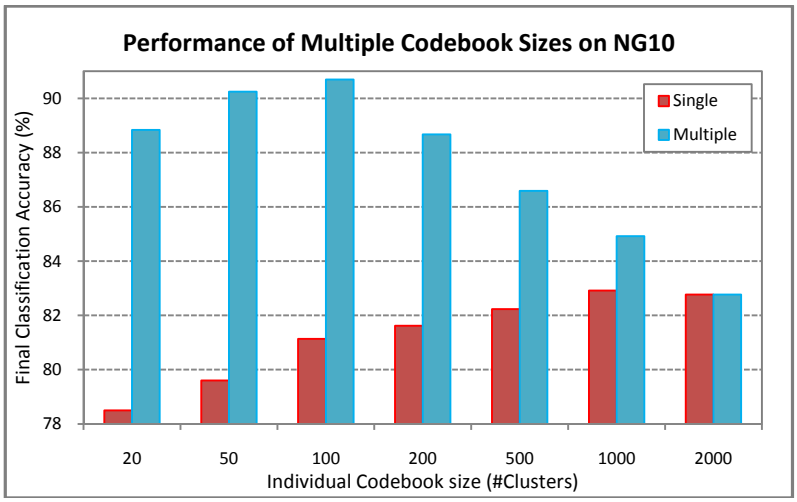

Figure 4.10 Performance of FLB using different codebook sizes on NG10

NG10			
Individual Codebook Size	Single (S)	Multiple (M)	
	Classification Accuracy (%)	# of non-redundant codebooks	Final Classification Accuracy (%)
20	78.4993 (0.93)	100	88.8387 (0.50)
50	79.6036 (0.65)	40	90.2474 (0.56)
100	81.1370 (1.11)	20	90.6945 (0.40)
200	81.6183 (0.31)	10	88.6722 (0.66)
500	82.2340 (0.96)	4	86.5920 (0.53)
1000	82.9151 (0.58)	2	84.9203 (0.80)
2000	82.7682 (0.27)	1	82.7682 (0.27)

Table 4.7 Classification accuracies (%) using different codebook sizes on NG10

4.2.4.2 Performance on Enron10 dataset

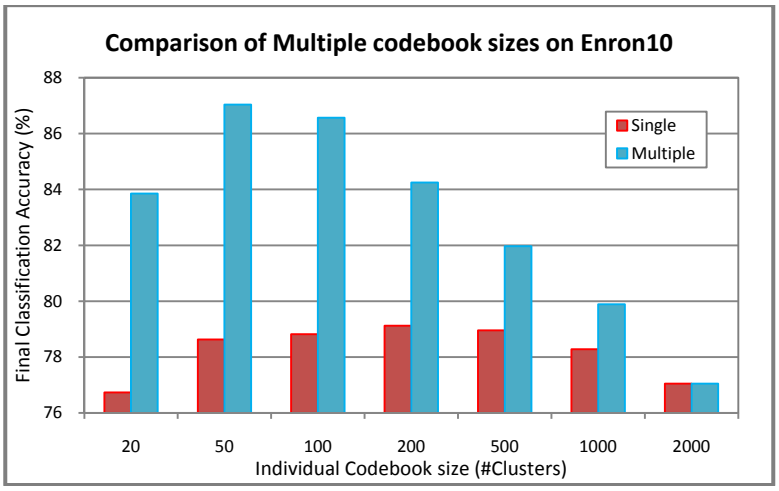

Figure 4.11 Performance of FLB using different codebook sizes on Enron10

Enron10			
Individual Codebook Size	Single (S)	Multiple (M)	
	Classification Accuracy (%)	# of non-redundant codebooks	Final Classification Accuracy (%)
20	76.7320 (0.96)	100	83.8517 (0.92)
50	78.6288 (1.93)	40	87.0414 (0.91)
100	78.8190 (0.85)	20	86.5711 (0.63)
200	79.1189 (0.13)	10	84.2480 (0.35)
500	78.9536 (0.40)	4	81.9662 (0.51)
1000	78.2818 (0.46)	2	77.8872 (0.61)
2000	77.0463 (0.72)	1	77.0463 (0.72)

Table 4.8 Classification accuracies (%) using different codebook sizes on Enron10

From the Enron dataset, the performance in classification accuracy peaks when the individual codebook size is set to 50 code words (M50). Setting the codebook size to 20, did not improve the final classification accuracy over that of M50. It was however

better than the codebooks having size of 500 code words and above. For this dataset, the accuracies obtained with single codebooks appear to attain a maximum at S200.

