UNIVERSITY OF NAIROBI

School of Computing & Informatics

Comparing the Performance of Naïve Bayes and Support Vector Machines in Text Classification

Submitted by

Nicholas Muchai Gateru

Supervisor: Mr. Evans Anderson Kirimi Miriti

A project report submitted in partial fulfillment of the requirement for the award of the degree of Master of Science in Computer Science

September 2014

DECLARATION

Student

I declare that this report is my original work and has not been presented to any other university for an academic award.

Sign: ……. Date: ……

Nicholas Muchai Gateru
P58/75750/2012

Supervisor

This project report has been submitted as a partial fulfillment of the requirements for the degree of Master of Science in Computer Science of the University of Nairobi with my approval as a university supervisor.

Sign: ……. Date: ……

Mr. Evans Anderson Kirimi Miriti
SCHOOL OF COMPUTING & INFORMATICS

ACKNOWLEDGEMENTS

I'm thankful to my family members and loved ones for their support and patience along the way.

I thank Mr. Evans K. Miriti of University of Nairobi for supervision and guidance through this process, his dedication and commitment was highlighted appreciated.

I thank and appreciate the contributors to Sckit-learn for assembling the tools used in analyzing the data.

I also give thanks to the school of computing and informatics lecturers who gave their time to educate and advise me on various aspects of my study.

Finally I thank the School of Computing and Informatics community for giving me a favorable environment to successfully carry out this study.

ABSTRACT

Classification is a supervised learning task whose goal is to infer a prediction model using a training dataset containing instances whose category membership is known, and then using the model to assign class labels to testing instances whose class labels are unknown. E.g. in spam filtering, already labelled mail as either spam or not spam is used to train a classifier, and the classifier is then used in the future to automatically place mail whose category is unknown, into either spam or not spam categories.

Training of a classifier progresses from gathering a training set that is representative of the real world, thereafter, the input data is represented into a feature vector that contains the features that describe the object. With input features in place, a training algorithm e.g. SVM or Naïve Bayes is selected and run on the training set to come up with a predicting function. The function is run on the testing set and its prediction accuracy and performance is measured.

Owing to the proliferation of easily available textual data of late, the need and interest to classify that data has increased. In the real-world, the ability to automatically classify documents into a fixed set of categories is highly desirable.

Machine learning offers powerful tools for automatically classifying documents. A techniques performance depends not only on the algorithm in use, but also on the characteristics of the data in use. As such, it's prudent to apply various techniques on classifying the same dataset and try to analyze the performance of each technique relative to the particular data.

In this project, we compared the performance of Support Vector Machines and Naïve Bayes algorithms in the task of text classification by using the '20 newsgroups' dataset. The '20 newsgroups' dataset comprises around 20,000 newsgroup posts on 20 topics split in two subsets: one for training and the other one for testing.

The data pre-processing, training of the classifiers, testing of the classifiers and performance evaluation was accomplished by implementing a python script.

Performance evaluation was done by comparing: training time, testing time, precision, recall, and F-measure scores for each classifier when each classifier was run against 4,887 documents and 10,794 documents.

We found that SVM achieved an F-score of 0.969 and Naïve Bayes an F-score of 0.964 when tested using 4,887 documents. When tested using 10,794 documents, SVM achieved an F-

score of 0.900, and Naïve Bayes an F-score of 0.869. We also found that, for 4,887 documents, SVM took 0.676s to train, while Naïve Bayes took 0.026s to train for the same number of documents. For 10,794 documents, SVM took 3.733s to train while Naïve Bayes took 0.106s to train for the same number of documents.

The findings show that the size of the dataset affected the performance of both classifiers, i.e. with more documents used, both classifiers were less able to place documents in their correct classes. The findings also confirm the existing findings of the suitability of SVM as compared to other classifiers to classify text.

