Classification of Textual Documents Using Learning Vector Quantization

Submitted By

Muhammad Fahad Umer

Reg#107-CS/MS/03

Supervised By

Dr. M. Sikander Hayat Khiyal

Department of Computer Science
Faculty of Applied Sciences
International Islamic University, Islamabad

Final Approval

Date: 5th May, 2007

We hereby declare that we have read this thesis thoroughly and it is our judgment that this thesis is of sufficient standard to warrant it acceptance by the International Islamic University, Islamabad for the award of degree of Master of Science in Computer Science.

Committee

External Examiner
Mr. Shaftab Ahmad
Senior Principal Engineer
~~House No 460, Street No 59~~
~~G-11/2, Islamabad~~

Shaftab Ahmad

Internal Examiner
Dr. Syed Afaq Hussain
Head, Departrment of Electronic Engineering
International Islamic University, Islamabad

[illegible]

Supervisor
Dr. M. Sikander Hayat Khiyal
Head, Department of Computer Science
International Islamic University, Islamabad.

[illegible]

To my fellow traveler,

Dissertation

A dissertation submitted as a partial fulfillment for the award of degree of Master of Science (Computer Science)

Declaration

I hereby declare that this thesis is an original research work. No part of thesis has been copied from any source. I am solemnly responsible for the material presented and expressed in this document. I further guarantee that this work has not been presented in any other institution for award of degree or for any other purpose whatsoever.

Muhammad Fahad Umer
107-CS/MS/03

Project Summary

Project Title:	**Classification of Textual Documents using LVQ**
Undertaken By:	**Muhammad Fahad Umer**
Supervised By:	**Dr. M. Sikander Hayat Khiyal**
Tools Used:	**JDK 1.5** **JBulider 2005 Enterpsise**
Operating System Used:	**Microsoft Windows XP (R)**
System Used:	**Intel Pentium IV**
Date Started:	**20-04-2006**
Date Completed:	**05-05-2007**

ABSTRACT

The classification of a large collection of texts into predefined set of classes is an enduring research problem. The comparative study of classification algorithms shows that there is a tradeoff between accuracy and complexity of the classification systems. This work evaluates the Learning Vector Quantization (LVQ) network for classifying text documents. In the LVQ method, each class is described by a relatively small number of codebook vectors. These codebook vectors are placed in the feature space such that the decision boundaries are approximated by the nearest neighbor rule. The LVQ require less training examples and are much faster than other classification methods. The experimental results show that the Learning Vector Quantization approach outperforms the k-NN, Rocchio, NB and Decision Tree classifiers, and is comparable to SVMs.

Table of Contents

CHAPTER 1

INTRODUCTION

1 Introduction

The information on the Internet continues to grow at an incredible speed with more than 4.5 billion pages available online. The copious amount of data on the organizational intranets is besides this. Due to this amazing growth of Internet, several research areas have gained an invigorated interest. Text Classification (TC), the activity of labeling natural language text documents with thematic categories from a predefined set, is one such task that is an active area of research since the last decade.

The term classification has been used in a broader context in human activity. The term can be defined over any context in which some decision or forecast is made on the basis of currently available information, and a 'classification procedure' is some formal method for repeatedly making such judgments in new situations based on the information provided to it. We may have a set of observations and want to infer classes or clusters within the data. Or we may have a certain number of classes, and we want to classify a new sample in one of the existing classes. The former type is known as Clustering, and the latter is known as Classification.

1.1 Definition of Text Classification

The TC system categorizes the documents into a fixed number of predefined classes. Formally, it can be defined as the task of assigning a Boolean value to each pair (d_i, c_j) where $d_j=\{d_1,d_2,d_3,...d_n\}$ is the set of text documents and $c_i=\{c_1,c_2,c_3,...c_n\}$ is the set of class labels. The value assigned to the pair could be true if the document d_i falls under class c_i or false if the document d_i does not belong to class c_i [1].

The research in automated text classification started in early 1960s. A long list of successes and failures are reported in this field. Many methods had been proposed and a lot of experiments had been carried out. All this research had been done in the field of Information Retrieval and classification was a part of it. The rapid growth of Internet has revived the interest in automated text classification. Hand-built directories of web content suggest one solution to the dilemma, but unfortunately creating and maintaining such directories requires enormous amounts of human effort.

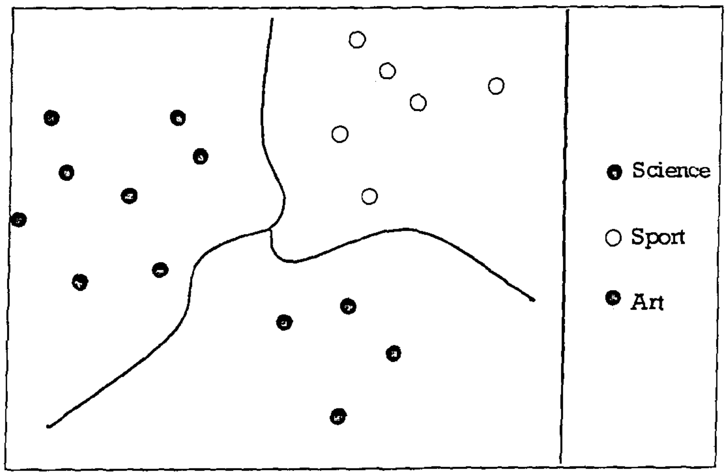


Figure 1-1 Classification Example

1.2 Application of Text Classification Systems

The applications of text classification system occur in wide range. The classification system may apply for Help-Desk support. The query obtained from the user may classify according to the subject and referred to an appropriate expert.

In newsgroup sites, it is sometime very cumbersome to find an article of interest. A text classification system can be employed to extract the relevant news stories. The popular benchmarking text classification data set, REUTERS, is indeed a collection of news stories from the Reuters Group.

Relevance feedback for the documents for a particular query can be calculated by using a classification system.

A large database of documents may be organized in semantic categories with the help of text classification system.

1.3 Issues in Text Classification

Text Classification systems generally have same structural design regardless of the algorithm used for the classification task. But there are some parameters, such as the soft or hard classification rule, single or multi-class classification results or the classification pivoting, which may differ according the problem domain. The reason for this diversity is the wide range of applications of classification systems.

1.3.1 Classification Rule

The procedure for text classification can be enforced to give a set of classes according to their relevance with the document. For a text document d_j ε D, the result could be a ranked list of classes C=$\{c_1, c_2, ... c_n\}$. The procedure doest not take any hard decision on any of the class. The ranked set of classes provided by the classification system can be of a great help to the human expert applying the classification. The classification rule is usually softened especially in critical application where the effectiveness of the text classification system is considered to be low than the human expert and an interactive session is maintained. This may be the case when the quality of the training data is low, or when the training documents cannot be trusted to be a representative sample of the unseen documents that are to come, so that the results of a completely automatic classifier could not be trusted completely [1].

1.3.2 Class Labeling

For every d_j, a single class label can be assigned to every document or a number of categories from C_1 to C_n may be assigned to the same d_j. The classification in which a single class label is assigned to every document is called single label or binary classification and in the later case the classification is called multi-label classification.

The binary case is more general than the multi label, since an algorithm for binary classification can also be used for multi label classification. The only requirement is to only transform the multi label classification problem to a single label classification problem. But the converse is not true, that an algorithm for multi label classification cannot be used for single label or binary classification. In fact, given a document *dj* to classify, the classifier might attribute $k > 1$ categories to *dj*, and it might not be obvious how to choose a "most appropriate" category from them; or the classifier might attribute to *dj* no category at all, and it might not be obvious how to choose a "least inappropriate" category from *C*. The binary case is important in itself because important classification applications, including filtering, consist of binary classification problems. Solving the binary case also means solving the multi label case, which is also representative of important text classification applications, including automated indexing for Boolean systems [1].

1.3.3 Classification Pivoting

There are two different ways of using a text classifier. Given $d_j \in D$, we might want to find all the classes to which it should belong (*document-pivoted categorization*); alternatively, given a class $c_i \in C$, we might want to find all the documents that should be filed under it (*category-pivoted categorization*). Document pivoted classification is suitable when documents become available at different moments in time, e.g., in filtering e-mail. Category pivoted classification is instead suitable when (i) a new category $c_{|c|+1}$ may be added to an existing set $C = \{c_1, \cdots, c_{|c|}\}$ after a number of documents have already been classified under *C*, and (ii) these documents need to be reconsidered for classification under $c_{|c|+1}$. Document pivoted classification is used more often than category pivoted classification, as the former situation is more common than the latter. Some specific problem domains apply to one technique and not to the other [1].

1.4 Text Classification Techniques

Three different fields have been in concern with the classification techniques; Statistics, Machine Learning and Artificial Neural Networks. The basic prototype methods for these techniques are linear discrimination, decision-tree and rule-based, k-nearest neighbor are prototypes for three types of classification procedure. Not surprisingly, they have been refined and extended, but they still represent the major strands in current classification practice and research. The procedures mainly used for classification can be directly linked to one or other of the above. However, within literature, the methods have been grouped around the more traditional headings of classical statistics, modern statistical techniques, Machine Learning and neural networks [2].

1.4.1 Classical Statistics

We can include in this group those procedures that start from linear combinations of the measurements, even if these combinations are subsequently subjected to some non-linear transformation. The procedures of this type are: Linear discriminants; logistic discriminants; quadratic discriminants. In these methods, the training set is a subset of n known class example. The n is typically 2. These methods require numeric value attributes with none of the value is missing. The attributes used binary indicators to indicate that a specific attribute belong to a class or not. When an attribute is classified against more than one class, the indicators are setup to drop the other classes.

1.4.2 Modern Statistics

The modern statistics includes the techniques that use density estimation to approximate the classification categories. The k-nearest neighbor, Projection pursuit classification, Casual networks and Naive Bayes are considered under modern statistical classification techniques.

1.4.3 Decision Trees

Decision tree learning is one of the most widely used techniques for classification. Its classification accuracy is competitive with other methods, and it is very efficient. The classification model is a tree, called decision tree in which internal nodes are labeled by terms, branches departing from them are labeled by tests on the weight that the term has in the test document, and leafs are labeled by categories [3].

The most popular approaches used for constructing Decision Tree classifiers are ID3, C4.5, and C5.

1.4.4 Neural Networks Techniques

The neural network techniques have a common procedure that is intimately linked with the training of the network and adjusting the input weights. The density estimate group contains: radial basis functions; Kohonen self-organizing maps; LVQ; and the kernel density method.

1.5 Artificial Neural Networks

Artificial Neural Networks are electronic models based on the neural structure of the brain. The brain basically learns from experience and neural networks try to mimic the same behavior. This new approach to computing also provides a more graceful degradation during system overload than its more traditional counterparts. Even simple animal brains are capable of functions that are currently impossible for traditional computing; these biologically inspired methods of computing are thought to be the next major advancement in computer science. But the true power of neural networks have not utilized till now. It is worth mentioning that there is very small numbers of real time neural network based applications. The reason for this is not that there is a problem with the neural network theory, but it is the inability to model the billions of brain neurons with the traditional computers.

Computers do simple mathematical things well, like keeping information stored for a long time or performing complex calculations. But computers have trouble recognizing even simple patterns and abstracting the real world ideas.

Now, advances in biological research promise an initial understanding of the natural thinking mechanism. This research shows that brains store information as patterns. Some of these patterns are very complicated and allow us the ability to recognize individual faces from many different angles. This process of storing information as patterns, utilizing those patterns, and then solving problems encompasses a new field in computing. This field, as mentioned before, does not utilize traditional programming but involves the creation of massively parallel networks and the training of those networks to solve specific problems. This field also utilizes words very different from traditional computing, words like behave, react, self-organize, learn, generalize, and forget [4].

1.5.1 Analogy to the Brain

The exact workings of the human brain are still an obscurity. Yet, some aspects of this astounding processor are known. The most basic element of the human brain is a specific type of cell, which, unlike the rest of the body, doesn't appear to regenerate. Because this type of cell is the only part of the body that isn't slowly replaced, it is assumed that these cells are what provide us with our abilities to remember, think, and apply previous experiences to our every action. These cells, all 100 billion of them, are known as neurons. Each of these neurons can connect with up to 200,000 other neurons, although 1,000 to 10,000 are typical. The power of the human mind comes from the sheer numbers of these basic components and the multiple connections between them. It also comes from genetic programming and learning. The individual neurons are complicated. They have a myriad of parts, sub-systems, and control mechanisms. They convey information via a host of electrochemical pathways. There are over one hundred different classes of neurons, depending on the classification method used. Together these neurons and their connections form a process which is not binary, not stable, and not synchronous. In short, it is nothing like the currently available electronic computers, or even artificial neural networks. These artificial neural networks try to replicate only the most basic elements of this complicated, versatile, and powerful organism. They do it in a primitive way. But for the software engineer who is trying to solve problems, neural computing was never about replicating human brains. It is about machines and a new way to solve problems.

1.6 Classification using ANN

A neural network consist of one or more than neurons connected which each other and having some weights associated with each weight. The simples case is that network will consist of single layer with only one neuron as given in the following figure [5].

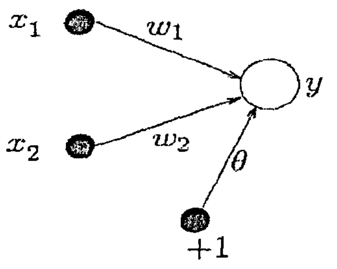


Figure 1-2 A simple neuron model

The output of the network is formed by the activation of the output of neuron, which is some function of input:

$$y = F\left(\sum_{i=1}^{2} w_i x_i + \theta\right) \tag{1.1}$$

The activation function can be linear if we have a linear network. The output function could be:

$$F(s) = \begin{cases} 1 & if\ s > 0 \\ -1 & otherwise. \end{cases} \tag{1.2}$$

The output could be +1 or -1 depending on the input so the network can be used for the classification task. A separation between the two classes is the straight line given by the equation:

$$w_1 x_1 + w_2 x_2 + \theta = 0 \tag{1.3}$$

A geometric representation of the linear threshold neural network is given Fig 1.1 and the Equation (1.2) can be written as:

$$x_2 = -\frac{w_1 x_1}{w_2} - \frac{\theta}{w_2} \tag{1.4}$$

The weights determine the slope of the line and the bias determines the offset i.e. how far is the line from the origin. The weight vector is always perpendicular to the input space [5].

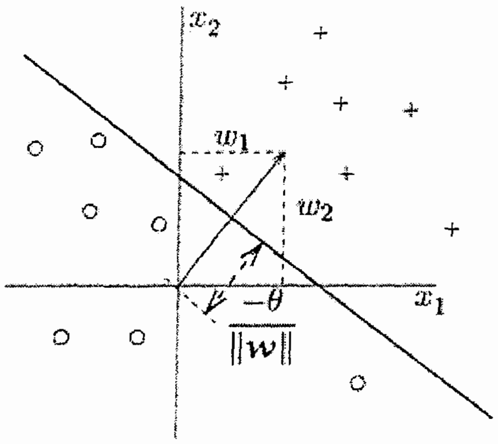


Figure 1-3 Perceptron Classification

This single layer network learns the weight and bias by using a perceptron learning rule. There are a number of learning rules defined for neural networks such as Error-correction learning rule, Perceptron learning rule, the Boltzmann learning rule, Hebbian learning rule and the competitive learning rule. The perceptron learning rule is the iterative procedure that adjusts the weights. A learning sample is presented to the network. For each weight the new value is computing by adding a correction to the old value.

A single layer network has several limitations and the class of tasks that can be accomplished is very limited. For solving complicated problems we need more complex multi-layer neural networks such as feed-forward networks, recurrent networks and self-organizing networks. A feed-forward network has a layered structure. Each layer consists of a number of units, which receive their input directly below and send their output to the layer directly above. There is no connection between the neurons in the same layer. The recurrent networks contain cycles within the layers. A hidden unit may be connected with itself over a weighted connection, connect hidden units with input units or all units may be connected with each other. The self-organizing networks do not require a sample set for training and use unsupervised learning rule. The most basic scheme for the self-organizing networks is competitive learning in which each unit competes for winning.

1.7 Learning

There are two models in artificial neural networks:

1. Activation transfer mode when activation is transmitted throughout the network
2. Learning mode when the network organizes usually on the basis of most recent activation transfer

Neural networks need not to be programmed when they encounter novel environment. Yet their behavior changes in order adapt to the new environment. Such behavioral changes are due to the changes in the weights of the networks. These changes in the weighs are called learning. The changes in the neural network are intended to model the changing synaptic efficiencies in real neural networks. There are three main types of learning for the neural networks.