4.2.4.3 Performance on WebKB dataset

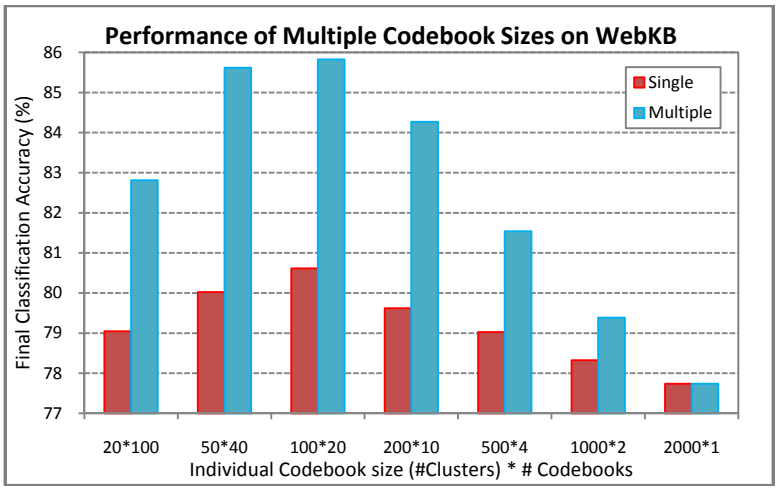

Figure 4.12 Performance of FLB using different codebook sizes on WebKB

WebKB			
Individual Codebook Size	Single (S)	Multiple (M)	
	Classification Accuracy (%)	# of non-redundant codebooks	Final Classification Accuracy (%)
20	79.0465 (1.02)	100	82.8140 (0.70)
50	80.0228 (0.44)	40	85.6173 (0.92)
100	80.6143 (1.22)	20	85.8246 (0.0.59)
200	79.6195 (0.86)	10	84.2672 (0.05)
500	79.0264 (0.63)	4	81.5410 (0.41)
1000	78.3231 (0.21)	2	79.3855 (0.56)
2000	77.7381 (0.33)	1	77.7381 (0.33)

Table 4.9 Classification accuracies (%) using different codebook sizes on WebKB

From the results obtained using the WebKB dataset, the performance in classification accuracy peaks when the individual codebook size is set to 100 code words (M100).

Setting the codebook size to 50, appears to have comparable results to M100. For this dataset, the accuracies obtained with single codebooks appear to attain a maximum at S100.

Discussion

Looking at the results above, we can conclude that there is an improvement in the classification accuracy when the codebook size is decreased. There appears to be a critical point for every dataset when the maximum is achieved (e.g. for the NG10 dataset the maximum is achieved when the codebook size is set to 100). Decreasing the codebook size beyond this critical point does not improve the classification accuracy over the accuracy obtained at the critical point. However, from the results we can observe that it is still better than the baseline, S2000. This may be due to the fact the small size of codebooks may not be able to capture enough discriminative information to affect the final combined classification accuracy. Also after comparing the accuracies obtained using single and multiple codebooks (of same codebook size), it appears that the single codebook classification accuracy does not affect the classification accuracy obtained using multiple codebooks.

5. Conclusion

This thesis proposes multiple techniques to design a framework for learning non-redundant codebooks in order to classify documents based on the bag of words representation more accurately. The results clearly show that the methods were significantly able to improve the classification accuracy over the baselines. This shows that the proposed methods were able to capture additional discriminative information, the information which was not captured by the baselines. Looking back at the space and time complexities of the three methods, FLB took the shortest amount of time for a single run, needed the least number of data structures, and produced the best improvement of all the three algorithms. Of the two IBSI methods, IBSI-2 performed significantly better than IBSI-1 in terms of both classification accuracy and time taken.

Based on the performance of FLB, we had selected it to carry out more experiments to compare the performance with different codebook sizes. We can see from the results that the final classification accuracy of multiple non-redundant codebooks increases when the codebook size is decreased. However this increase is only up to a certain point i.e. a certain codebook size which is unique for each dataset. On further decreasing the codebook size after this point, the final classification accuracy starts decreasing as shown by the results. However, this decrease is not that significant and is still comparatively better than the accuracy obtained with certain larger codebook sizes. This has great implementation benefits: as the experiments can be carried out on a much smaller scale (using smaller codebook sizes) and still produce a better final classification accuracy.

Support Vector Machines appears to be more robust of the two classifiers used to conduct experiments. The performance curves look more stable and smoother on SVM than Naïve Bayes classifier. SVM does not show any signs of overfitting in any of the experiments and all the algorithms tend to converge after about ten to fifteen iterations. With the Naïve Bayes classifier, the algorithms appear to converge very

fast, within the first five to eight iterations. Some overfitting occurs on the Enron10 data set, with the FLB algorithm.

6. Future Work

For the Future work, the efficiency of the proposed algorithms can be improved so they can be applied to larger scale experiments with more variety of data sets. From the results, we can observe that each dataset is associated with a unique critical codebook size at which the final classification accuracy is the best. Finding this out is imperative as it can allow a better knowledge understanding of the dataset.