Keywords: Text classification, SVM, Naïve Bayes, Recall, Precision, F-measure

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS AND KEY TERMINOLOGIES

Supervised learning:	the machine learning task of inferring a function from labeled training data
Classifier:	mathematical function, implemented by a classification algorithm that maps input data to a category
SVM:	Support Vector Machine
NB:	Naïve Bayes
MultinomialNB:	Multinomial Naïve Bayes
BernoulliNB:	Bernoulli Naïve Bayes
K-NN:	K-Nearest Neighbors
Python:	a general-purpose, high-level programming language
Sckit-Learn:	an open source machine learning library for the Python programming language
Liblinear:	an open source library for large-scale linear classification
NumPy:	the fundamental package for scientific computing with Python
SciPy:	an open source library of scientific tools
Matplotlib:	a python 2D plotting library which produces publication quality figures
Tf–idf:	term frequency-inverse document frequency
Recall:	the ratio of correct assignments by the system divided by the total number of correct assignments. (Intuitively, recall is the ability of the classifier to find all the positive samples)
Precision:	the ratio of correct assignments by the system divided by the total number of the system's assignments. (Intuitively, precision is the ability of the classifier not to label as positive a sample that is negative)
F-measure:	combines recall (r) and precision (p) with an equal weight

1. INTRODUCTION

1.1. Background

Classification is a supervised learning task whose goal is to infer a prediction model using a training dataset containing instances whose category membership is known, and then using the model to assign class labels to testing instances whose class labels are unknown. E.g. in spam filtering, already labelled mail as either spam or not spam is used to train a classifier, and the classifier is then used in the future to automatically place mail whose category is unknown, into either spam or not spam categories.

Training of a classifier progresses from gathering a training set that is representative of the real world, thereafter, the input data is represented into a feature vector that contains the features that describe the object. With input features in place, a training algorithm e.g. SVM or Naïve Bayes is selected and run on the training set to come up with a predicting function. The function is run on the testing set and its prediction accuracy and performance is measured.

Owing to the proliferation of easily available textual data of late, the need and interest to classify that data has increased. In the real-world, the ability to automatically classify documents into a fixed set of categories is highly desirable.

Some common application areas of automatic text classification include:

i. **News filtering and Organization**

   Most of the news services today are electronic in nature in which a large volume of news articles are created every single day by the organizations. In such cases, it is difficult to organize the news articles manually. Therefore, automated methods can be very useful for news categorization in a variety of web portals.

ii. **Document Organization and Retrieval**

   A variety of supervised methods may be used for document organization in many domains. These include large digital libraries of documents, web collections, scientific literature, or even social feeds. Hierarchically organized document collections can be particularly useful for browsing and retrieval.

iii. **Opinion Mining**

This deals with the computational treatment of opinion, sentiment, and subjectivity in text. Customer reviews or opinions are often short text documents which can be mined to determine useful information from the review.

iv. Email Classification and Spam Filtering

It is often desirable to classify email in order to determine either the subject or to determine junk email in an automated way.

Machine learning offers powerful tools for automatically classifying documents. A techniques performance depends not only on the algorithm in use, but also on the characteristics of the data in use. As such, it's prudent to apply various techniques on classifying the same dataset and try to analyze the performance of each technique relative to the particular data.

Key methods commonly used for text classification

i. Decision Trees
ii. Pattern (Rule)-based Classifiers
iii. SVM Classifiers
iv. Neural Network Classifiers
v. Bayesian (Generative) Classifiers
vi. Nearest Neighbor Classifiers
vii. Genetic algorithm-based classifiers

Previous studies related to this study include: work done by Thorsten Joachims (1997), comparing SVM, Naive Bayes for multivariate Bernoulli models, C4.5, Rocchio algorithm, and K-NN. From his results, SVMs performed substantially better than all other methods. Comparing training time, SVMs roughly compared to C4.5, but were more expensive than naive Bayes, Rocchio, and k-NN. Specifically, as far as SVM and Naive Bayes for multivariate Bernoulli models were concerned, Thorsten Joachims (1997) obtained a F1-score of (.860-864) for SVM and a F1-score of 0.720 for Naive Bayes for multivariate Bernoulli models, using the 'Reuters' and the 'Ohsumed collection' datasets.