1.7.1 Supervised learning

With this type of learning the network is provided with the input data with the correct answers i.e. output that is intended to receive form the network from the network. The input data is propagated forward through the network till activation reaches to the output neurons. The answers that are calculated from the network can be compared from the desired output. If the answers agree, no change is made to the network, however f the answers are different, and the weights are adjusted to ensure that the network more likely to give the correct result in future if it is presented with the same input data. This weight adjustment schemes is known as supervised learning or learning with a teacher. The delta learning rule and the LMS rule are examples of supervised learning.

1.7.2 Unsupervised learning

In this type of learning the network is only provided with input data. The network is required to self organize depending on some structure in the input data. Typically this structure is may be some form of redundancy in the input data or clusters in the data. Self Organizing Maps use unsupervised learning rule.

1.7.3 Reinforcement learning

This type of learning is halfway house between the above two types of learning. We provide the network with the input data and the activation is propagated but only to tell the network that it has produced correct result or not [4].

1.8 LVQ Networks

Learning Vector Quantization networks are based on the supervised competitive learning. LVQ networks attempt to define decision boundaries in the input space, given a large set of exemplary decisions (the training data). Topologically, the network contains an input layer, a competitive layer and an output layer. The output layer has the neurons equal to the number of classes. The competitive layer has a number of neurons assigned to each class. The competitive layer learns and performs relational classifications with the aid of a training set. Unlike perceptron, LVQ networks can classify any set of input vectors, not just linearly separable sets of input vectors. The only requirement is that the competitive layer must have enough neurons, and each class must be assigned enough competitive neurons [4].

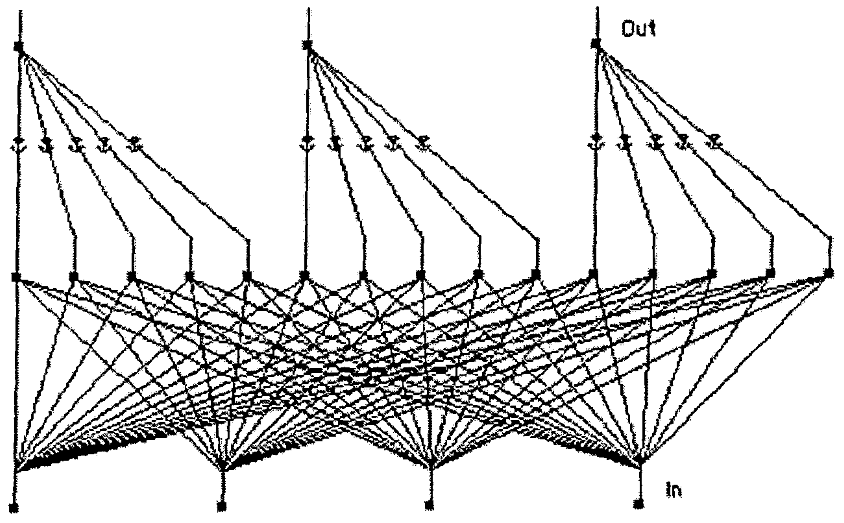


Figure 1-4 LVQ Network model

1.9 Learning Vector Quantization Algorithms

The LVQ method assume that a number of codebook vectors m_i are used to classify various domains of input vector x, and x is then decided to belong to the same class to which the nearest m_i belongs. Let

$$c = \arg\min_i \{\|x - m_i\|\} \tag{1.5}$$

Let $x(t)$ be a sample input and let $m_i(t)$ represent the sequence of m_i in the discrete tome domain. Then starting with the initial values of m_i, the following equations define the basic Learning Vector Quantization process:

$$m_c(t+1) = m_c(t) + \alpha(t)[x(t) - m_c(t)] \qquad (1.6)$$

if x and m_c belong to the same class

$$m_c(t+1) = m_c(t) - \alpha(t)[x(t) - m_c(t)] \qquad (1.7)$$

if x and m_c belong to the different class

The *a(t)* defines the learning rate for the learning process. This algorithm is called ***LVQ1*** and it is the basic LVQ learning algorithm. There are many other variations of the LVQ method. The ***LVQ2*** algorithm takes two codebook vectors m_i and m_j, which are the two nearest neighbors to x, are updated simultaneously. One of them belongs to the correct class and the other to a wrong class respectively. Moreover x must fall into a zone of values called 'window' which is defined in between the mid-plane of m_i and m_j. A relative window length of 0.2 to 0.3 is recommended.

The algorithm process is given by the following equations:

$$m_i(t+1) = m_i(t) - \alpha(t)[x(t) - mi(t)] \qquad (1.8)$$

$$m_j(t+1) = m_j(t) + \alpha(t)[x(t) - mj(t)] \qquad (1.9)$$

where m_i and m_j are the two closest codebook vectors to x, where x and m_i belong to the same class while x and m_i belong to the different classes. The LVQ2 algorithm does not give attention to m_i that what will happen to the location of the m_i in the long run. The ***LVQ3*** algorithm includes necessary corrections that ensure that the m_i continue approximation class distributions roughly. In addition to the previous equations defined for the m_i and m_j, x and m_i belong to different classes, respectively; furthermore x must fall into the 'widow';

$$m_k(t+1) = m_k(t) + \varepsilon\alpha(t)[x(t) - m_k(t)] \qquad (1.10)$$

for k Є $\{i, j\}$, if x, m_i, and m_j belong to the same class. The optimal value of Є seems to depend on the size of the window, being smaller for narrower windows. The value between 0.1 and 0.5 is recommended for Є. This algorithm seems to be self stabilizing, i.e.; the optional placement of the m_i does not change in continual learning.

The basic LVQ1 and LVQ3 algorithms are modified in s such a way that an individual learning rate $a_i(t)$ is assigned to each m_i. The optimal learning rate can be defined as:

$$\alpha_c(t) = \frac{\alpha_c(t-1)}{1 + s(t)\alpha(t-1)} \qquad (1.11)$$

Where *s(t)* =1, id the classification is correct and *s(t)* = -1 if the classification decision is wrong. The LVQ1 and LVQ3 with individual training rate are called ***optimized LVQ1 and LVQ3*** algorithms because the learning rate is optimized [6].

The different training algorithms available yield almost similar accuracies, although different techniques underlie each other. In this work, we will perform the experiment using all available algorithms and analyze the result to determine that which algorithm performs well for text classification. In the next step we will compare the LVQ algorithms results with the other text classification algorithms for determining performance of LVQ algorithms.

CHAPTER 2
LITERATURE SURVEY

2 Literature Survey

The older work done in classification mainly relates to Statistics. The classification work in statistics can be identified in two phases. The first or the classical phase is the Fisher's work on linear discrimination. The second or modern phase exploits more flexible classes of models, which attempt to provide an estimate of the joint distribution of the features in every class. Statistical approaches used in text categorizations problems generally have an underlying probability model, which provides a probability of a test example fro being in each class rather than simply a classification.

Machine Learning encompasses automatic computing procedures based on logical or binary operations that learn a task from a series of examples. In the ML approach, the pre-classified documents are the key resource. In an ideal case, they are already available, typically for the problems that are solved manually. The less favorable case is when no manually classified examples are available, this happens for a newborn problem. The ML approach is convenient also in the latter case. Classifiers built by means of ML techniques nowadays achieve impressive levels of effectiveness making automatic classification a qualitatively viable alternative to manual classification [1]. In Machine Learning, attention has focused on decision-tree approaches, in which classification results from a sequence of logical steps. These are capable of representing the most complex problem given sufficient data.

Another field of concern for text classification is the Neural Networks. Generally, neural networks consist of layers of interconnected nodes called neurons, each neuron produce a non-linear function of its input. The input to a neuron may come from other neurons or directly from the input data. The neurons in the output layer define the output of the network. The complete network therefore represents a very complex set of inter-dependencies, which may incorporate any degree of nonlinearity, allowing very complex functions to be modeled. The simplest neural network receives input from one neuron and forwards it to the next neuron in the way that the results are propagated through the network. These networks are called feed-forwarded networks. In more complex networks the results are propagated backward to improve the efficiency and to minimize the error. These networks are called Recurrent or Feedback Networks. It has been argued that neural networks emulate to a certain level the behavior of networks of neurons in the brain.

The above discussion involves broad fields of concern with text classification. The classification solutions suggested in the literature belong to one or more of the above described fields; for example, Naïve Bayes (NB) probabilistic classifiers [7], Decision Tree classifiers [8], *k*-NN classifiers [9], Support Vector Machine (SVM) [10] and [11], and Rocchio classifiers [12] and [13] etc. Most of the methods come from the Machine Learning approach, while Neural Networks have also been studied extensively for the application of text classification.

2.1 Rocchio's Classification

Rocchio's classification algorithm [12] is used for inducing linear, profile-style classifiers. The Rocchio method operates by building a prototype vectors for every class. The prototype vector is the profile of a category is the difference between the *centriod* of the positive and the negative examples.

$$C_i = \beta \times centroid(POS_i) - \gamma \times centroid(NEG_i) \tag{2.1}$$

The ß and γ are the control parameters. If we take ß=1 and γ=0, then the profile of the class becomes the centriod of the positive examples. The role of the negative examples is denigrated by using higher value for ß and lower value for γ. [13] used ß=16 and γ=4. The prediction rule used to compute the classification of a new example is the cosine of the new example with the profile vector. For a new example $\vec{x}$,

$$p(\vec{x}) = \begin{cases} 1, & if \cos(x, Ci) > \theta \\ -1, & otherwise \end{cases} \tag{2.2}$$

An improvement to the classic Rocchio method is the use of near-positive examples employed by [14]. The near-positive examples are the most positive examples in the set of negative examples and are the most difficult documents to apart from the positive examples

This method is quite efficient and easy to implement. A drawback of Rocchio method described by [1] is that if the documents in the category tend to occur in disjoint clusters (e.g., a set of newspaper articles labeled with the Sports category and dealing with either boxing or rock-climbing), such a classifier may miss most of them, as the centriod of these documents may fall outside all of these clusters. The improved versions of Rocchio based classifiers are discussed [15] and [13].

2.2 Naïve Bayes Classifiers

These classifiers work a generative framework in which each document is generated by a parametric distribution governed by a set of hidden parameters. Naïve Bayes method assumes that all attributes of the examples independent of each other given the context of a single class while this assumption is clearly wring in the real world, the Naïve Bayes often works well. Because of the independence assumption the parameters for every attribute can be learned separately. This makes the learning process very simple especially when the attributes are large.

This model views the training examples as $\Pr(Y \mid X = \vec{x})$ that is, the probability that a document represented by a vector x belongs to Y, and compute this probability by an application of Bayes' theorem given by:

$$\Pr(Y \mid X = \vec{x}) = \frac{\Pr(Y)\Pr(X = \vec{x} \mid Y)}{\Pr(X = \vec{x})} \tag{2.3}$$

The classification rule can be defined as that predicts class 1 if the following predicate is true or the Class -1 otherwise.

$$\Pr(Y = 1)\prod_{i=1}^{lx} \Pr(w = w_i \mid Y = 1) > \Pr(Y = -1)\prod_{i=1}^{lx} \Pr(W = w_i \mid Y = -1) \quad (2.4)$$

$\Pr(X = \vec{x})$ is thus the probability that a randomly picked document has vector x as its representation, and $\Pr(Y)$ is the probability that a randomly picked document belongs to Y. The multinomial Naive Bayes model assumes that any two coordinates of the document vector are, when viewed as random variables, statistically independent of each other; this independence assumption is encoded by the equation

$$\Pr(X = \vec{x} \mid Y) = \prod_{i=1}^{lx} \Pr(W = w_i \mid Y) \quad (2.5)$$

There are many variations to the Naive Bayes approach suggested by [7] also discussed by [1]. A variation is to use weighted indexed vectors instead of binary-valued vectors [16]. Another way of improvement is to introduce document length normalization. The value of document ranking can be very high or very low for long documents. Taking length into account is easy in non-probabilistic approaches to classification, but is problematic in probabilistic ones. One possible answer is to switch an interpretation of Naive Bayes in which documents are events to one in which terms are events, but at the same time, the solution given above has the drawback that the different occurrences of the same word within the same document are viewed as independent.

2.3 Decision Tree Classification

Decision tree learning is one of the most widely used techniques for classification. Its classification accuracy is competitive with other methods, and it is very efficient. The classification model is a tree, called decision tree in which internal nodes are labeled by terms, branches departing from them are labeled by tests on the weight that the term has in the test document, and leafs are labeled by categories [3].

A decision tree based classification learning process consists of two steps. In the first step of tree induction, a tree is induced from the given training set for category C_i using "divide and conquer" strategy by checking whether all the training examples have the same label (either C_i or $\bar{C}_i$) and if not, a term t_k is selected to partition the training set into classes of documents that have the same value for t_k, and placing each such class in a separate subtree. The process is recursively repeated on the subtrees until each leaf of the tree generated contains training examples assigned to the same category C_i, which is then chosen as the label for the leaf. The key step is the choice of the term t_k on which the partition is performed. In the second step of tree pruning, the induced tree is made more concise and robust by removing any statistical dependencies on the specific training dataset. The induction step is computationally much more expensive as compared to the pruning step.

The most popular approaches used for constructing Decision Tree classifiers are ID3 [17], C4.5 [18], and C5 [19].

2.4 Associate Rule Mining

The Association Rules are canonical data mining taking aim at discovering relationships between items in the dataset. The association-rule-based classifiers model the text documents as a collection of transactions where each transaction represents a document, and the items in the transaction are the terms selected from the document and the categories documents is assigned to. The most popular algorithm use to compute association rules effectively are apriori algorithm [20] and FP-tree algorithm [21]. The [22] uses a similar approach to construct a rule-base classifier with apriori-based algorithm but the results obtained on the Reuter-21578 collection are not promising as for five categories out of ten; the precision/recall breakeven point is around 60 %. For a relatively difficult category the breakeven point is 25.8 %, which is not acceptable for practical classification.

2.5 Support Vector Machines

SVM is learning methods introduced by [23]. SVM are based on the structural risk minimization principal from the computational theory. SVM use the Vapnik-Chervonenkis (VC) dimensions of a problem to characterize its complexity, which can be independent of the dimensionality of the problem. The basic idea is to find decision surfaces between use a hyper plane to the classes of data, positive and negative with maximum margins from the both sides. Kernel functions are used for nonlinear separation. The groups of vectors that lie near the separating hyperplane are called support vectors. Once the separating hyper plane is found the new examples can be classifies by simply checking that on which side of the hyperplane they fall. SVM not only has a rigorous theoretical foundation, but also performs classification more accurately than most other methods in applications, especially for high dimensional data. In classifiers using SVM, term selection is often not needed, as SVMs tend to be fairly robust to over fitting and can scale up to considerable dimensionalities. Also there is no human and machine effort in parameter tuning on a validation set is needed, as there is a theoretically calculated "default" choice of parameter settings.

SVM have shown superb performance for text classification tasks. The reasons that SVMs work well for TC is that during learning classifiers, one has to deal with many features such as more than 10,000. Since SVM use over fitting protection that does not depend on the number of features and have the potential to deal with the large number of attributes. Most of the document vectors are sparse and contained very few non-zero entries. It is shown in [24] that additive algorithms having inductive base like SVM work very well for problems with dense concepts and sparse instances.

Most of the text categorization problems are linearly separable such as the reuters-21578 [26].

SVMs are accurate, robust, and quick to apply to test instances. Their only potential drawback is their training time and memory requirement. For *n* training instances held in memory, the best-known SVM implementations take time proportional to n^a, where a is typically between 1.8 and 2.1.

2.6 Bagging Algorithms

Since most of classification algorithm work poorly, voting algorithms make use of them in smarter way i.e. many poor classifiers => one good classifier. Some base classifiers are generated and they are used to give decision about the classification of a document. In order to guarantee good effectiveness, the classifiers should be independent from each other as possible [25]. However the classifier may use the same or different indexing approach or the induction method.

For constructing the classification decision, the simplest rule used is the majority voting. For k classifiers the decision which takes $(k+1)/2$ votes are taken [19].