For the algorithms based on IBSI, we can think of "What else can we consider as side information?" The weights of the side information arrays can also be fine-tuned to produce better classification accuracy.

For, FLB, for each document, instead of changing the feature/word frequencies by a fixed common amount (the document weight), we can make it more flexible by having a weight associated with the combination of the word and document.

Bibliography

20 Newsgroups – Retrieved from http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-20/www/data/news20.html

Burges C. (1998) A Tutorial on Support Vector Machines for Pattern Recognition, *Data Mining and Knowledge Discovery 2:121–167, 1998*

Baker L. D., McCallum A., (1998) Distributional Clustering of Words for Text Classification, *Proceedings of SIGIR'98, 21st ACM International Conference on Research and Development in Information Retrieval*, pages 96–103, Melbourne, AU. ACM Press, New York, US

Bekkerman R., El-yaniv R., Tishby N., Winter Y., Guyon I., Elisseeff A., (2003). Distributional word clusters vs. words for text categorization, *Journal of Machine Learning Research, Vol 3 1183-1208*

Blei D., NG A., Jordan M., (2003) Latent Dirichlet Allocation, *Journal of Machine Learning Research 3 993-1022*

Chang C, Ling C., (2009) *LIBSVM - A Library for Support Vector Machines* http://www.csie.ntu.edu.tw/~cjlin/libsvm/

Chechik G., Tishby N., (2002) Extracting Relevant Structures with Side Information, *NIPS, pp. 857 – 864.*

Cui Y., Fern X., Dy J., (2007) Non-redundant Multi-view Clustering via Orthogonalization, *ICDM, pp. 133-142.*

Deerwester S., Dumais S., Furnas G., (1990) Landauer T., Harshman R., Indexing by Latent Semantic Analysis, *Journal of the American Society for Information Science, 41(6):391-407*

Dhillon I., Mallela S. and Kumar R. (2003) A Divisive Information-Theoretic Feature Clustering Algorithm for Text Classification *Journal of Machine Learning research, Vol 3 1265-1287*

Duda R., Hart P., Stork D., *Pattern Classification, (2nd Ed.)*

Fern X., Bradley C., Cluster Ensembles for High Dimensional Clustering: An Empirical Study, *Technical report, CS06-30-02.*

Hofmann T., (1999) Probabilistic latent semantic Indexing, In *Proc. ACM SIGIR. ACM Press, August 1999*

Hsu C., Chang C, Ling C. (2009) A Practical Guide to Support Vector Classification http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf

Jain P., Meka R., Dhillon I. S., (2008) Simultaneous Unsupervised Learning of Disparate Clusterings, *Statistical Analysis and Data Mining. Vol 1, pp. 195-210*

Jain A., Murty M., Flynn P., (1999) Data Clustering: A Review, *ACM Computing Surveys, Vol. 31, No. 3, September 1999*

Kalal Z., Matas J. Mikolajczyk K., (2008). Weighted sampling for large-scale boosting, *BMVC*

Klimt, B. & Yang, Y. (2004) The Enron Corpus: A New Dataset for Email Classification Research *European Conference on Machine Learning*

Koller D., Sahami M., (1996) Toward optimal feature selection. *In Proceedings of International Conference on Machine Learning (ICML-96), pages 284–292*

McCallum A. K., (2002) MALLET: A machine learning for language toolkit http://mallet.cs.umass.edu

Pereira, F., Tishby, N., Lee, L. (1993) Distributional Clustering of English Words *In Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics*

Rish I. (2001) An Empirical Study of the Naive Bayes classifier, *IJCAI 2001 Workshop on Empirical Methods in Artificial Intelligence.*

Salton G., McGill M., (1983) Introduction to Modern Retrieval, *McGraw-Hill, New York*

Salton G., Buckley C., (1988). Term-weighting approaches in automatic text retrieval, *Information Processing & Management 24 (5): 513–523.*

Sebastiani F., (2002) Machine learning in automated text categorization. *ACM Computing Surveys, 34(1):1–47, 2002*

Shwartz S., Singer Y., Srebro N., (2007) Pegasos: Primal Estimated sub-GrAdient Solver for SVM, *In Proceedings of the 24th International Conference on Machine Learning, Corvallis, OR, 2007*

Slonim N., Friedman N., Tishby N., (2002) Unsupervised Document Classification using Sequential Information maximization, *In Proceedings of the 25th International*

ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR02)

Slonim, N., Tishby, N., (2001) The Power of Word Clusters for Text Classification, *In Proceedings of the 23rd European Colloquium on Information Retrieval Research*

Slonim N., Tishby N., (2000) Document Clustering using Word Clusters via the Information Bottleneck Method, *In Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*

WebKB, The 4 Universities Dataset. Retrieved from http://www.cs.cmu.edu/afs/cs/project/theo-20/www/data/