Andrew McCallum and Kamal Nigam (1998), compared Naive Bayes for multinomial models and Naive Bayes for multivariate Bernoulli models, using five datasets, and showed that multinomial model to be almost uniformly better than the multivariate Bernoulli model. Results on five real world corpora they found that the multinomial model reduced errors by an average of 27%, and sometimes by more than 50%.

1.2. Problem Statement

The goal of text categorization is the classification of documents into a fixed number of predefined categories. This allows users to find desired information faster by searching only the relevant categories and not the entire information space. The importance of text classification is even more apparent when the information space is huge such as the World Wide Web.

Machine learning offers powerful tools for automatically classifying documents. A techniques performance depends not only on the algorithm in use, but also on the characteristics of the data in use. Hence given a particular dataset and a particular task, it's important to know the right tool for the task. As such, evaluating how different algorithms fair in classifying text is of much value.

1.3. Objectives

The main objective of this research was to evaluate and compare the performance of Support Vector Machines versus Naïve Bayes in the task of text classification.

The specific objectives were to:

i. To experiment with the feature sets and compare the performance of SVM and Naïve Bayes techniques, in categorizing the '20 newsgroup' dataset.
ii. Examine the classifier learning abilities for an increasing number of documents in the dataset.
iii. To evaluate each classifier from the perspectives of precision, recall, and F-measure.

1.4. Significance

There are billions of text documents available in electronic form. More and more are becoming available every day. The Web itself contains over a billion documents. Millions of people send e-mail every day. Academic publications and journals are becoming available in electronic form. These collections and many others represent a massive amount of information that is easily accessible. However, seeking value in this huge collection requires organization. Many web sites offer a hierarchically-organized view of the Web. E-mail clients offer a system for filtering e-mail. Numerous academic communities have a Web site that allows searching on papers and shows an organization of papers. However, organizing documents by hand or creating rules for filtering is painstaking and labor-intensive. This can be greatly aided by automated classifier systems.

Of late, many data management tools such as: Database management systems, Data mining systems come equipped with a number of classification and clustering algorithms. Given such a scenario, it's important for the end user, end user here refers to the potential data analyst, software developer, and Business Intelligence developer and so on, to have an idea on the suitability of each algorithm for a particular task. As such it's important to analyze the performance of different algorithms given a particular dataset or scenario.

1.5. Scope

This project aimed at classifying the datasets by using ***multiclass classification***, which makes the assumption that each sample is assigned to one and only one label rather than ***multilabel classification,*** which assigns each sample a set of target labels. This can be thought of as predicting properties of a data-point that are not mutually exclusive, such as topics that are relevant for a document. A text might be about any of religion, politics, finance or education at the same time or none of these.

In addition, this project didn't identify region(s) of a document corresponding to topic(s), which is place different regions of the same document into the respective categories. Neither did we capture correlation between topics.

2. LITERATURE REVIEW

2.1. Introduction

In this chapter we present a review of related literature. We proceed with an overview of the general classification space, follow that up with a detailed review of algorithms that can be used in text classification. Then present results from previous classifier comparison work, and close with a summary of the chapter.

2.2. Classification overview

Classification is a supervised learning task whose goal is to infer a prediction model using a training dataset containing instances whose category membership is known, and then using the model to assign class labels to testing instances whose class labels are unknown. E.g. in spam filtering, already labelled mail as either spam or not spam is used to train a classifier, and the classifier is then used in the future to automatically place mail whose category is unknown, into either spam or not spam categories.

Training of a classifier progresses from gathering a training set that is representative of the real world, from thence, the input data is represented into a feature vector that contains the features that describe the object. With input features in place, a training algorithm e.g. SVM or Naïve Bayes is selected and run on the training set to come up with a predicting function. The function is run on the testing set and its prediction accuracy and performance is measured.