A variant of classifier's committee, called *boosting* method is also used in the classification applications [15]. The main idea of this algorithm is to maintain a distribution or set of weights over the training set. Initially, all weights are set equally, but in every iteration, the weights of incorrectly classified examples are increased so that the base classifier is forced to focus on the 'hard' examples in the training set. For those correctly classified examples, their weights are decreased so that they are less important in next iteration.

One example of the voting algorithm is the bagging algorithm. In bagging algorithm, multiple versions of a training set D of size N, each created by re-sampling N examples from the data set are taken. Each of training sets is used to train a base classifier; the majority voting of these classifiers makes the final classification decision. Although voting algorithms give relatively high accuracy rate but they need extensive calculation and memory as there is more than one type of classifiers are working.

2.7 K-Nearest Neighbor Classification

Unlike all the other classifications methods, k-NN [9] does not build model from the training data but rely on the category labels attached to the training documents similar to the test document. These methods are called lazy learners, since "they defer the decision on how to generalize beyond the training data until each new query instance is encountered"[1].

To classify a test instance d, it defines k-neighborhood P as k nearest neighbors of d and count number n of training instances in P that belong to class C_j. No training is needed. Classification time is linear in training set size for each test case. The problem involved in using KNN classifiers is to determine the optimal value of k. The value of k is a tradeoff between the accuracy and the classification time. Large value of k will generate high accuracy but the classification time will be very slow.

The value for k is usually chosen empirically via a validation set or cross-validation by trying a range of k values.

k-NN is considered a lazy learning algorithm as it defers data processing until it receives a request to classify an unlabelled example. It replies to a request for information by combining its stored training data. After the classification decision it discards the intermediate results and the constructed answer. This strategy is opposed to other learning algorithms where the data model is described in the form of a density estimator or a graphical structure with weights.

The k-NN can deal with complex and arbitrary decision boundaries. Despite its simplicity, researchers have shown that the classification accuracy of k-NN can be quite strong and in many cases as accurate as those elaborated methods. The k-NN method is slow at the classification time and also does not produce an understandable model.

CHAPTER 3

PROBLEM DOMAIN

3 Problem Domain

The problem discussed here is the construction of a procedure that will be applied to a collection of documents. Each new document of the collection is assigned a predefined class label on the basis of some observed attributes and features.

There are some issues regarding classification presented in [2] that are to be discussed. Most of the classifiers discussed in previous section lack one or more of these requirements.

3.1 Requirements of a Text Classification System

The requirements of a text classification system are described as follows:

Accuracy: There is the reliability of the rule, usually represented by the proportion of correct classifications. If the classification is performed by an intelligent procedure (e.g. human), and the results are compared then sufficient accuracy should be present in the results.

Speed: In some circumstances, the speed of the classifier is a major issue. A classifier that is 90% accurate may be preferred over one that is 95% accurate if it is 100 times faster in testing (and such differences in time-scales are not uncommon in neural networks for example). Such considerations would be important for automatic reading of postal codes or automatic fault detection of items on a production line for example.

Comprehensibility: If it is a human operator that who applys the classification procedure, the procedure must be easily understood else mistakes will be made in applying the rule. It is important also, that human operators believe the system. An oft-quoted example is the Three-Mile Island case, where the automatic devices correctly recommended a shutdown, but the human operators who did not believe that the recommendation was well founded did not act upon this recommendation. A similar story applies to the Chernobyl disaster.

Time to Learn: Especially in a rapidly changing environment, it may be necessary to learn a classification rule quickly, or making adjustments to an existing rule in real time. "Quickly" might imply also that we need only a small number of observations to establish our rule. Statistical approaches completely lack this requirement.

While there are still lacks of classification methods that fulfill the above-mentioned requirements; it will be quite reasonable to find an adequate solution to solve the problem of document classification. The basis of this research can be stated as below:

"To find a reasonable accurate, efficient, and flexible solution for automatic classification of text documents"

3.2 Scope of Work

Neural Networks is the emerging field of today and it provides a mechanism to define a solution for the above quoted research problem. If designed with care, neural networks perform very well as measured by error rate. They seem to provide either the best or near to best predictive performance in nearly all cases. In terms of computational burden, and the level of expertise required, they are little complex than, say, the machine learning procedures, but with an exception to Learning Vector Quantization (LVQ) which is easy to set up and fast to run [2].

Neural Networks provide sufficient accuracy rate, they can learn new situations, and are quite fast (especially in case of LVQ). The most important factor is that neural networks roughly mimic human behavior that is the essence of intelligent text classification.

This work would discuss an application of a specific type of neural network called Learning Vector Quantization for the text classification problem.

3.3 Proposed Solution

The experiment will be conducted by selecting a data set of random text documents. A neural network based on Learning Vector Quantization will be designed. This network will be trained using a subset of the dataset available for classification. There will be pre-defined classification classes and every element of dataset will belong to one of the pre-defined classes. The research can be divided in three phases, representation of text documents, designing and training of the network, providing real data and the analysis of results.

3.3.1 Representation of Documents

The documents will be represented as two-dimensional matrix using Vector Space Information model. The document set comprises an $m \times n$ term-document matrix in which a column A_j will represent a document and the cell A_{ij} will represent the frequency of a particular term present in the document. A major benefit of this approach is that algebraic structure of the vector space can be exploited. The conversion of documents from text to matrix form is standardized as follows:

1. A list of unique strings will be created from the documents selected for training.
2. The list can be scanned for deleting common words using a stop list. A stemming algorithm such as Porter's stemming algorithm can be also applied for removing suffixes from different form of a single word.
3. Third, the document collection can be indexed on the basis of scanned list using Vector Space Information model. Different weighting schemes might be used; one that seems effective is the "Term Frequency-Inverse Document Frequency" (TF-IDF), that is, the number of times the word appears in the document multiplied by a function of the inverse of the number of documents in which the word appears. Terms that appear often in a document and do not appear in many documents therefore have an important weight.

3.3.2 Training of Network

In the training phase, the codebook vectors are initialized from the given training data. Each codebook vector must fall within the correct class boundary for which it is initialized. A sample is classified against all other samples in the training set and is accepted only if it has the same classification as the initial class label given. There are different learning algorithms available for the training of the network discussed in Chapter 1.

3.3.3 Analysis of Results

After the training of the network, the real data can be provided to the network and the results can be analyzed using standard evaluation measures such as F1-mesure to judge the performance of the classifier.

The F1-measure is the harmonic mean of the precision and recall of the classifier.

$$F1 = \frac{2pr}{p + r} \tag{3.1}$$

where p= Precision, and r = Recall.

The Precision & Recall measures are widely used in Information Retrieval and Text classification. Precision is defined as the number of correctly classified positive examples divided by the total number of examples that are classified as positive. Recall is defined as the number of correctly classified positive examples divided by the total number of actual positive examples in the test set. The class of interest is called the positive class, while the rest of all are negative classes

CHAPTER 4
SYSTEM DESIGN

4 System Design

The structural design of the text classification system with training and testing phase is described in Fig 4-1. The system operates in two modes; training mode and testing mode. In training mode the network developed pattern for classification of text documents. The inputs to training mode are the documents with pre-defined documents while inputs to testing mode are unclassified documents. The pre-processing phase is similar for both modes with exception that the dictionary is construction in training mode and it is used for document conversion during testing mode. The training data is used to construct classifier and it takes classification decision for unclassified documents.

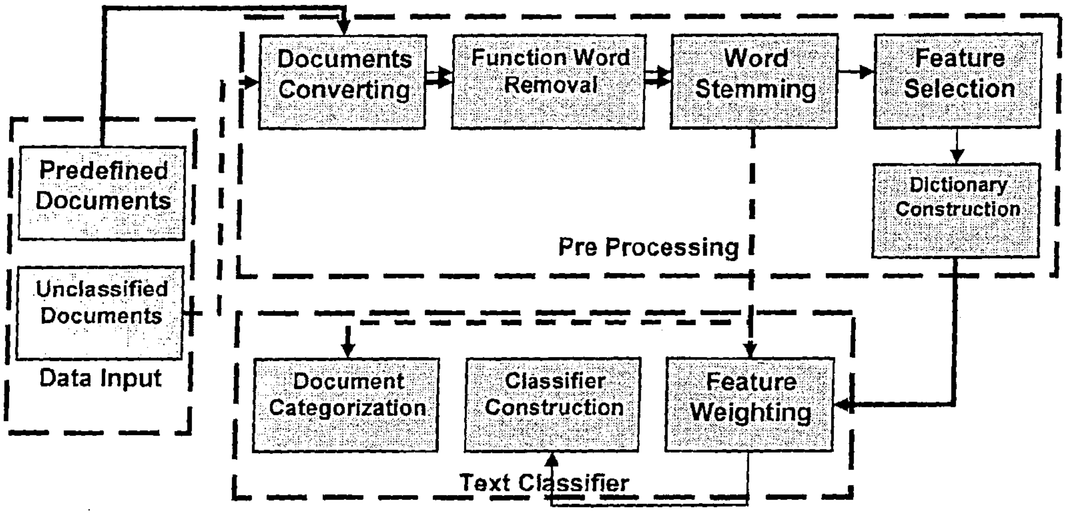


Figure 4-1 Classification System

4.1 Data Input

The data for the classification system will consist of the text documents. Text classification system takes both classified and unclassified documents as input. The classified documents have class label attached with them and used for the training of classification procedure.

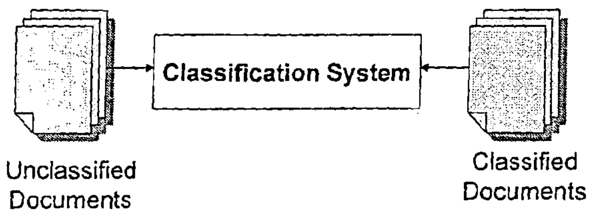


Figure 4-2 Classification Flow

The system will be able to extract text from XML and HTML formatted text pages and form Internet URLs.

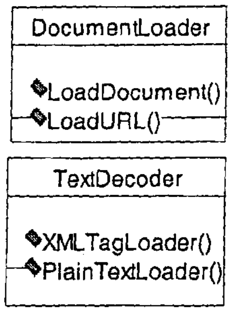


Figure 4-3 The Document Loader Classes

4.1.1 DocumentLoader Class

The DocumentLoader class will load document from a single file, disk directory, or from the URL.

4.1.2 TextDecoder Class

The TextDecoder class will parse documents into raw text. The functions defined for this class are

a. XMLTagLoader will ignore XML or HTML formatted tags and will extract actual text.

b. The PlainTextLoader() method will be used for simple text documents.

4.2 Data Pre-processing

Generally the classification algorithm cannot operate on simple raw texts. Several pre-processing steps are required for classification of texts using Learning Vector Quantization. This preprocessing increases the accuracy of classification and reduces the complexity of the procedure by decreasing the size of dictionary. The steps for pre-processing are described below:

4.2.1 Common Words Removal

The removal of high frequency words prevents document vectors from becoming sparse. Comparing input text with a stop list of words can easily perform it.

The removal of least occurring terms helps in making vocabulary list more meaningful. The terms having frequency of less than 2 or 3 do not count much towards calculating the relevancy of document to a class. So, it is good practice to remove them to reduce vocabulary size and computational complexity [27].

4.2.1.1 StopWord Class

The StopWord class has a list of stop word. Each input token is compared with stop list and is not processed if a match is found. The function `isStopWord(String token)` will return true or false for a string token extracted form text documents depending if the match is found or not.

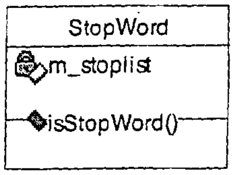


Figure 4-4 Stopword Class

4.2.2 Word Stemming

After the removal of high frequency terms, the next step is of suffix stripping from words having same meaning but more than morphological form. Two words should be compared for their conceptual meaning because a conventional string comparison will produce high error rate.

Table lists some examples where words should be taken as equivalent but traditional string comparison will show them different. The solution to this problem is to remove suffixes from different forms of a word. Removal of ility, ual, es, en from the following words will work out. There are standard algorithms defined for suffix stripping called stemming algorithms such as Lovins and Porter algorithms [28].

Table 4-1 Suffix Stripping

Words		String Comparison	Meaning
Flexible	Flexibility	Different	Same
Fact	Factual	Different	Same
Matrix	Matrixes	Different	Same
Give	Given	Different	Same

Many words, after suffix removal map to one morphological form, but still there are some, which don't. One way to deal with this problem is to have a list of equivalent stems and two words should be considered equivalent if and only if their stems match and there is an entry in the list defining their suffixes as equivalent. This work does not consider the problem of equivalency of stems as there are very few such words in a document and usually don't affect the accuracy of the classifier.

4.2.2.1 Lovins Stemmer Wrapper Class

This class wraps the Lovins stemmer algorithm for determining the root of a word.

4.2.2.2 Lowercase Stemmer Wrapper Class

The Lowercase Stemmer class does not use any stemming algorithm but it simply converts the all input tokens to lower case and eliminates the duplicate term.

4.2.3 Dictionary Creation

After the removal of high frequency words and word stemming, a dictionary of important words of document space is created. All documents are indexed on the basis of this dictionary using a suitable weighting scheme.

4.2.3.1 WordList Class

The WordList class has a list of important terms. We can also manually add and delete the words from the word list.

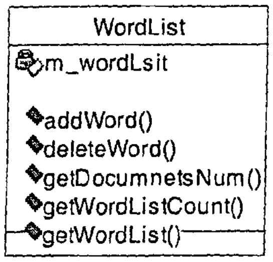


Figure 4-5 WordList Class

4.2.4 Document Conversion

Generally the classification algorithm cannot operate on the simple raw texts. A structure representation of the text documents is required. The representation used is the Vector Space Information Model. This model views the documents as vectors of words. Each cell of the document vector shows the weighted frequency of that term in that document. There are different schemes for the term weighting:

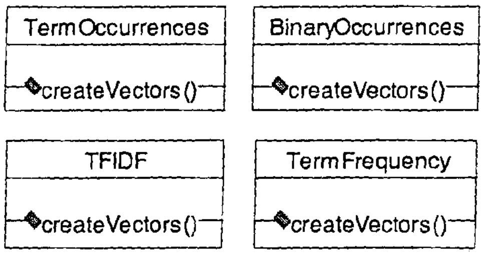


Figure 4-6 Weighting Scheme Classes

4.2.4.1 BinaryOccurences Class

The cells of the document vectors have 0 or 1 depending on the presence of a term in that document. For the ith cell of the jth document in documents space v, the corresponding value will be

$$V_{ij} = \begin{cases} 1, f_{ij} > 0 \\ 0 \end{cases} \tag{4.1}$$

where f_{ij} is the frequency of ith term in the jth document.

4.2.4.2 TermOccurences Class

The absolute number of occurrences of a term $v_{ij} = f_{ij}$ is used.

4.2.4.3 TermFrequency Class

Each cell of the vectors has relative frequency of a term in that document. The frequency of each term is normalized to the Euclidean unit length by dividing the each term frequency with the total number of terms in the document vector.

$$V_{ij} = \frac{f_{ij}}{f_{dj}} \quad (4.2)$$

where f_{ij} is the frequency of ith term in the jth document, and f_{dj} is the total number of terms in the document j.

4.2.4.4 TFIDF Class

The calculation of TF-IDF is done by multiplying the term frequency with an inverse function of document frequency of the term. The weight value for the term can be defined as:

$$V_{ij} = \frac{f_{ij}}{fd_j} \log\left(\frac{|D|}{ft_i}\right) \quad (4.3)$$

where,

$$tf_{ij} = \frac{f_{ij}}{fd_j} \quad (4.4)$$

The longer the document, the more likely it is for a given term to appear in it. It would be better to reduce the importance attached to a term appearing in a document based on the length of the document. The term weights are than normalized so longer documents are not unfairly given more weight.

$$V_{ij} = \frac{tf_{ij} \log(D / ft_i)}{\sqrt{\sum_{i=1}^{t} (tf_{ij})^2 [\log(D / ft_i)]^2}} \quad (4.5)$$

4.2.5 The Configuration Class

This class will include variables for configuring each step of the vector creation. For every step in the vectorization process, user sets the class that would be used for this step. This class can be one already included in the package or a new one, written by the user. The only constraint is that it has to implement the corresponding interface of a given step.

- STEP_CHAR_MAPPER: If a different character mapping scheme is used other than default
- STEP_LOADER: The loader for loading the documents i.e. XML documents or simple text documents
- STEP_OUTPUT: If the vectors are stored in a file then the output configuration
- STEP_STEMMER: The stemming algorithm wrapper class that will be used for suffix stripping.
- STEP_TOKENIZER: The tokenizer class for the input stream.
- STEP_VECTORCREATION: The class that will be used for the vector creation; TF, TO or the TFIDF etc.