Weka 3: *Data Mining Software in Java* http://www.cs.waikato.ac.nz/ml/weka/

Williams T., Kelley C. (2004) Gnuplot. Retrieved from http://www.gnuplot.info/

Zaffalon M., Hutter M., (2002) Robust Feature Selection by Mutual Information Distributions, *Proceedings of the 18th International Conference on Uncertainty in Artificial Intelligence (UAI-2002)*

Zhang H., (2004) The Optimality of Naïve Bayes, *American association for Artificial Intelligence*

Appendix A

The clustering algorithms were built around a framework of the standard s-IB algorithm which is implemented using the idea proposed in Slonim et al. (2002). It chooses a word based on some predefined method (sequentially, randomly or repeatedly). It then computes the payoffs with respect to clusters i.e. the payoffs of moving the word to all clusters (obviously except the one which it currently belongs to). This is a very slow method when the number of clusters is increased as the overhead on the implementation of the algorithm increases. Hence a speedup method was proposed and implemented to ignore computing certain payoffs and thus shorten the time taken to process the program. In this section, we will briefly describe the time-stamp method which was attached to the existing s-IB algorithm to speed up the run time.

Time-stamp method for s-IB: In this method, every word-selection iteration is considered to be a time stamp for the words and clusters. Every word and cluster is associated with a time stamp. Initially, all timestamps are reset, i.e. set to 0. Consider the example in which the vocabulary size is 10 words and the number of predefined clusters is set to 10. These are randomly distributed among the 10 clusters.

During word-iteration 1, using random word-selection method, $Word_{23}$ is selected; it belongs to $Cluster_5$. We need to compute all payoffs here since it is the first time $Word_{23}$ has been selected. The computed payoffs are stored in *Payoffs* data structure. Continuing this we get to say word-iteration 100. During this time (word-iteration 2-99), $Cluster_5$ & $Cluster_9$ were not changed (no words were added or no words were taken out). Hence it is not necessary to compute this payoff. Now this may seem trivial for a small number of clusters and words. However, the real large scale experiments were conducted on a large vocabulary of words (usually greater than ten thousand words) and a large number of clusters (>100). As the word-iterations proceed, towards the end when the algorithm starts approaching its convergence point, the word

reallocations between clusters seem to stabilize and very few words are actually moved between clusters. This method comes in handy at this point where it is absolutely unnecessary to re-compute payoffs hence saving a lot of implementation time by speeding up the process.

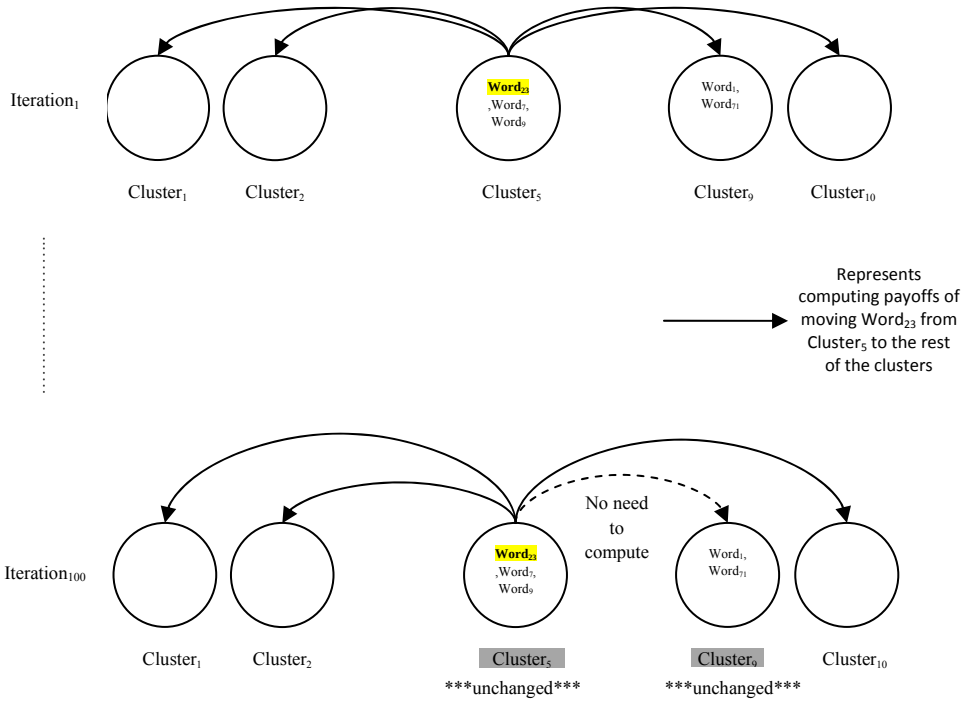

Figure A-1: Time-stamp extension to s-IB