Text classification presents different challenges, this is because some of the words are much more likely to be correlated to the class distribution than others. As such, a wide array of methods have been proposed with a goal of determining the most important features for the purpose of classification. In the following sub sections, we review the algorithms that can be used in the task of text classification.

2.3. Classifiers used in text classification

2.3.1. Decision Tree Classifiers

A decision tree is essentially a hierarchical decomposition of the (training) data space, in which a predicate or a condition on the attribute value is used in order to divide the data space hierarchically. In the context of text data, such predicates are typically conditions on the presence or absence of one or more words in the document. The division of the data space is performed recursively in the decision tree, until the leaf nodes contain a certain minimum

number of records, or some conditions on class purity. The majority class label (or cost-weighted majority label) in the leaf node is used for the purposes of classification. For a given test instance, we apply the sequence of predicates at the nodes, in order to traverse a path of the tree in top-down fashion and determine the relevant leaf node. In order to further reduce the overfitting, some of the nodes may be pruned by holding out a part of the data, which are not used to construct the tree. The portion of the data which is held out is used in order to determine whether or not the constructed leaf node should be pruned or not. In particular, if the class distribution in the training data (for decision tree construction) is very different from the class distribution in the training data which is used for pruning, then it is assumed that the node overfits the training data. Such a node can be pruned. In the particular case of text data, the predicates for the decision tree nodes are typically defined in terms of the terms in the underlying text collection. For example, a node may be partitioned into its children nodes depending upon the presence or absence of a particular term in the document. We note that different nodes at the same level of the tree may use different terms for the partitioning process. Many other kinds of predicates are possible. It may not be necessary to use individual terms for partitioning, but one may measure the similarity of documents to correlated sets of terms. These correlated sets of terms may be used to further partition the document collection, based on the similarity of the document to them.

2.3.2. Rule-based Classifiers

Decision trees are also generally related to rule-based classifiers. In rule-based classifiers, the data space is modeled with a set of rules, in which the left hand side is a condition on the underlying feature set, and the right hand side is the class label. The rule set is essentially the model which is generated from the training data. For a given test instance, we determine the set of rules for which the test instance satisfies the condition on the left hand side of the rule. We determine the predicted class label as a function of the class labels of the rules which are satisfied by the test instance. In its most general form, the left hand side of the rule is a Boolean condition, which is expressed in Disjunctive Normal Form (DNF). However, in most cases, the condition on the left hand side is much simpler and represents a set of terms, all of which must be present in the document for the condition to be satisfied. The absence of terms is rarely used, because such rules are not likely to be very informative for sparse text data, in which most words in the lexicon will typically not be present in it by default (sparseness property). Also, while the set intersection of conditions on term presence is used often, the union of such conditions is rarely used in a single rule. This is because such rules can be split

into two separate rules, each of which is more informative on its own. For example, the rule Honda ∪ Toyota ⇒ Cars can be replaced by two separate rules Honda ⇒ Cars and Toyota ⇒ Cars without any loss of information. In fact, since the confidence of each of the two rules can now be measured separately, this can be more useful. On the other hand, the rule Honda ∩ Toyota ⇒ Cars is certainly much more informative than the individual rules. Thus, in practice, for sparse data sets such as text, rules are much more likely to be expressed as a simple conjunction of conditions on term presence. Decision trees and decision rules both tend to encode rules on the feature space, except that the decision tree tends to achieve this goal with a hierarchical approach. The main difference is that the decision tree framework is a strict hierarchical partitioning of the data space, whereas rule-based classifiers allow for overlaps in the decision space. The general principle is to create a rule set, such that all points in the decision space are covered by at least one rule. In most cases, this is achieved by generating a set of targeted rules which are related to the different classes, and one default catch-all rule, which can cover all the remaining instances.