- STEP_WORDFILTER: The class for the stop word list

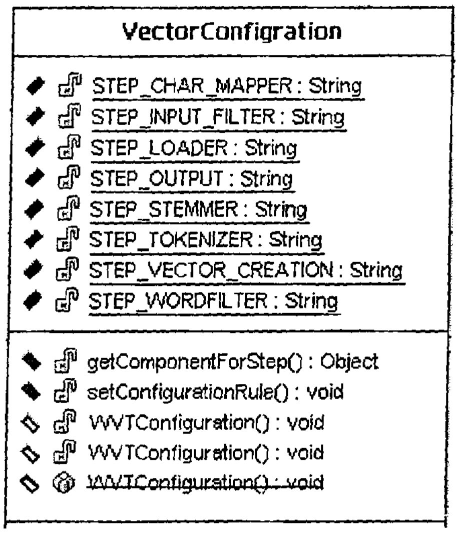

VectorConfigration
STEP_CHAR_MAPPER : String
STEP_INPUT_FILTER : String
STEP_LOADER : String
STEP_OUTPUT : String
STEP_STEMMER : String
STEP_TOKENIZER : String
STEP_VECTOR_CREATION : String
STEP_WORDFILTER : String
getComponentForStep() : Object
setConfigurationRule() : void
WVTConfiguration() : void
WVTConfiguration() : void
WVTConfiguration() : void

Figure 4-7 Vector Configuration Class

4.3 Document Classification

The Document Classification phase will implement LVQ algorithms and classify the input vectors. The classes in this phase will analyze the vector file for several statistical measures.

4.3.1 File Loader

This class will load the vector file in a set of instances. A single instance will be a horizontal linear array of numeric values and it will show a document with the actual class label at the inserted at the end of the array.

4.3.2 Parameter Initialization

This class will collect the required initiation and configuration algorithm parameters from the user for every LVQ algorithm. This class will be instantiated with default values for the algorithm parameters. User will be able to configure every parameter of LVQ algorithms.

The class will include following parameters:

- epsilon value for OLVQ3
- learning rate for algorithms
- total number of code book vectors
- initialization mode
- learning function
- total training iterations
- use of voting
- window size

Figure 4-8 Algorithm Parameter Class

4.3.3 LVQ Algorithms

These classes will implement LVQ algorithms.

4.3.3.1 LVQ1

The LVQ1 [29] selects a single set of best matching codebook vectors is selected and moved closer or further away from each data vector, per iteration if the classification decision is correct or wrong respectively.

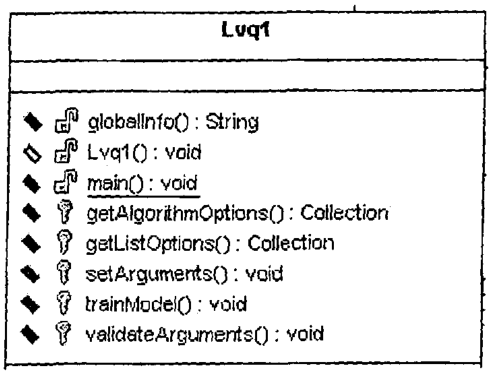


Figure 4-9 LVQ1 wrapper class

4.3.3.2 LVQ2

Two sets of best matching codebook vectors are selected and only updated if one set belongs to the desired class and the other does not, and the distance ratio is within a defined window. The value of the window is defined as the mid-point of the set of codebook vectors. [30]

4.3.3.3 LVQ3

The same as LVQ2.1 except if both set of codebook vectors belong to correct class; they are updated but adjusted using an epsilon value. The epsilon value is used to adjust the global learning rate.

4.3.3.4 OLVQ1

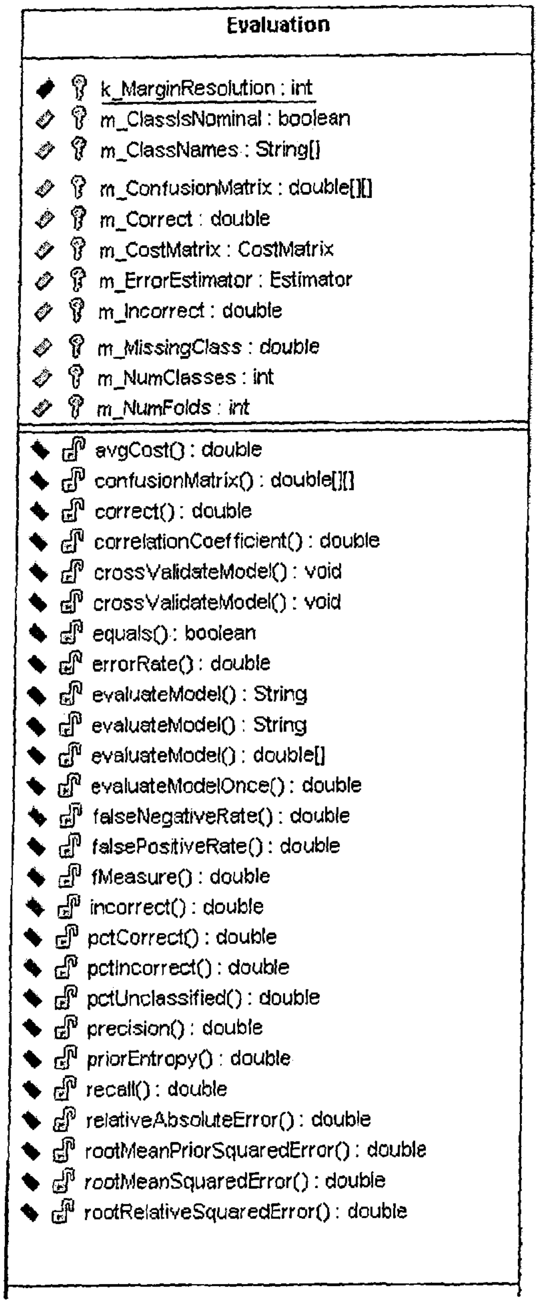
The Optimized LVQ1 [29] is same as LVQ1, except that each codebook vector has its own learning rate.

4.3.3.5 OLVQ3

The Optimized LVQ3 [31] is same as LVQ3 except each codebook vector has its own learning rate in the same manner as OLVQ1.

4.4 Classification Evaluation

The classification evaluation will consist of performing 10-fold cross validation method for all algorithms. 10-fold cross validation method divides the test set in 10 equal sets and uses one set for testing and the remaining nine sets are used for the training of classification model. The evaluation class will give F1-measure, Precision and Recall rates, and the Percentage of correct and incorrect classified instances



Evaluation

k_MarginResolution : int
m_ClassIsNominal : boolean
m_ClassNames : String[]
m_ConfusionMatrix : double[][]
m_Correct : double
m_CostMatrix : CostMatrix
m_ErrorEstimator : Estimator
m_Incorrect : double
m_MissingClass : double
m_NumClasses : int
m_NumFolds : int

avgCost() : double
confusionMatrix() : double[][]
correct() : double
correlationCoefficient() : double
crossValidateModel() : void
crossValidateModel() : void
equals() : boolean
errorRate() : double
evaluateModel() : String
evaluateModel() : String
evaluateModel() : double[]
evaluateModelOnce() : double
falseNegativeRate() : double
falsePositiveRate() : double
fMeasure() : double
incorrect() : double
pctCorrect() : double
pctIncorrect() : double
pctUnclassified() : double
precision() : double
priorEntropy() : double
recall() : double
relativeAbsoluteError() : double
rootMeanPriorSquaredError() : double
rootMeanSquaredError() : double
rootRelativeSquaredError() : double

Figure 4-10 Algorithm Evaluation Class

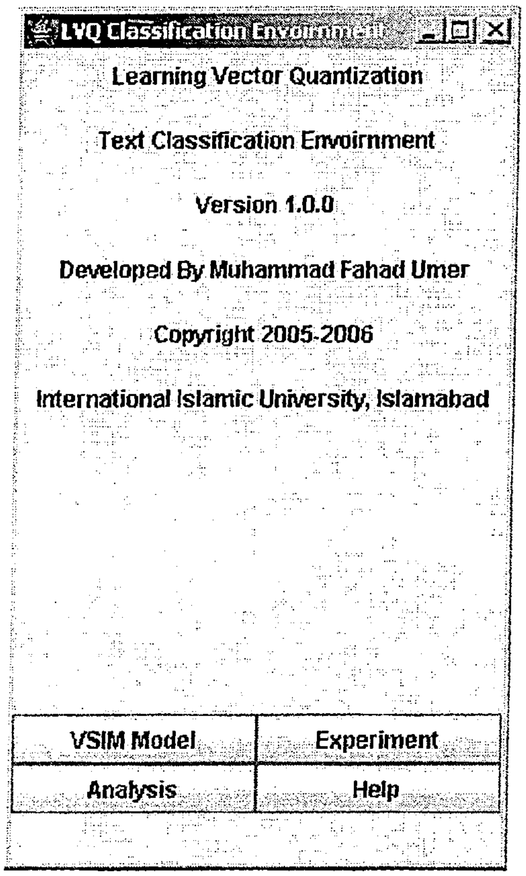
CHAPTER 5

EXPERIMENT

5 Experiment

The design of a text classification system plays a vital role in the construction of classifier. Following the design described in chapter 4, the classification system is constructed in two separate modules and these modules are joined resulting in a complete text classification application.

The application for the experiment is developed in Java as it provides relatively easy handling of text files and string tokens. The application has a main window, which combines the different modules of the application.


LVQ Classification Environment

Learning Vector Quantization

Text Classification Envoirnment

Version 1.0.0

Developed By Muhammad Fahad Umer

Copyright 2005-2006

International Islamic University, Islamabad

Figure 5-1 LVQ Text Classification Environment

5.1 VSIM Converter

This module converts the text documents to vectors using Vector Space Information Model. This method takes two inputs from the user, the directory path where the documents are placed and the dictionary file for those documents. This dictionary file is used to filter the string tokens received from the documents. The documents used in this experiment are in XML format. Therefore, an XML filter is used to remove the XML formatting tags. The Lovins stemmer algorithm was used to remove the suffixes from the different morphological forms of a single word. The other stemmer algorithms are Porters and Snowball stemmer algorithms. To remove the stop words, a text file containing the common stop words is supplied to filter tokens. The class can generate the document vectors with Binary weighting, Term Occurrence, Term Frequencies and normalized-TFIDF. The TFIDF is the most suitable weighting scheme as it makes the most occurring terms less effective and prevents the long documents from scoring high.

5.1.1 Initialization

The class constructor is initialized with the path of documents directory and dictionary of the words.

```
class VSIMTask
{
    WVTool wvt;
    Configuration config;
    InputList list;
    WordList wordList ;
    WordList wordListFile;
    String DocList[];
    String DirPath;
    public VSIMTask(int VSIMM, String dirpath) throws Exception
    {
      ...
      ...
    }
    ...
    ...
}
```

The Configuration class is initialized and the appropriate values for the different steps have been set. This class will provide a configuration object that will be used to configure LVQ algorithms for classification.

```
//Initialize the configuration
config = new WVTConfiguration();
//Set the input filter for XML documnets
config.setConfigurationRule(
    WVTConfiguration.STEP_INPUT_FILTER,new WVTConfigurationFact
            (new XMLInputFilter()));
//Set the Lovins stemmer algorithm for suffux stripping
config.setConfigurationRule(WVTConfiguration.STEP_STEMMER,
    new WVTConfigurationFact(new LovinsStemmerWrapper()));
try {
    //Initialize the stopword class
    config.setConfigurationRule(WVTConfiguration.STEP_WORDFILTER,
    new WVTConfigurationFact(new StopwordFilter()));
    }
    catch (Exception e)
    {
          e.printStackTrace();
    }
    list = new WVTInputList(10);

    //Load The Documnets from the dircetory
    list.addEntry(new WVTDocumentInfo(DirPath,"xml","","english"));

    //Create the word List
    wordList = wvt.createWordList(list,config);

    //store the word lists created in temporary files
    wordList.store(new FileWriter("wordlistinfo.txt"));
    wordList.storePlain(new FileWriter("wordlistplain.txt"));

    //Create the WVTOOL Format Doc Vectors and store in a
    temporary file
    FileWriter outFile = new FileWriter("wordvectors.txt");
    WordVectorWriter wvw = new WordVectorWriter (outFile,true);
    config.setConfigurationRule(WVTConfiguration.STEP_OUTPUT,new
    WVTConfigurationFact(wvw));
    ...
```

The integer value VSIM initialized in the constructor is used to define that which weighting scheme will be used for the vector creation.

```
switch (VSIMM)
{
    case 1:
          config.setConfigurationRule(
          WVTConfiguration.STEP_VECTOR_CREATION,
          new WVTConfigurationFact(new BinaryOccurrences()));
          break;
    case 2:
          config.setConfigurationRule(
          WVTConfiguration.STEP_VECTOR_CREATION,
          new WVTConfigurationFact(new TermOccurrences()));
          break;
    case 3:
          config.setConfigurationRule(
          WVTConfiguration.STEP_VECTOR_CREATION,
          new WVTConfigurationFact(new TermFrequency()));
```

```
            break;
        case 4:
            config.setConfigurationRule(
            WVTConfiguration.STEP_VECTOR_CREATION,
            new WVTConfigurationFact(new TFIDF()));
            JOptionPane.showMessageDialog(null,"The data is saved in
a file");
            break;
        default:
            config.setConfigurationRule(
            WVTConfiguration.STEP_VECTOR_CREATION,
                new WVTConfigurationFact(new TFIDF()));
}
//create the vectors
wvt.createVectors(list,config,wordList);
```

7The vectors are created and stored in a text file. The createVectors() function use the list of the documents, configuration object and dictionary words to create vectors. The vectors stored in text file at this stage are in raw form and are not readable. The classification algorithm requires input data in tab-separated two-dimensional array. This data is processed to generate the tab-separated vector values.

```
//Read the temporary files
File ReadableFile = new File("wordvectors.txt");
FileReader vectorFile = new FileReader("wordvectors.txt");
StringTokenizer strToken = new StringTokenizer(strBuffer,"\n");
int row = wordList.getNumWords();
int col = wordList.getNumDocuments();
float docArray [][] = new float[row][col];
...
while (strToken.hasMoreTokens())
{
    Token = strToken.nextToken();
    ...
}
    String s = Token;
    StringTokenizer termToken = new StringTokenizer(s,":");
    if (termToken.hasMoreTokens())
    {
        String termNo = termToken.nextToken();
        if (termToken.hasMoreTokens())
        {
```

```
                    String TFIDF = termToken.nextToken();
                    Integer ItemNo = new Integer(termNo);
                    Float F = new Float(TFIDF);
                    //round the value to 0.00
                    float in = Math.round(F.floatValue()*1000.00);
                    in = in/1000;
                    try {
                    docArray[ItemNo.intValue()][docNum] = in;
                    } catch (Exception e)
                    { e.printStackTrace(); }
               }
          }
     }
...
         FileWriter docVector = new FileWriter("vector.txt");
         writeBuffer = new StringBuffer[col];
...
         for (k=0; k<row; k++)
         {
           for (m=0; m<col; m++)
           {
             writeBuffer[m].append(docArray[k][m]).append(Delim);
             if (k==(row-1))
             {
        writeBuffer[m].append("").append(DocList[m]).append("\n");
             }
           }
         }
         for (int s=0; s<col; s++)
           docVector.write(writeBuffer[s].toString());
         ...
}
```

As every vector contains each of the vocabulary word, most of the vectors are sparse. These vectors are then converted in a comma separated two dimensional matrix form are store in a text file. The format of the text file is ARFF i.e. Attribute Relation File Format. The description of the ARFF format is given in the following table.

Table 5-1 ARFF File Format Description

ARFF Tag	Description
`@relation name`	A dataset has to start with a declaration of its name

`@attribute attribute_name specification`	It followed by a list of all the attributes in the dataset (including the class attribute). These declarations have the form
`@attribute nominal_attribute {first_value,second_value, third_value}`	If an attribute is nominal, specification contains a list of the possible attribute values in curly brackets
`@attribute numeric_attribute numeric`	If an attribute is numeric, specification is replaced by the keyword numeric: (Integer values are treated as real numbers in WEKA.)
`@attribute string_attribute string`	In addition to these two types of attributes, there also exists a string attribute type. This attribute provides the possibility to store a comment or ID field for each of the instances in a dataset
`@data`	Actual data is introduced by this tag, which is followed by a list of all the instances. The instances are listed in comma-separated format, with a question mark representing a missing value

The frame window for the VSIM conversion takes required data and dictionary path from user. The user than can select which of the term weighting scheme will be used, by clicking the appropriate checkbox.