2.3.3. Probabilistic Classifiers

Probabilistic classifiers are designed to use an implicit mixture model for generation of the underlying documents. This mixture model typically assumes that each class is a component of the mixture. Each mixture component is essentially a generative model, which provides the probability of sampling a particular term for that component or class. This is why this kind of classifiers are often also called generative classifiers. The Naive Bayes classifier, which is one of the subject classifiers to be covered in this survey, is perhaps the simplest and also the most commonly used generative classifier. It models the distribution of the documents in each class using a probabilistic model with independence assumptions about the distributions of different terms. Two classes of models are commonly used for naive Bayes classification. Both models essentially compute the posterior probability of a class, based on the distribution of the words in the document. These models ignore the actual position of the words in the document, and work with the "bag of words" assumption. The major difference between these two models is the assumption in terms of taking (or not taking) word frequencies into account, and the corresponding approach for sampling the probability space:

- Multivariate Bernoulli Model: In this model, what is used the presence or absence of words in a text document as features to represent a document. Thus, the frequencies of the words are not used for the modeling a document, and the word features in the text are assumed to be binary, with the two values indicating

presence or absence of a word in text. Since the features to be modeled are binary, the model for documents in each class is a multivariate Bernoulli model.

- Multinomial Model: In this model, what is captured is the frequencies of terms in a document by representing a document with a bag of words. The documents in each class can then be modeled as samples drawn from a multinomial word distribution. As a result, the conditional probability of a document given a class is simply a product of the probability of each observed word in the corresponding class.

No matter how we model the documents in each class (be it a multivariate Bernoulli model or a multinomial model), the component class models (i.e., generative models for documents in each class) can be used in conjunction with the Bayes rule to compute the posterior probability of the class for a given document, and the class with the highest posterior probability can then be assigned to the document.

The Naive Bayes classifier has also been extended to modeling temporally aware training data, in which the importance of a document may decay with time. As in the case of other statistical classifiers, the naïve Bayes classifier can easily incorporate domain-specific knowledge into the classification process. The particular domain that the work addresses is that of filtering junk email. Thus, for such a problem, we often have a lot of additional domain knowledge which helps us determine whether a particular email message is junk or not. For example, some common characteristics of the email which would make an email to be more or less likely to be junk are as follows:

- The domain of the sender such as .edu or .com can make an email to be more or less likely to be junk.
- Phrases such as “Free Money” or over emphasized punctuation such as “!!!” can make an email more likely to be junk.
- Whether the recipient of the message was a particular user, or a mailing list.

The Bayes method provides a natural way to incorporate such additional information into the classification process, by creating new features for each of these characteristics. The standard Bayes technique is then used in conjunction with this augmented representation for classification. The Bayes technique has also been used in conjunction with the incorporation of other kinds of domain knowledge, such as the incorporation of hyperlink information into the classification process.

The Bayes method is also suited to hierarchical classification, when the training data is arranged in taxonomy of topics. For example, the Open Directory Project (ODP), Yahoo! Taxonomy, and a variety of news sites have vast collections of documents which are arranged into hierarchical groups. The hierarchical structure of the topics can be exploited to perform more effective classification, because it has been observed that context-sensitive feature selection can provide more useful classification results. In hierarchical classification, a Bayes classifier is built at each node, which then provides us with the next branch to follow for classification purposes.

2.3.4. Linear Classifiers

Linear models for classification separate input vectors into classes using linear (hyperplane) decision boundaries.

Linear Classifiers are those for which the output of the linear predictor is defined to be p = A · X + b, where X = (x1 . . . xn) is the normalized document word frequency vector, A = (a1 . . . an) is a vector of linear coefficients with the same dimensionality as the feature space, and b is a scalar.

One characteristic of linear classifiers is that they are closely related to many feature transformation, which attempt to use these directions in order to transform the feature space, and then use other classifiers on this transformed feature space.

Simple neural networks are a form of linear classifiers, since the function computed by a set of neurons is essentially linear. The simplest form of neural network, known as the perceptron (or single layer network) are essentially designed for linear separation, and work well for text. However, by using multiple layers of neurons, it is also possible to generalize the approach for non-linear separation. Two linear classifiers: SVMs and Neural Networks are discussed below.

Support Vector Machine Classifiers

SVM Classifiers attempt to partition the data space with the use of linear or non-linear characterizations between the different classes. The key in such classifiers is to determine the optimal boundaries between the different classes and use them for the purposes of classification. That is, given labeled training data, the algorithm outputs an optimal hyperplane which categorizes new examples as depicted below.

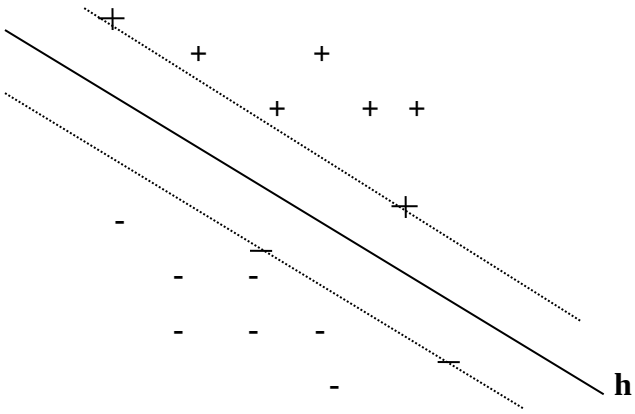


Figure 1: SVM representation

Some of the properties of text that make SVMs work for text categorization are:

- When learning text classifiers, one has to deal with very many (more than 10000) features. Since SVMs use overfitting protection, which does not necessarily depend on the number of features, they have the potential to handle these large feature spaces.
- Few irrelevant features: Feature selection tries to determine these irrelevant features. Unfortunately, in text categorization there are only very few irrelevant features. A classifier using only those "worst" features has a performance much better than random. Since it seems unlikely that all those features are completely redundant, this leads to the conjecture that a good classifier should combine many features (learn a "dense" concept) and that aggressive feature selection may result in a loss of information.

The first set of SVM classifiers, as adapted to the text domain were proposed in Thorsten Joachims (1998).

In particular, it has been shown why the SVM classifier is expected to work well under a wide variety of circumstances. This has also been demonstrated experimentally in a few different scenarios. For example, the work applied the method to email data for classifying it as spam or non-spam data. It was shown that the SVM method provides much more robust performance as compared to many other techniques such as boosting decision trees, the rule based RIPPER method, and the Rocchio method.

We note that the problem of finding the best separator is essentially an optimization problem, which can typically be reduced to a Quadratic Programming problem. For example, many of these methods use Newton's method for iterative minimization of a convex function. This can sometimes be slow, especially for high dimensional domains such as text data.

S. Dumais, et al (1998) showed that by breaking a large Quadratic Programming problem (QP problem) to a set of smaller problems, an efficient solution can be derived for the task.
S. Dumais and H. Chen (2000) used SVM successfully in the context of a hierarchical organization of the classes, as often occurs in web data. In this approach, a different classifier is built at different positions of the hierarchy.
V. Sindhwani, S. and S. Keerthi. (2006) also showed SVM to be useful in large scale scenarios in which a large amount of unlabelled data and a small amount of labelled data is available, which is essentially a semi-supervised approach because of its use of unlabelled data in the classification process.

Why SVMs work well for text categorization

The following are some of the reasons why SVMs work well for text categorization.

- **High dimensional input space:** When learning text classifiers, one has to deal with very many (more than 10000) features. Since SVMs use overfitting protection, which does not necessarily depend on the number of features, they have the potential to handle these large feature spaces.
- **Few irrelevant features:** One way to avoid these high dimensional input spaces is to assume that most of the features are irrelevant. Feature selection tries to determine these irrelevant features. Unfortunately, in text categorization there are only very few irrelevant features.
- **Document vectors are sparse:** For each document, the corresponding document vector contains only few entries which are not zero.
- **Most text categorization problems are linearly separable:** All Assumed categories are linearly. The idea of SVMs is to find such linear (or polynomial, RBF, etc.) separators.