The first text field will take document directory path and second text filed will take the dictionary path. Checking appropriate radio button can specify the method that will be used for vector creation.

When 'Start Vector Transformation' button will be clicked, application will start processing on the document and load output vectors in the text area at the bottom of the application window

These vectors can be saved in a file in the current working directory by clicking the 'Save Vectors' button.

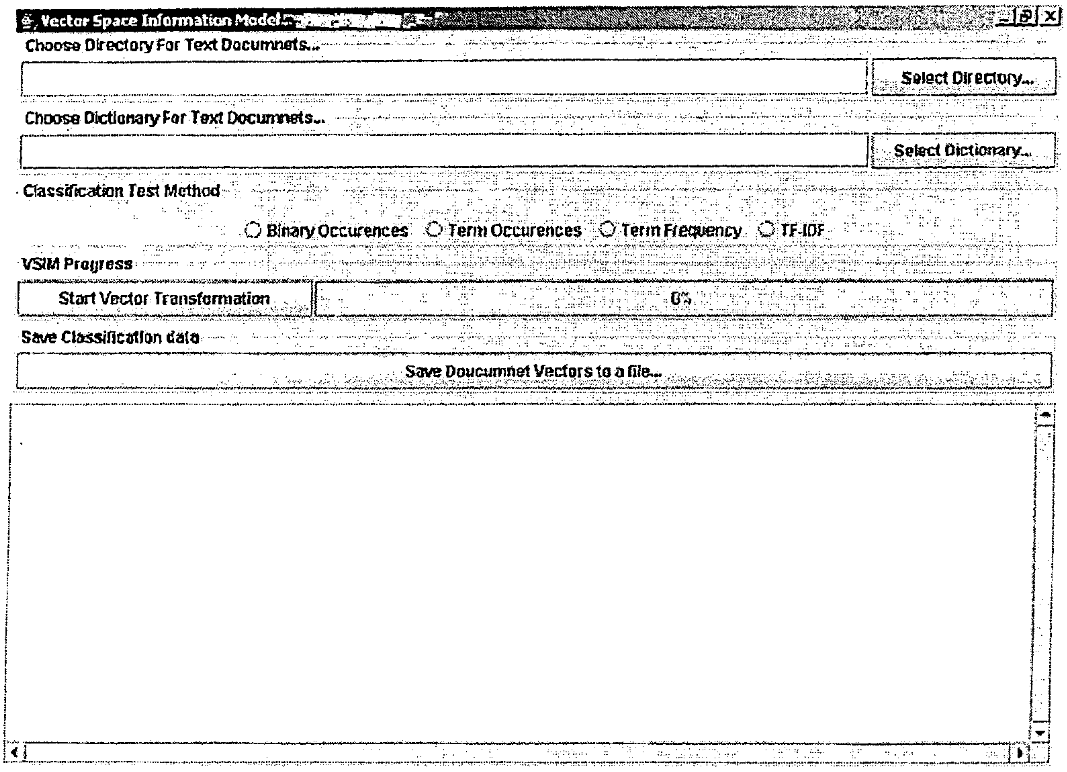

Vector Space Information Model
Choose Directory For Text Documnets...
Select Directory...
Choose Dictionary For Text Documnets...
Select Dictionary...
Classification Test Method
Binary Occurences
Term Occurences
Term Frequency
TF-IDF
VSIM Progress
Start Vector Transformation
0%
Save Classification data
Save Doucumnet Vectors to a file...

Figure 5-2 Vector Creation Application

5.2 Classification Experiment

This is main module used for performing the classification task. This module has the class ClassificationExp() that performs the classification operation. Main window of this class has three tabbed panes associated with it. The first pane is used for loading the ARFF file. The window displays the documents vectors in a table. When the user clicks on any attribute value in the table, following values about that attribute are displayed:

Attribute Name:	The name of the attribute.
Missing:	The number of missing values in data set for the attribute.
Type:	The type of attribute of value. i.e. numeric or nominal. Numeric attributes have real number values while the value for nominal attributes belong from a definite set of values such as the day of week.
Distinct:	The number of distinct values.
Unique:	The number of values that appear once.

5.2.1 Class Constructor

The class ClassificationExp() is declared with components required to display the user interface. Class constructor initializes the algorithm parameters with their default values. As the classification experiment window has three panes, three panels are initialized and added to tabbed windowpanes.

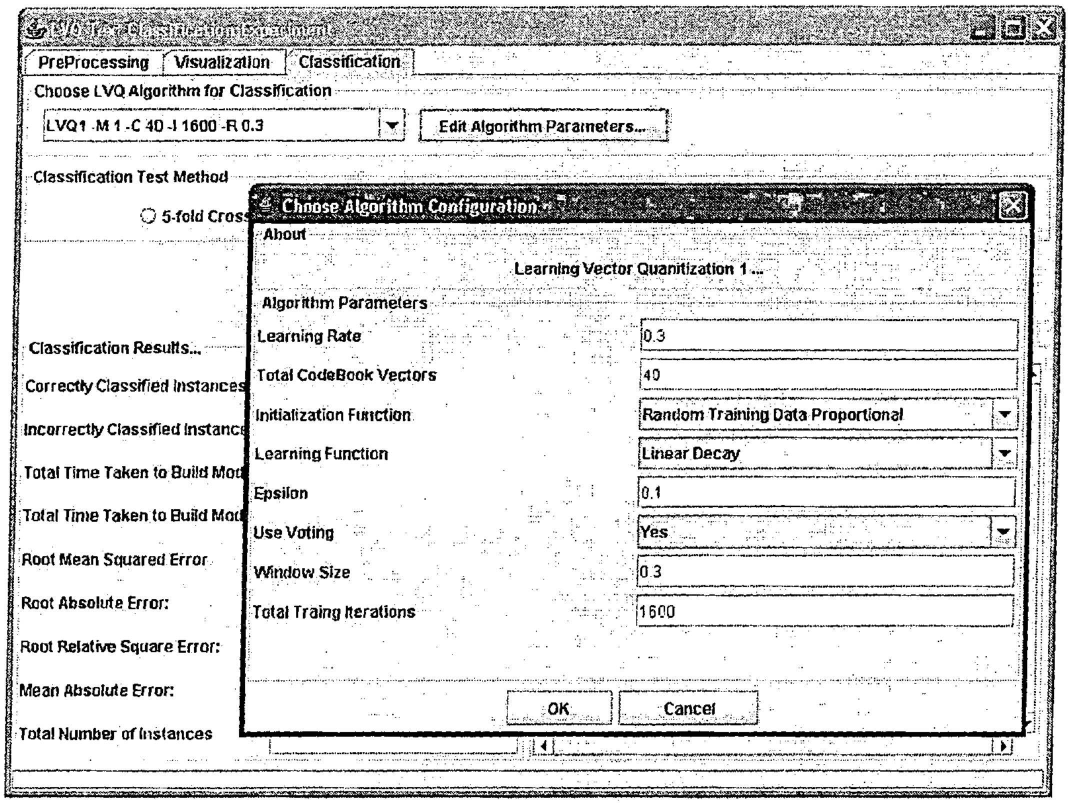
```
public class ClassficationExp extends JFrame {
  Instances data,Inst;
  BasePane BaseWindow;
  AlgoParam []algoParam;
  VisualizePanel VP;
  ...
  ...
  public ClassficationExp()
  {
      JFrame expFrm = this;
      algoParam = new AlgoParam[5];
      algoParam[0] = new AlgoParam(40,1600,1,1,1,0.1,0.3,0.3);
      algoParam[1] = new AlgoParam(40,1600,1,1,1,0.1,0.3,0.3);
      ...
      Container  expFrmCnt = expFrm.getContentPane();
      BaseWindow = new BasePane();
      ...
      JPanel PrePanel = new JPanel(new GridLayout(2,1));
      JPanel ClassPanel = new JPanel(new BorderLayout(5,5));
      VisualPanel = new JPanel(new GridLayout(1,2));
      BaseWindow.addTab("PreProcessing",PrePanel);
      BaseWindow.addTab("Visualization",VisualPanel);
      BaseWindow.addTab("Classification",ClassPanel);
      VisualPanel.add(VP);
      expFrmCnt.add(BaseWindow);
```

5.2.2 Algorithm Configuration

The classification pane has a drop down list, which displays all LVQ algorithms with their default values. When an algorithm is selected, its initialization parameters can be edited by clicking the Edit Parameters button next to it. A dialog box is displayed where user can supply the appropriate values. After user presses the OK button, new values are retrieved respectively and algorithm configuration is updated.

```
public void editParamBtn_Clicked(ActionEvent e)
{
    //get the selected algorithm
    int sItem = chooseLVQAlgoCmb.getSelectedIndex();
    dlg=new ParamDlg(
    chooseLVQAlgoCmb.getSelectedIndex(),algoParam[sItem]);
    ...
    if (sItem <= 2)
    {
          int temp = sItem+1;
```

```
                String updateIteml =
                "LVQ"+ temp + " -M "+algoParam[sItem].numInitMode+" -C "+
                algoParam[sItem].numCBV+" -I "+
                algoParam[sItem].numTranIter+" -R "+
                algoParam[sItem].learnRate;
                chooseLVQAlgoCmb.removeItemAt(sItem);
                chooseLVQAlgoCmb.insertItemAt(updateIteml,sItem);
                ...
                ...
        }
        ...
        ...
    }
```


PreProcessing
Visualization
Classification
Choose LVQ Algorithm for Classification
LVQ1 -M 1 -C 40 -I 1600 -R 0.3
Edit Algorithm Parameters...
Classification Test Method
5-fold Cross
Classification Results...
Correctly Classified Instances
Incorrectly Classified Instanc
Total Time Taken to Build Mod
Total Time Taken to Build Mod
Root Mean Squared Error
Root Absolute Error:
Root Relative Square Error:
Mean Absolute Error:
Total Number of Instances

Choose Algorithm Configuration
About
Learning Vector Quanitization 1 ...
Algorithm Parameters

Learning Rate	0.3
Total CodeBook Vectors	40
Initialization Function	Random Training Data Proportional
Learning Function	Linear Decay
Epsilon	0.1
Use Voting	Yes
Window Size	0.3
Total Traing Iterations	1600

OK
Cancel

Figure 5-3 Dialog for Algorithm Configuration

5.2.3 Algorithm Initiator

The algorithm initiator class 'ParamDlg' is derived from JDialog(...). This class stores the algorithm initialization parameters as an object of class 'AlgoParam'. An object of this class is used for every LVQ algorithm.

```
public class AlgoParam
{
    int numCBV,numTranIter,numLearnFunc,numInitMode,useVoting;
    double epsilon,wndSize,learnRate;
```

```
     public AlgoParam(int ncbv,int ntiter, int nlrnfnc,
     int nimode,int uvtng,double ep,double wsize,double lrate) {
     numCBV = ncbv;
     numTranIter = ntiter;
     numLearnFunc = 1;
     numInitMode = nimode;
     useVoting = uvtng;
     epsilon = ep;
     wndSize = wsize;
     learnRate = lrate;

  }
  public AlgoParam()
  {
  }
}
```

5.2.4 Classification Test Methods

Classification test methods can be specified by clicking the appropriate radio button. The test method options are follows:

a. 5-fold cross validation method

b. 10-fold cross validation method

c. 66% training split

d. User supplied test set

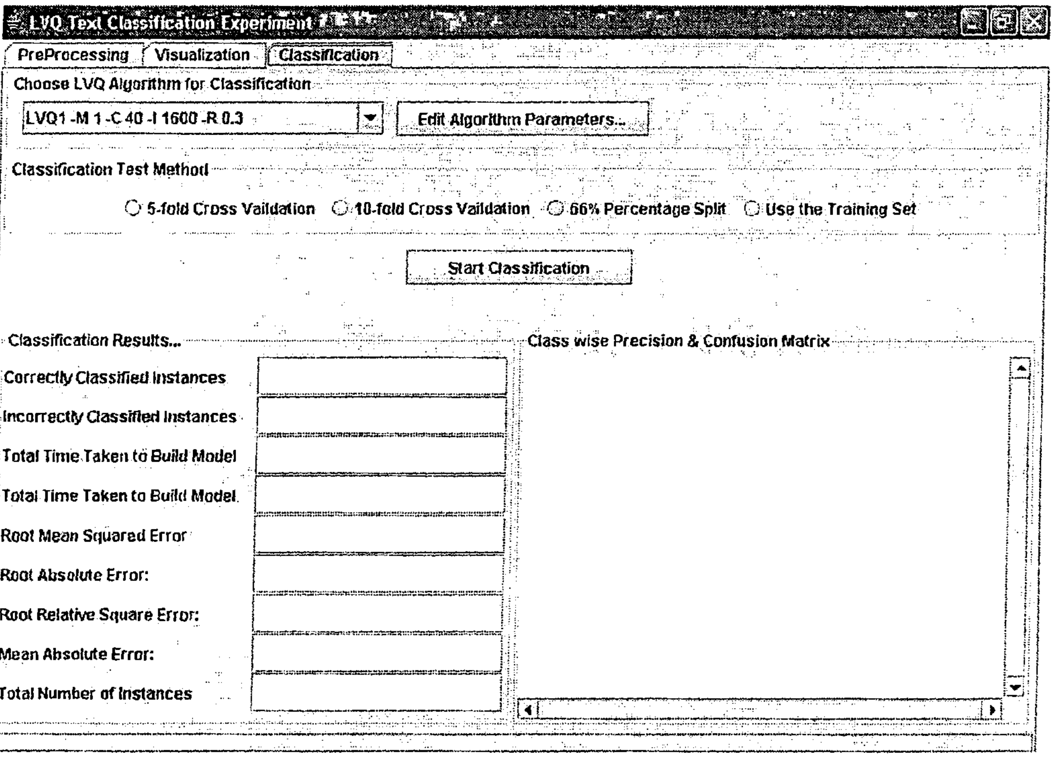
5.2.5 Classification Algorithm

The implementation of each algorithm is defined in a function. There are wrapper classes for each of LVQ algorithm. An object of the respective class is created and initiated with given parameters.

```
 public void OLVQ1 ()
 {
     try {
     Olvq1 algorithm = new Olvq1();
     switch (algoParam[3].numInitMode) {
     case 0:
           init = new SelectedTag(
           InitialisationFactory.INITALISE_TRAINING_PROPORTIONAL,
           InitialisationFactory.TAGS_MODEL_INITALISATION);
           break;
     ...
     ...
     }
     algorithm.setInitialisationMode(init);
     switch (algoParam[3].numLearnFunc) {
     case 0:`
```

```
            lfunc = new SelectedTag(
            LearningKernelFactory.LEARNING_FUNCTION_LINEAR,
            LearningKernelFactory.TAGS_LEARNING_FUNCTION);
        break;
      }
      algorithm.setLearningFunction(lfunc);
      algorithm.setSeed(1);
      algorithm.setTotalCodebookVectors(algoParam[3].numCBV);
      algorithm.setTotalTrainingIterations(algoParam[3].numTranIter);
      algorithm.setUseVoting(true);
```

Once the parameters for algorithm object are set, an instance of the Evaluation class is created and initialize with the given data set. Evaluation class has a function `crossValidateModel(...)`, which takes algorithm instance and data set and use n-fold cross validation model to obtain the result for given algorithm on the given data set.


LVQ Text Classification Experiment
PreProcessing
Visualization
Classification
Choose LVQ Algorithm for Classification
Edit Algorithm Parameters...
Classification Test Method
5-fold Cross Validation
10-fold Cross Validation
66% Percentage Split
Use the Training Set
Start Classification
Classification Results...
Class wise Precision & Confusion Matrix
Correctly Classified Instances
Incorrectly Classified Instances
Total Time Taken to Build Model
Total Time Taken to Build Model
Root Mean Squared Error
Root Absolute Error:
Root Relative Square Error:
Mean Absolute Error:
Total Number of Instances

LVQ1 -M 1 -C 40 -I 1600 -R 0.3

Figure 4 Classification Window

```
    // train and test the model (10 fold cross validation)
    Evaluation evaluation = new Evaluation(data);
    evaluation.crossValidateModel(algorithm, data, 10,
    new Random(algorithm.getSeed()));

    Double icrtval = new Double(evaluation.incorrect());
    Double crtval = new Double(evaluation.correct());
    ...
    ...
```

```
        ...

        }
        catch (Exception exp)
        {
                exp.printStackTrace();
        }

    }
```

As user clicks the Start Classification button, processing on the document vectors is started. The respective algorithms initialize the codebook vectors automatically. These codebook vectors are used to classify the test instances. Result values are populated in the respective text fields.