Neural Network Classifiers

The basic unit in a neural network is a neuron or unit. Each unit receives a set of inputs, which are denoted by the vector Xi, which in this case, correspond to the term frequencies in the *ith* document. Each neuron is also associated with a set of weights A, which are used in order to compute a function f(·) of its inputs. A typical function which is often used in the neural network is the linear function as follows: $pi = A \cdot Xi$.

The goal of the neural network approach is to learn the set of weights A with the use of the training data. The idea is that we start off with random weights and gradually update them when a mistake is made by applying the current function on the training example. The magnitude of the update is regulated by a learning rate μ. This forms the core idea of the perceptron algorithm.

Some Observations about Linear Classifiers

While the different linear classifiers have been developed independently from one another in the research literature, they are surprisingly similar at a basic conceptual level. Interestingly, these different lines of work have also resulted in a number of similar conclusions in terms of the effectiveness of the different classifiers.

In Charu C. Aggarwal and ChengXiang Zhai (2012) it's noted that the main difference between the different classifiers is in terms of the details of the objective function which is optimized, and the iterative approach used in order to determine the optimum direction of separation. For example, the SVM method uses a Quadratic Programming (QP) formulation, whereas the perceptron method does not try to formulate a closed-form objective function, but works with a softer iterative hill climbing approach. This technique is essentially inherited from the iterative learning approach used by neural network algorithms. However, its goal remains quite similar to other linear methods. Thus, the differences between these methods are really at a detailed level, rather than a conceptual level, in spite of their very different research origins.

Y. Yang, L. Liu (1999) observe that all these methods can be implemented with non-linear versions of their classifiers. For example, it is possible to create non-linear decision surfaces with the SVM classifier, just as it is possible to create non-linear separation boundaries by using layered neurons in a neural network.

However, the general consensus has been that the linear versions of these methods work very well, and the additional complexity of non-linear classification does not tend to pay for itself, except for some special data sets. The reason for this is perhaps because text is a high dimensional domain with highly correlated features and small non-negative values on sparse features. On the other hand, the high dimensional nature of correlated text dimensions is especially suited to classifiers which can exploit the redundancies and relationships between the different features in separating out the different classes. Common text applications have generally resulted in class structures which are linearly separable over this high dimensional

domain of data. This is one of the reasons that linear classifiers have shown an unprecedented success in text classification.

2.3.5. Proximity-based Classifiers

Introduction

In G. Salton (1983) proximity-based classifiers essentially use distance-based measures in order to perform the classification. The main thesis is that documents which belong to the same class are likely to be close to one another based on similarity measures such as the dot product or the cosine metric.

In order to perform the classification for a given test instance, two possible methods can be used:

- We determine the k-nearest neighbors in the training data to the test instance. The majority (or most abundant) class from these k neighbors are reported as the class label. Some examples of such methods are discussed in, E.-H. Han, G. Karypis and V. Kumar (2001). The choice of k typically ranges between 20 and 40 in most of the afore-mentioned work, depending upon the size of the underlying corpus.
- We perform training data aggregation during pre-processing, in which clusters or groups of documents belonging to the same class are created. A representative meta-document is created from each group. The same k-nearest neighbor approach is applied as discussed above, except that it is applied to this new set of meta-documents (or generalized instances, W. Lam and C. Y. Ho. (1998) rather than to the original documents in the collection. A pre-processing phase of summarization is useful in improving the efficiency of the classifier, because it significantly reduces the number of distance computations. In some cases, it may also boost the accuracy of the technique, especially when the data set contains a large number of outliers. Some examples of such methods are discussed in.

A method for performing nearest neighbor classification in text data is the WHIRL. The WHIRL method is essentially a method for performing soft similarity joins on the basis of text attributes. By soft similarity joins, we refer to the fact that the two records may not be exactly the same on the joined attribute, but a notion of similarity used for this purpose. It has been observed that any method for performing a similarity-join can be adapted as a nearest neighbor classifier, by using the relevant text documents as the joined attributes.