CHAPTER 6
ANALYSIS OF RESULTS

6 Analysis of Results

In order to objectively evaluate the LVQ and other classification algorithms, the well known Reuters-21578 collection has been used. The analysis of results is carried out in two phases: first the all five LVQ algorithms are compared for determining the most accurate algorithm for the text classification, Second: the best LVQ algorithm is compared with other classification approaches.

6.1 Comparison of LVQ algorithms

Each classification algorithm requires a configuration before it can be applied to the classification task. These configurations determine that how the algorithm will be trained, the length pf the training time, number of instances that will be used for he training and the learning rate of the algorithm. The parameters required to configure with their recommended values are discussed below:

epsilon -- Epsilon learning weight modifier used when both BMUs are of the instances class (recommend 0.1 or 0.5 should be smaller for smaller windowSize values).

initialisationMode -- Model (codebook vector) initalisation mode (1==Random Training Data Proportional, 2==Random Training Data Even, 3==Random Values In Range, 4==Simple KMeans, 5==Farthest First, 6==K-Nearest Neighbour Even)

learningFunction -- Learning rate function to use while training, linear is typically better (1==Linear Decay, 2==Inverse, 3==Static)

learningRate -- Initial learning rate value (recommend 0.3 or 0.5)

totalCodebookVectors -- Total number of codebook vectors in the model

totalTrainingIterations -- Total number of training iterations (recommended 30 to 50 times the number of codebook vectors).

useVoting -- Use dynamic voting to select the assigned class of each codebook vector, provides automatic handling of misclassified instances.

windowSize -- Window size matching codebook vectors must be within (recommend 0.2 or 0.3)

For evaluation purpose, the 10-fold cross validation method is used. This method divides the data set in 10 equal folds. The 9 folds are used for training the network while the remaining 1 fold is used for testing of algorithm. The folds for the cross validation method can be any number but 10 and 5-fold method are used normally.

6.1.1 LVQ1 Results

Table 6-1 LVQ1 Algorithm Configuration

Parameter	Configuration
Total codebook vectors	40
Use of voting	yes
Learning rate	0.3
Total training iterations	1600
Window size	0.3

Table 6-2 Classification Accuracy

Parameter	Value
Correctly Classified Instances	400, 83.50 %
Incorrectly Classified Instances	79, 16.49 %
Root mean squared error	0.2185

Table 6-3 LVQ1 Classification Result

Class	Precision	Recall	F-Measure
whe	0.911	0.911	0.911
tra	0.927	0.935	0.931
shi	0.639	0.568	0.601
int	0.911	0.988	0.948
cru	0.789	0.795	0.792

Table 6-4 LVQ1 Confusion Matrix

cru	Int	shi	tra	whe	←Classified as
51	4	0	0	1	whe
1	101	0	6	0	tra
1	2	46	1	31	shi
0	1	0	82	0	int
3	1	26	1	120	cru

Table 6-5 Classification Time Breakdown

Parameter	Configuration
Model Initialization Time	15ms
Model Training Time	1032ms
Total Model Preparation Time	1047ms
Time Taken to Build Model	1050ms

6.1.2 LVQ2 Results

Table 6-6 LVQ2 Algorithm Configuration

Parameter	Configuration
Total codebook vectors	50
Use of voting	Yes
Learning rate	0.5
Total training iterations	2200
Window size	0.3

Table 6-7 Classification Accuracy

Parameter	Value
Correctly Classified Instances	404, 84.3424 %
Incorrectly Classified Instances	75, 15.6576 %
Root mean squared error	0.2223

Table 6-8 LVQ2 Classification Result

Class	Precision	Recall	F-Measure
whe	0.941	0.857	0.897
tra	0.934	0.917	0.925
shi	0.616	0.654	0.635
int	0.921	0.988	0.953
cru	0.83	0.808	0.819

Table 6-9 LVQ2 Confusion Matrix

cru	Int	shi	Tra	whe	←Classified as
48	3	4	0	1	whe
1	99	2	6	0	tra
1	3	53	0	24	shi
0	1	0	82	0	int
1	0	27	1	122	cru

Table 6-10 Classification Time Breakdown

Parameter	Time
Model Initialization Time	0ms
Model Training Time	1750ms
Total Model Preparation Time	1750ms
Time Taken to Build Model	1750ms

6.1.3 LVQ3 Results

Table 6-11 LVQ3 Algorithm Configuration

Parameter	Configuration
Total codebook vectors	40
Use of voting	Yes
Learning rate	0.3
Total training iterations	2000
Epsilon	0.1
Window size	0.3

Table 6-12 Classification Accuracy

Parameter	Value
Correctly Classified Instances	399, 83.2985 %
Incorrectly Classified Instances	80, 16.7015 %
Root mean squared error	0.2179

Table 6-13 LVQ3 Classification Result

Class	Precision	Recall	F-Measure
whe	0.98	0.875	0.925
tra	0.927	0.944	0.936
shi	0.591	0.679	0.632
int	0.929	0.952	0.94
cru	0.809	0.755	0.781

Table 6-14 LVQ3 Confusion Matrix

cru	int	shi	tra	whe	←Classified as
49	2	3	3	1	whe
0	102	1	4	1	tra
1	0	55	0	25	shi
0	4	0	79	0	int
0	2	34	1	114	cru

Table 6-15 Classification Time Breakdown

Parameter	Configuration
Model Initialization Time	0ms
Model Training Time	1313ms
Total Model Preparation Time	1313ms

6.1.4 OLVQ1 Results

Table 6-16 OLVQ1 Algorithm Configuration

Parameter	Configuration
Total codebook vectors	40
Use of voting	yes
Learning rate	0.3
Total training iterations	1600
Window size	0.3

Table 6-17 Classification Accuracy

Parameter	Value
Correctly Classified Instances	408, 85.17 %
Incorrectly Classified Instances	71, 14.82%
Root mean squared error	0.2133

Table 6-18 OLVQ1 Classification Result

Class	Precision	Recall	F-Measure
Whe	0.926	0.893	0.909
Tra	0.927	0.944	0.936
Shi	0.667	0.642	0.654
Int	0.921	0.988	0.953
Cru	0.824	0.808	0.816

Table 6-19 OLVQ1 Confusion Matrix

cru	Int	shi	tra	whe	←Classified as
50	3	1	1	1	whe
0	102	0	5	1	tra
1	4	52	0	24	shi
0	1	0	82	0	int
3	0	25	1	122	cru

Table 6-20 Classification Time Breakdown

Parameter	Configuration
Model Initialization Time	15ms
Model Training Time	907ms
Total Model Preparation Time	922ms

6.1.5 OLVQ3 Results

Table 6-21 OLVQ3 Algorithm Configuration

Parameter	Configuration
Total codebook vectors	40
Use of voting	Yes
Learning rate	0.3
Total training iterations	1600
Window size	0.3

Table 6-22 Classification Accuracy

Parameter	Value
Correctly Classified Instances	400, 83.50 %
Incorrectly Classified Instances	79, 16.49 %
Root mean squared error	0.2185

Table 6-23 OLVQ3 Classification Result

Class	Precision	Recall	F-Measure
Whe	0.911	0.911	0.911
Tra	0.927	0.935	0.931
Shi	0.639	0.568	0.601
Int	0.911	0.988	0.948
Cru	0.789	0.795	0.792

Table 6-24 OLVQ3 Confusion Matrix

cru	Int	shi	tra	whe	←Classified as
51	4	0	0	1	whe
1	101	0	6	0	Tra
1	2	46	1	31	Shi
0	1	0	82	0	Int
3	1	26	1	120	Cru

Table 6-25 Classification Time Breakdown

Parameter	Configuration
Model Initialization Time	15ms
Model Training Time	1032ms
Total Model Preparation Time	1047ms
Time Taken to Build Model	1050ms

Table 6-26 Comparison of LVQ algorithms F1-measure

Algorithms/ Classes	LVQ1	LVQ2.1	LVQ3	OLVQ1	OLVQ3
wheat	0.911	0.897	0.925	0.909	0.923
trade	0.931	0.925	0.936	0.936	0.922
ship	0.601	0.635	0.632	0.654	0.642
interest	0.948	0.953	0.94	0.953	0.94
crude	0.792	0.819	0.781	0.816	0.809

The three LVQ-algorithms LVQ1, LVQ2.1 and LVQ3, yield almost similar accuracy for the classification procedure. The LVQ1 and the LVQ3 define a more robust process, whereby the codebook vectors assume stationary values even after extended learning periods. For the LVQ1 the learning rate can approximately be optimized for quick convergence. In LVQ2.1, there is no guarantee for the codebook vectors being placed optimally to describe the forms of the class borders. Therefore the LVQ2.1 should only be used in a differential fashion, using a small value of learning rate and low number of training steps.

LVQ1 are LVQ3 with optimized learning rate have given higher accuracies than other LVQ algorithms. Optimized-LVQ1 performs better than its other counterparts. It gives F1-measure over 90% for whe, gra and tra categories. The shi category has the lowest F1-measure.i.e 65.4%. The reason for this is not the poor performance of the classifier but there were some documents which were common between the shi and cru category. Because the classification system was based on hard categorization rule [2], it had to classify the overlapping documents in one of the target classes.

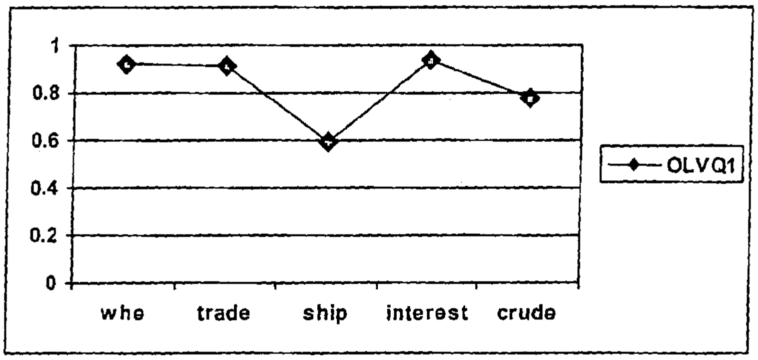
The performance of the OLVQ1 algorithm is also fairly reasonable if the training time of the classifier is considered. It took only 922 ms on a P-IV 2.4GHz to perform 1600 training iterations on 40 codebook vectors for building model on 10-folds cross validation method.

6.2 Comparison of Classification Algorithms

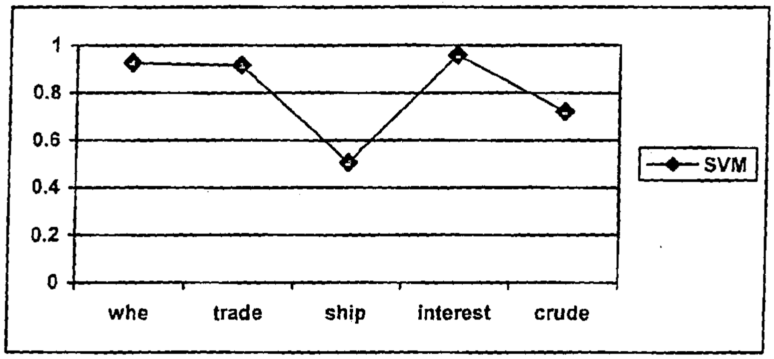
The results generated by the experiment are quite excellent Table 6 compares the results with the other classification algorithms with same training and test set. Other classification model, such as Naïve Bayes, K-NN, and C4 give low accuracy and also their training and classification time is greater than LVQ.

Due to the memory and computation limitations, the results in Table 6 were obtained using 5-folds cross validation. The results show that for the whe, tra and int category, SVMs works well than OLVQ1 and for the shi and cru, the OLVQ1 is better than SVM. One can say that as SVM works well for three categories and OLVQ1 is good only for two categories, it is better than OLVQ1. In fact, SVM work well than LVQ but the problem lies with the time taken by SVM to build the model. SVMs took 6 times more time than OLVQ1. This short training and classification time encourages the use of LVQ for the classification task

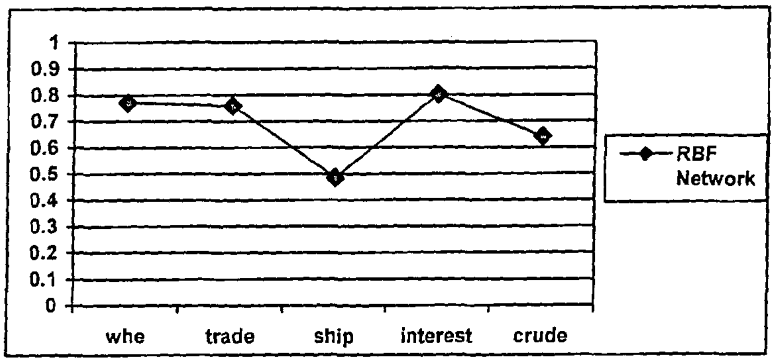
With the greater accuracy and very short training and classification time, LVQ networks seem to be prospective for the classification of text data.



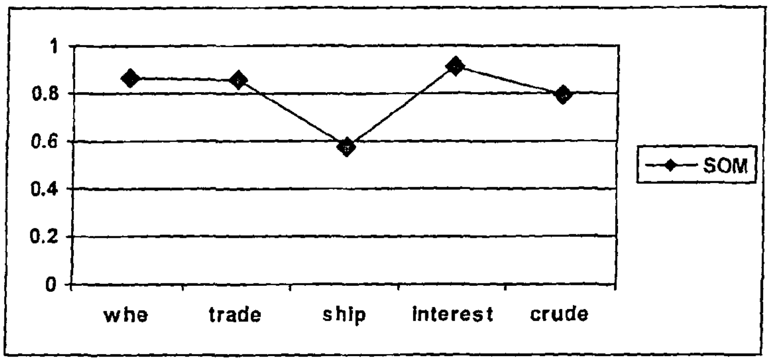
(a) F1-measure for OLVQ1



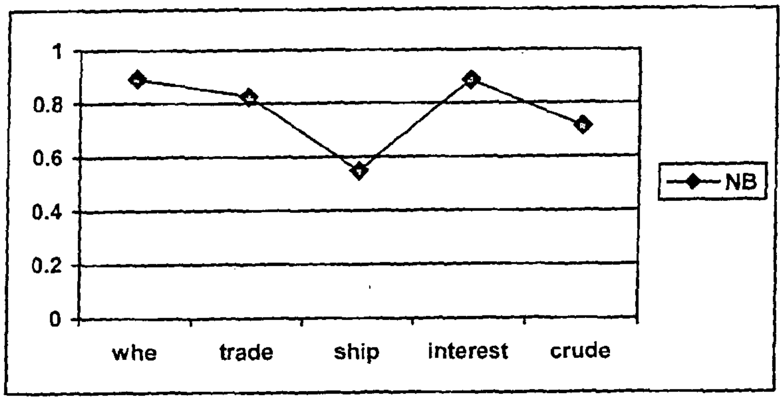
(b) F1-measure for SVM



(c) F1-measure for RBF Network



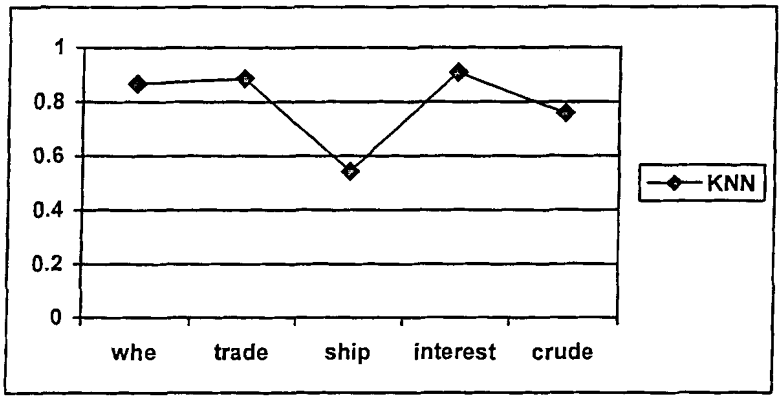
(d) F1-measure for Self-organizing Maps



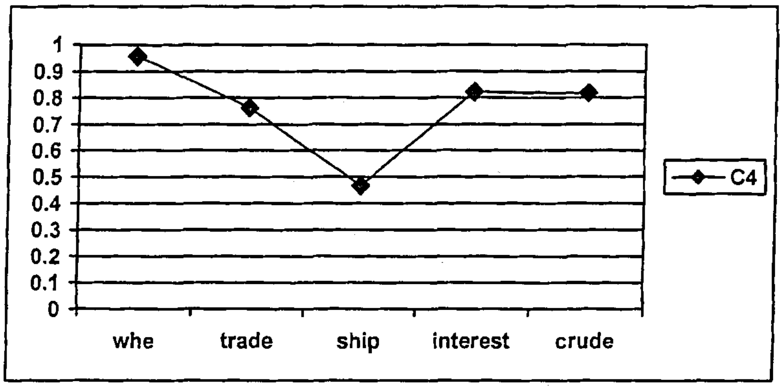
(g) F1-measure for Naïve Bayes

Table 6-27 F1-measure comparison of classification procedures

Classifier/ Class	OLVQ1	SVM	RBF Network	SOM	KNN=7	C4	Naïve Bayes
whe	0.917	**0.927**	0.771	0.867	0.867	0.957	0.893
tra	0.91	**0.917**	0.759	0.853	0.886	0.762	0.825
shi	**0.593**	0.506	0.484	0.577	0.542	0.467	0.547
int	0.936	**0.959**	0.802	0.911	0.909	0.822	0.886
cru	**0.776**	0.722	0.641	0.795	0.758	0.817	0.714
Time (Normalized)	0.1099	0.745	0.4005	0.5103	0.001	0.0989	0.1444



(e) F1-measure for K-NN (K=7)



(f) F1-measure for K-NN (K=7)

CHAPTER 7

CONCLUSION & FUTURE WORK

7. Conclusion & Future Work

7.1 Conclusion

This paper presents an application of LVQ for text classification. The process of classifying documents with LVQ consists of three phases; pre-processing of data, training of the network and the testing of classification network. In pre-processing phase, common words are removed from the texts, suffix striping is carried out and least and most occurring terms are removed. The text documents are represented as vectors using Vector Space Information Model. Each cell of the vector denotes the weighted frequency of a tem in that document.

The experimental results show that the LVQ classifier performs well and its effectiveness is comparable to mostly well known text classifiers. One major advantage of the LVQ based classifier is its relatively fast training time.

The results generated by the experiment are relatively exceptional. The LVQ networks seem to be prospective for the classification of text documents, with the advantage of restricting documents to be part of certain classes.

7.2 Future Work

This research uses Vector Space Information Model for the representation of text. The problem with Vector Space Information Model is that in the representation, all pairs are considered equally similar. Semantically relationships between the terms are not taken into account (Honkela, 1997). The LVQ classification can be applied by using some other approach such as Poisson distribution. The use of Poisson model is widely investigated in Information Retrieval but it is rarely used for the text classification.

There is no way to determine a good number of codebook vectors. Some mechanism of finding an optimal number of codebook vectors should be embedded in the learning algorithm.

The results show that SVMs are also very effective in the classification task. To obtain highly accurate classification system, a classification committee can be constructed using LVQ and SVM.

This study demonstrates how promising LVQ text classification systems could be. The next step in this research is to study the effectiveness of LVQ algorithms with image collections.

References

[1] Sebastiani, F., Machine Learning in Classification, ACM Computing Surveys, (34) 1, 2002, 1–47.

[2] Michie, D., Spiegelhalter, D.J. and Taylor. C.C, (Ed), *ML, Neural and Statistical Classification*, 1994.

[3] Mitchell, T.M., *Machine Learning*, McGraw Hill, New York, 1996.

[4] Dave A., and George M. *Artificial Neural Network Technology*, Data & Analysis Center for Software, NY, 1992.

[5] Krose, B., and Smagt, P., *An introduction to Neural Networks*, University of Amsterdam.1996.

[6] Kohonen, T., Hynninen, J., Kangas, J., Laaksonen, J., and Torkkola, K., *LVQ_PAK-The Learning Vector Quantization Program Package*, Helsinki University of Technology, 1995

[7] Lewis D. D. Naïve (Bayes) at forty: The independence assumption in information retrieval. *Proceedings of 10th European Conference on Machine Learning*, 1998, 4-15.

[8] Dumais, S. T., Platt, J., Heckerman, D., and Sahami, M. Inductive learning algorithms and representations for text categorization. In *Proceedings of 7th ACM International Conference on Information and Knowledge Management*, Bethesda, MD, 1998, 148–155.

[9] Yang, Y., and Liu, X. A Re-examination of Text Categorization Methods. In *Proceedings of 22nd ACM International Conference on Research and Development in Information Retrieval*, Berkeley, CA, 1999, 42-49.

[10] Joachims, T. Text categorization with support vector machines: learning with many relevant features. *In Proceedings of 10th European Conference on Machine Learning*, Chemnitz, Germany, 1998, 137–142.

[11] Joachims, T. Transductive inference for text classification using support vector machines. In *Proceedings of 16th International Conference on Machine Learning*, Bled, Slovenia, 1999, 200–209.

[12] Rocchio, Jr. J. J. Relevance Feedback in Information Retrieval. The SMART project Experiments in Automatic Document Processing, editor: Gerard Salton, Prentice-Hall, Englewood Cliffs, New Jersey, 1971.

[13] Joachims, T. A probabilistic analysis of the Rocchio algorithm with TFIDF for text categorization. In *Proceedings of ICML-97, 14th International Conference on Machine Learning*, Nashville, TN, 1997, 143–151.

[14] SINGHAL, A., MITRA, M., AND BUCKLEY, C. 1997. Learning routing queries in a query zone. In Proceedings of SIGIR-97, 20th ACM International Conference on Research and Development in Information Retrieval (Philadelphia, PA,1997), 25–32.

[15] SCHAPIRE, R. E., SINGER, Y., AND SINGHAL, A. 1998. Boosting and Rocchio applied to text filtering. In Proceedings of SIGIR-98, 21st ACM International Conference on Research and Development in Information Retrieval (Melbourne, Australia, 1998), 215–223.

[16] SALTON, G. AND BUCKLEY, C. 1988. Term-weighting approaches in automatic text retrieval. Inform. Process. Man. 24, 5, 513–523. Also reprinted in Sparck Jones and Willett [1997], pp. 323–328.

[17] Fuhr, N., Hartmann, S., Knorz, G., LustiG, G., Schwantner, M., and Tzeras, K. AIR/X—a rule-based multistage indexing system for large subject fields. In *Proceedings of 3rd International Conference "Recherche d'Information Assistee par Ordinateur"*, Barcelona, Spain, 1991, 606–623.

[18] Cohen, W. W., and Hirsh, H. Joins that generalize: text classification using WHIRL. In *Proceedings of 4th International Conference on Knowledge Discovery and Data Mining*, New York, 1998, 169–173.

[19] Li, Y.H. and Jian, A.K. Classification of Text Documents *The Computer J.*, (41)8, 1998, 537-546.

[20] Agrawal, R. and Srikant, S., Fast algorithm for mining association rules. In Proc. 1994 Int. Conf. Very Large Data Bases, pages 487-499, Santiago, Chile, September 1994.

[21] Han, J., Pei, J. and Yin, Y., Mining frequent patterns without candidate generation. In ACM SGMOD, Dallas, 2000.

[22] Osmar R. Zaiane, Maria-Luiza Antonie, Classifying Text Documents by Associating Terms with Text Categories, Department of Computer Science, University of Alberta, Canada, 2001.

[23] Vladimir N. Vapnik, *The Nature of Statistical Learning Theory*, Springer, 1995.

[24] Fuhr, N., Hartmann, S., Knorz, G., LustiG, G., Schwantner, M., and Tzeras, K. AIR/X—a rule-based multistage indexing system for large subject fields. In *Proceedings of 3rd International Conference "Recherche d'Information Assistee par Ordinateur"*, Barcelona, Spain, 1991, 606–623.

[25] Lam, W., Ho, C. Y. Using a generalized instance set for automatic text categorization. In *Proceedings of 21st ACM International Conference on Research and Development in Information Retrieval*, Melbourne, Australia, 1998, 81–89.

[26] http://www.daviddlewis.com/resources/testcollections/reuters21578/

[27] C.J. van Rijsbergen. *Information Retrieval*, Butterworth's, London, 1979.

[28] Andrew, K., 'The development of a fast conflation algorithm for English'. Dissertation submitted for the Diploma in Computer Science, University of Cambridge, (unpublished), 1971.

[29] Kohonen, T. The self-organizing maps. *Proceedings of the IEEE*, 78(9), 1990, 1464-1480

[30] Kohonen, T. Improved versions of Learning Vector Quantization. In *proceeding of the National Joint Conference of Neural Networks*, San Diego, 1990, 545-550

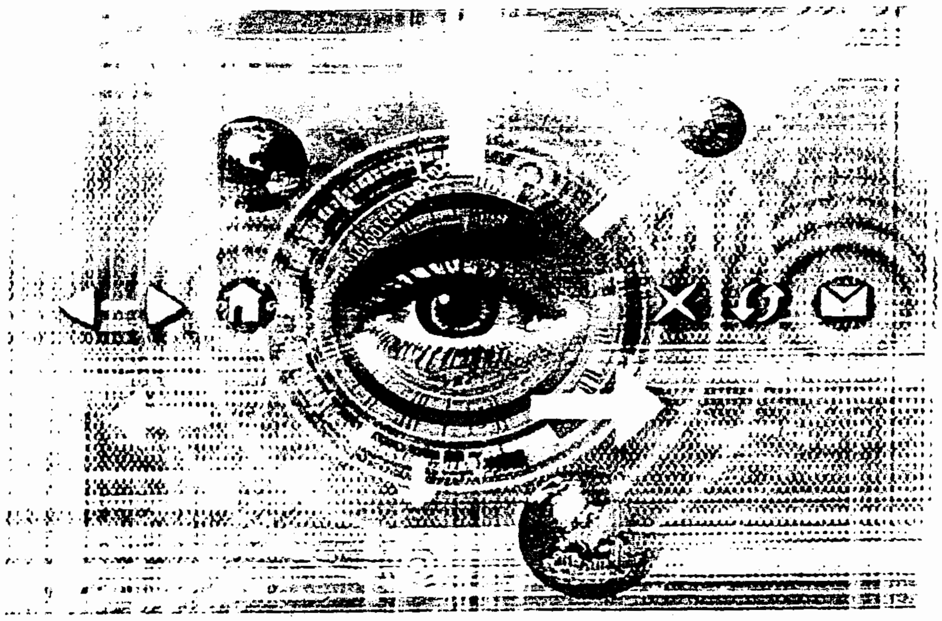
[31] Kohonen, T. New Developments of Learning Vector Quantization and the Self Organizing Map. In *symposium on Neural Networks; Alliances and Perspectives in Senri*, Osaka, Japan, 1992.

[32] Honkela, T. *Self-Organizing-Maps in Natural Language Processing*, Ph. D Thesis, Helsinki University of Technology, Finland, 1997.

APPENDIX A

RESEARCH PAPER

INFORMATION TECHNOLOGY JOURNAL



ANSInet

308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Tel: 0092 300 6661982, Fax: 0092 21 5206036, E-mail: support@ansimail.org

Information Technology Journal 6 (1): 154-159, 2007
ISSN 1812-5638
© 2007 Asian Network for Scientific Information

Classification of Textual Documents Using Learning Vector Quantization

Muhammad Fahad Umer and M. Sikander Hayat Khiyal
Department of Computer Science, International Islamic University, Islamabad, Pakistan

Abstract: The classification of a large collection of texts into predefined set of classes is an enduring research problem. The comparative study of classification algorithms shows that there is a tradeoff between accuracy and complexity of the classification systems. This study evaluates the Learning Vector Quantization (LVQ) network for classifying text documents. In the LVQ method, each class is described by a relatively small number of codebook vectors. These codebook vectors are placed in the feature space such that the decision boundaries are approximated by the nearest neighbor rule. The LVQ require less training examples and are much faster than other classification methods. The experimental results show that the Learning Vector Quantization approach outperforms the k-NN, Rocchio, NB and Decision Tree classifiers and is comparable to SVMs.

Key words: Learning vector quantization, text classification, artificial neural networks

INTRODUCTION

The automated classification has gained invigorated interest in the last decade. The information on the Internet continues to grow at an incredible speed with more than 4.5 billion pages available online. It has become a very challenging task to classify such large collection of information. Text Classification (TC) is one of the prime techniques to deal with the textual data. TC systems are used in a number of applications such as, filtering email messages, classifying customer reviews for large e-commerce sites, web page classification for an internet directory (e.g., Google), evaluating exams paper answers and organizing document databases in semantic categories.

The term classification has been used in a broader context in human activity. We may have a set of observations and want to infer classes or clusters within the set. Or we may have a certain number of classes and we want to classify a new sample into one of the existing classes. The former type is known as Clustering and the latter is known as Classification.

The TC system categorizes the documents into a fixed number of predefined classes. Formally, it can be defined as the task of assigning a Boolean value to each pair (d_j, c_i) where $d_j = \{d_1, d_2, d_3, \ldots d_n\}$ is the set of text documents and $c_i = \{c_1, c_2, c_3, \ldots c_n\}$ is the set of class labels. The value assigned to the pair could be true if the document d_j falls under class c_i or false if the document d_j does not belong to class c_i (Sebastiani, 2002).

The research in automated text classification started in early 1960s. A long list of successes and failures are reported in this field. Many methods had been proposed and a lot of experiments had been carried out. All this research had been done in the field of Information Retrieval and classification was a part of it. The rapid growth of Internet has revived the interest in automated text classification. Hand-built directories of web content suggest one solution to the dilemma, but unfortunately creating and maintaining such directories requires enormous amounts of human effort.

Many classification methods have been suggested in literature such as; Rocchio's classifiers (Rocchio, 1971) and (Joachims, 1997), k-NN (Yang and Liu, 1999), Naïve Bayes (Lewis, 1998), Decision Tree (Dumais, 1998) and Support Vector Machines (Joachims, 1998, 1999).

The k-NN is an example based classifier. For deciding whether a document d belongs to a class c or not, k-NN retrieve the k neighboring documents of d and they vote for the classification; if there is a majority vote for class c, a positive decision is taken and negative otherwise. The success of classification in k-NN depends upon the value of k, but there is no defined way to calculate it. Since k-NN is a lazy classifier, i.e., there is no training stage and all the computation is performed at the classification time, it cannot be used in real-time scenarios to classify large collection of texts.

The Rocchio method (1971) and (Joachims, 1997) selects an average prototype vector for every class. It calculates the similarity between a document and each of prototype vectors and the document is assigned to the class with maximum similarity. A problem with Rocchio classifier discussed by Lam and Ho (1998) is that it restricts the hypothesis space to the set of linear separable hyper plane regions, which has less expressive power than that of k-NN algorithms.

Corresponding Author: Muhammad Fahad Umer, Department of Computer Science, International Islamic University, Islamabad, Pakistan

The Naïve Bayes algorithm (Lewis, 1998) calculates the probability of each class for a document. The document is assigned to the class for which the probability is highest. There are many improvements to the Naïve Bayes classification model. A problem with Naïve Bayes discussed by Shen and Jiang (2003), is that when asked to make predictions; it always gives class posteriors very close to 0 or 1 and smoother class posteriors cannot be determined.

Support Vector Machines are employed in text classification by Joachims (1998, 1999). SVMs are linear classifiers that define a decision surface to separate classes of data as positive and negative. Kernel functions are used for nonlinear separation. SVMs have shown superb performance for text classification tasks and perhaps the best classifiers till now (Yang and Liu, 1999). SVMs' only potential drawback is their training time and memory requirement. For n training instances held in memory, the best-known SVM implementations take time proportional to n^a, where a is typically between 1.8 and 2 (Chakrabart *et al.*, 2002).

The Decision Tree (DT) classifiers are based on tree induction algorithms. A DT classifier is a tree, which internal nodes denote the terms and the branches departing from them are labeled with predicate applied to the terms. Each leaf nodes denotes a class. The DT classifiers discussed in literature are based on ID3 (Fuhr *et al.*, 1991), C4.5 (Cohen and Hirsh, 1998) and C5 (Li and Jian, 1998).

THE CLASSIFICATION SYSTEM

The automated text classification system is shown in Fig 1. The first phase is the selection of training and test material. This training set is processed through several pre-processes phases such as, the removal of common words, feature selection and word stemming.

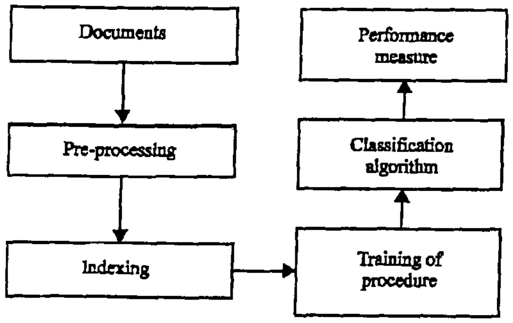


Fig. 1: The classification system

A vocabulary list is constructed containing all of the important terms and this list is used to index the training set. The training set is used for learning the classification. The test set is used to generate the results and the results are analyzed by using some standard performance measures for the evaluation of the classifier.

TEXT SELECTION AND INDEXING

The Reuters-21578 text collection (Lewis, 2006) has been used for evaluation purpose. There are many version of Reuters-21578 available and the ModeApte version was selected. From this collection, the documents having more than 2,000 characters have been selected, as most of the documents in the collection contain only a single line or just news heading. This filtering process made the collection more meaning full and helped us to quickly generate the results from the experimental setup. The final subset contains five categories and the number of document as shown in Table 1.

Normally, text categorization systems use a vector model representation of the documents. The same representation has been used for this system. Each document is represented as a vector and each cell of the vector represents the weighted frequency of the term in that document. There are many different schemes in use for the weighting of the term frequencies. One that seems effective is the Term Frequency-Inverse Document Frequency. That is, the number of times the word appears in the document multiplied by a function of the inverse of the number of documents in which the word appears. Terms that appear often in a document and do not appear in many documents therefore have an important weight.

The creation of vectors for documents consists of following steps:

- A word list containing all of the important terms is created for all the documents.
- This list can be scanned further to remove some common words and to eliminate the most and least frequently occurring terms.
- The document collection is indexed on the basis of word list using normalized TF-IDF.

The creation of a vocabulary list for a collection of documents is not a trivial task. The problems involved in constructing such systems as described by

Table 1: Total number of documents per category

Category	Interest	Ship	Trade	Crude	Wheat	Total
Document	83	81	108	151	56	479

van Rijsbergen (1979) are (1) removal of high frequency or common words, (2) suffix stripping, (3) detecting equivalent stems. Comparing the input text with a stop list of words can easily carry out the removal of high frequency words. This process reduces the size of the dictionary to a considerable limit.

The next problem of suffix stripping is more complicated. Two words should be compared for their conceptual meaning because a conventional string comparison will produce high error rate. The solution to this problem is to remove the suffixes from the different forms of a word. There are standard algorithms defined for suffix stripping called stemming algorithms such as Lovins' algorithm (Andrew, 1971).

Many words, after suffix removal map to one morphological form, but still there are some, which don't. One way to deal with this problem is to have a list of equivalent stems and two words should be considered equivalent if and only if their stems match and there is an entry in the list defining their suffixes as equivalent. This paper does not consider the problem of equivalency of stems as there are very few such words in a document and usually they don't affect the accuracy of the classifier (van Rijsbergen, 1979).

After applying stop list and stemming algorithm, a vocabulary list of 468 words was obtained containing important terms of all documents. We indexed the document collection on the basis of vocabulary list obtaining a two dimensional sparse matrix which contained documents row wise and terms column wise.

NEURAL NETWORKS CLASSIFICATION

A Neural Network (NN) is a network of units called neurons. The neuron is the basic processing element of NN. The inputs to a neuron arrive through synaptic connections. The efficacy of inputs is modeled by the weights attached with every input. The response of the neuron is a nonlinear function of its weighted inputs.

The classification model based on NN has generally, more than one layer of connected neurons. The input layer has the input units representing terms and the output layer output units representing the classes. The intermediate or hidden layers are used for the computing the classification decision. For classifying a document d_i its terms t_k with weights w_k are loaded into the input units; the output of these units is propagated through the intermediate neuron layers (if present) to the output layers and the value of the output units determine the classification decision.

A usual way of training NN is back-propagation. When a learning pattern is presented, the activation values of input neurons are propagated through the intermediate layers to the output layers and the actual output is compared with target output, if a miss-match occurs, the error is back-propagated so as to change the parameters of the network to minimize the error.

The simplest type of NNet classifier is the perceptron discussed in Dagan *et al.* (1997) and (Ng *et al.*, 1997) which is a linear classifier. A nonlinear NNet (Lam and Lee, 1999) and (Ruiz and Srinivasan, 1999) is instead a network with one or more additional layers of units, which in TC usually represent higher order interactions between terms that the network is able to learn (Sebastiani, 2002).

LVQ networks are based on the supervised competitive learning. LVQ networks attempt to define decision boundaries in the input space, given a set of exemplary decisions (the training data). Topologically; the network contains an input layer, a competitive layer and an output layer. The output layer has the neurons equal to the number of classes. In the competitive layer, each competitive unit corresponds to a cluster, the center of which is called a codebook vector. The Euclidean distance of an input vector is computed with each codebook vector and the nearest codebook vector is declared winner. Unlike perceptron, LVQ networks can classify any set of input vectors, not just linearly separable sets of input vectors. The only requirement is that the competitive layer must have enough neurons and each class must be assigned enough competitive neurons.

LVQ ALGORITHMS

There are a number of somewhat different LVQ algorithms appearing in the literature, they are all based on the following basic algorithm:

- A learning sample consisting of input vector x_i together with its correct class label c_i is presented to the network.
- A suitable number of codebook vectors are selected for every class label c_i.
- Using distance measures between codebook vectors and input vector d_i, the winner is determined. In some cases, the second best winner is also determined.

We have used LVQ1 (Kohonen, 1990b), LVQ2.1 (Kohonen, 1990a), LVQ3 (Kohonen, 1990b) and the optimized learning rate algorithms OLVQ1 (Kohonen, 1992), OLVQ3 for the classification task.

LVQ1: The LVQ1 (Kohonen, 1990b) selects a single set of best matching codebook vectors is selected and moved closer or further away from each data vector, per iteration if the classification decision is correct or wrong, respectively.

OLVQ1: The Optimized LVQ1 (Kohonen, 1992) is same as LVQ1, except that each codebook vector has its own learning rate

LVQ2.1: Two sets of best matching codebook vectors are selected and only updated if one belongs to the desired class and one does not and the distance ratio is within a defined window. The value of the window is defined as the mid-point of the two codebook vectors.

LVQ3: The same as LVQ2.1 except if both set of codebook vectors are of the correct class; they are updated but adjusted using an epsilon value. The epsilon value is used to adjust the global learning rate.

OLVQ3: The Optimized LVQ3 is same as LVQ3 except each codebook vector has its own learning rate in the same manner as OLVQ1

These training algorithms yield almost similar accuracies, although different techniques underlie each other. The experiment has been performed using all five algorithms and the results are analyzed to determine that which algorithm performs well for text classification. In the next step, the best LVQ algorithm is compared with the other classification algorithms.

THE EXPERIMENTAL RESULTS

Table 2 shows the experimental results of the five LVQ algorithms applied to the document vectors. The parameters for every algorithm have been selected empirically by slightly increasing and decreasing their value and analyzing the output.

Precision and Recall measures are widely used for evaluating the classifiers. Recall is defined to be the ratio of correct assignments by the system divided by the total number of correct assignments. Precision is the ratio of correct assignments by the system divided by the total number of the system's assignments. It is hard to compare classifiers using two measures, the F1 measure, introduced by (van Rijsbergen, 1979), combines recall (r) and precision (p) with an equal weight in the following form:

$$F\beta(r,p) = \frac{(\beta^2+1)pr}{\beta^2(p+r)}, \text{where } \beta = 1$$

The F1-measure has been used for evaluating the accuracy of the classifiers. The n-fold cross validation method was used to obtain the results. This method divides the data set in equal n partitions with using one partition as test set and rest of them as training set. The results show that almost each LVQ algorithm gives similar accuracies. These algorithms also took almost similar time for building the training model. But the optimized-LVQ1 performs well than its counterparts. It gives F1-measure over 90% for wheat and trade categories. The ship category has the lowest F1-measure.i.e., 65.4%. The reason for this is not the poor performance of the classifier but the fact that there were some documents which were common between the ship and crude category and the classification system had to classify the overlapping documents in one of the target classes.

The performance of the OLVQ1 algorithm is also quite reasonable if we consider the training time of the classifier. It took only 922 ms on a 2.4 GHz machine to perform 1600 training iterations on 40 codebook vectors for building a single model for 10-folds cross validation method.

Table 3 compares the results with the other classification algorithms with the same training and test set. Other classification model, such as Naïve Bayes, k-NN and C4 give low accuracy and also their training and classification time is greater than the LVQ. Due to the memory and computation limitations, the results in Table 3 were obtained using 5-folds cross validation method. The results show that for the wheat, trade and interest category, SVMs work well than OLVQ1 and for the ship and crude, the OLVQ1 is better than SVMs. One can say that as SVMs work well for three categories and OLVQ1 is good only for two categories, it is better than OLVQ1. In fact, SVMs are, but the problem lies with the time taken by SVM to build the model. SVMs took 6.25s to build the model while OLVQ1 took only 1.06s that is six times less. This modest training and classification time encourages the use of LVQ for the classification task.

Table 2: F1-measure comparison of LVQ algorithms (10-fold cross validation)

Algorithms/Classes	LVQ1	LVQ2.1	LVQ3	OLVQ1	OLVQ3
Wheat	0.911	0.897	0.925	0.909	0.923
Trade	0.931	0.925	0.936	0.936	0.922
Ship	0.601	0.635	0.632	0.654	0.642
Interest	0.948	0.953	0.940	0.953	0.940
Crude	0.792	0.819	0.781	0.816	0.809

Table 3: F1-measure comparison of classification procedures (5-fold cross validation)

Classifier/			RBF		k-NN		
Class	OLVQ1	SVM	Network	SOM	K = 7	C4	NB
Wheat	0.917	0.927	0.771	0.867	0.867	0.957	0.893
Trade	0.910	0.917	0.759	0.853	0.886	0.762	0.825
Ship	0.593	0.506	0.484	0.577	0.542	0.467	0.547
Interest	0.936	0.959	0.802	0.911	0.909	0.822	0.886
Crude	0.776	0.722	0.641	0.795	0.758	0.817	0.714

FUTURE WORK

This research uses Vector Space Information Model for the representation of text. The problem with Vector Space Information Model is that in the representation, all pairs are considered equally similar. Semantically relationships between the terms are not taken into account (Honkela, 1997). The LVQ classification can be applied by using some other approach such as Poisson distribution. The use of Poisson model is widely investigated in Information Retrieval but it is rarely used for the text classification.

There is no way to determine a good number of codebook vectors. Some mechanism of finding an optimal number of codebook vectors should be embedded in the learning algorithm.

The classification of binary data such as images can be explored with the LVQ.

CONCLUSIONS

This study presents an application of LVQ for text classification. The process of classifying documents with LVQ consists of three phases; pre-processing of data, training of the network and the testing of classification network. The text documents are represented as vectors using Vector Space Information Model. The results generated by the experiment are relatively exceptional. The LVQ network seems to be prospective for the classification of text documents, with the advantage of restricting documents to be a part of certain classes.

This study provides sufficient theoretical base for the development of a fully functional text classification application.

REFERENCES

Andrew, K., 1971. The development of a fast conflation algorithm for English. Dissertation submitted for the Diploma in Computer Science, University of Cambridge, (unpublished).

Chakrabart, S., S. Roy and M. Soundalgekar, 2002. Fast and accurate text classification via multiple linear discriminant projections. Proceedings of the 28th VLDB Conference, Hong Kong, China.

Cohen, W.W. and H. Hirsh, 1998. Joins that generalize: Text classification using WHIRL. In Proceedings of 4th International Conference on Knowledge Discovery and Data Mining, New York, pp: 169-173.

Dagan, I., Y. Karov and D. Roth, 1997. Mistaken driven learning in text categorization. In: Proceedings of 2nd Conference on Empirical Methods in Natural Language Processing, Providence, RI, pp: 55-63.

Dumais, S.T., J. Platt, D. Heckerman and M. Sahami, 1998. Inductive learning algorithms and representations for text categorization. In Proceedings of 7th ACM International Conference on Information and Knowledge Management, Bethesda, MD, pp: 148-155.

Fuhr, N., S. Hartmann, G. Knorz, G. LustiG, M. Schwantner and K. Tzeras, 1991. AIR/X-a rule-based multistage indexing system for large subject fields. In Proceedings of 3rd International Conference Recherche d'Information Assistee par Ordinateur, Barcelona, Spain, pp: 606-623.

Honkela, T., 1997. Self-Organizing-Maps in Natural Language Processing, Ph.D Thesis, Helsinki University of Technology, Finland.

Joachims, T., 1997. A probabilistic analysis of the Rocchio algorithm with TFIDF for text categorization. In Proceedings of ICML-97, 14th International Conference on Machine Learning, Nashville, TN, pp: 143-151.

Joachims, T., 1998. Text categorization with support vector machines: Learning with many relevant features. In Proceedings of 10th European Conference on Machine Learning, Chemnitz, Germany, pp: 137-142.

Joachims, T., 1999. Transductive inference for text classification using support vector machines. In Proceedings of 16th International Conference on Machine Learning, Bled, Slovenia, pp: 200-209.

Kohonen, T., 1990a. Improved versions of Learning Vector Quantization. In: proceeding of the National Joint Conference of Neural Networks, San Diego, pp: 545-550

Kohonen, T., 1990b. The self-organizing maps. Proceedings of the IEEE, 78: 1464-1480.

Kohonen, T., 1992. New Developments of Learning Vector Quantization and the Self-Organizing Map. In: Symposium on Neural Networks; Alliances and Perspectives in Senri, Osaka, Japan.

Lam, W. and C.Y. Ho, 1998. Using a generalized instance set for automatic text categorization. In: Proceedings of 21st ACM International Conference on Research and Development in Information Retrieval, Melbourne, Australia, pp: 81-89.

Lam, S.L., and D.L. Lee, 1999. Feature reduction for neural network based text categorization. In Proceedings of 6th IEEE Int. Conference on Database Advanced Systems for Advanced Application, Taiwan, pp: 195-202.

Lewis, D.D., 1998. Naïve (Bayes) at forty: The independence assumption in information retrieval. Proceedings of 10th European Conference on Machine Learning, Chemnitz, Germany, pp: 4-15.

Lewis, D.D., 2006. http://www.daviddlewis.com/resources/testcollections/reuters21578/, Last accessed Mar 2006.

Li, Y.H. and A.K. Jian, 1998. Classification of text documents. Computer J., 8: 537-546.

Ng, H.T., W.B. Goh and K.L. Low, 1997. Feature selection, perceptron learning and a usability case study for text categorization. In: Proceedings of 20th ACM Intl. Conference on Research and Development in Information Retrieval, Philadelphia, PA, pp: 67-73.

Rocchio, Jr. J.J., 1971. Relevance Feedback in Information Retrieval. The SMART project Experiments in Automatic Document Processing, Editor: Gerard Salton, Prentice-Hall, Englewood Cliffs, New Jersey.

Ruiz, M.E. and P. Srinivasan, 1999. Hierarchical neural networks for text categorization. In Proceedings of 22nd ACM International Conference on Research and Development in Information Retrieval, Berkeley, CA, pp: 281-282.

Sebastiani, F., 2002. Machine learning in classification, ACM Computing Surveys, 1: 1-47.

Shen, Y. and J. Jiang, 2003. Improving the performance of Naive Bayes for text classification, technical report (Unpublished), Stanford Natural Language Processing (NLP) Group, Stanford University, CA.

van Rijsbergen, C.J., 1979. Information Retrieval, Butterworth's, London.

Yang, Y. and X. Liu, 1999. A Re-examination of Text Categorization Methods. In: Proceedings of 22nd ACM International Conference on Research and Development in Information Retrieval, Berkeley, CA, pp: 42-49.