One observation in Y. Yang (1995) about nearest neighbor classifiers was that feature selection and document representation play an important part in the effectiveness of the classification process. This is because most terms in large corpora may not be related to the category of interest.

Therefore, Y. Yang (1995) proposed a number of techniques in order to learn the associations between the words and the categories. These are then used to create a feature representation of the document, so that the nearest neighbor classifier is more sensitive to the classes in the document collection. A similar observation has been made in E.-H. Han, G. Karypis and V. Kumar (2001), in which it has been shown that the addition of weights to the terms (based on their class-sensitivity) significantly improves the underlying classifier performance. The nearest neighbor classifier has also been extended to the temporally-aware scenario in T. Salles, et al, (2010), in which the timeliness of a training document plays a role in the model construction process.

In order to incorporate such factors, a temporal weighting function has been introduced in T. Salles, et al (2010), which allows the importance of a document to gracefully decay with time.

For the case of classifiers which use grouping techniques, the most basic among such methods was proposed in J. Rocchio (1971). In this method, a single representative meta-document is constructed from each of the representative classes.

We note that the nearest neighbor classifier can be used in order to generate a ranked list of categories for each document. In cases where a document is related to multiple categories, these can be reported for the document, as long as a thresholding method is available. The work in Y. Yang (2001) studies a number of thresholding strategies for the k-nearest neighbour classifier. It has also been suggested in Y. Yang (2001) that these thresholding strategies can be used to understand the thresholding strategies of other classifiers which use ranking classifiers.

2.4. Previous comparative studies

In the realm of comparing algorithms, which was the main objective of this paper, previous studies include: work done in Thorsten Joachims (1997) comparing SVM and Naive Bayes for multivariate Bernoulli models, C4.5, Rocchio algorithm, and K-NN. From his results, SVMs performed substantially better than all other methods. Comparing training time, SVMs roughly compared to C4.5, but were more expensive than naive Bayes, Rocchio, and k-NN. Specifically, as far as SVM and Naive Bayes for multivariate Bernoulli models were

concerned, Thorsten Joachims (1997) obtained a F1-score of (.860-864) for SVM and a F1-score of 0.720 for Naive Bayes for multivariate Bernoulli models, using the 'Reuters' and the 'Ohsumed collection' datasets. Of note is, for Naïve Bayes, this project experiments with both Naive Bayes for multinomial models and Naive Bayes for multivariate Bernoulli models, and shows that Naive Bayes for multinomial models gives better scores than Naive Bayes for multivariate Bernoulli models.

In Andrew McCallum and Kamal Nigam (1998), comparison was done between Naive Bayes for multinomial models and Naive Bayes for multivariate Bernoulli models, using five datasets, and showed that multinomial model to be almost uniformly better than the multivariate Bernoulli model. From results on five real world corpora they found that the multinomial model reduced errors by an average of 27%, and sometimes by more than 50%.

In Y. Yang and L. Liu (1999), using the 'Reuters-21578' corpus, they showed from the resulting F1-scores, that SVM (Support Vector Machines), KNN (K-Nearest Neighbor) and LLSF (Linear Least Squares Fit) belong to the same class, and significantly outperform NB (Naïve Bayes) and NNet (Neural Net). That is, for SVM and Naïve Bayes, the F1-scores were 0.8599 and 0.7956 respectively.

2.5. Chapter summary

In summary, the work presented in this paper confirms on previous research to compare the performance of Naïve Bayes and SVM in the task of text classification. While earlier work focused on the performance of a particular algorithm in general, in this study we explore further the performance of each algorithm under different parameters for SVM and different versions for Naïve Bayes.

3. METHODOLOGY

3.1. Introduction

The general objective of the study was to compare the performance of SVM and Naïve Bayes in the task of text classification. This chapter describes the methodology that was used for the study. The chapter describes the data used, requirements analysis, system design, and finally the system implementation.

3.2. Dataset

This project used text documents from the '20 newsgroups' data set for performing the classification and comparison of the performance of the classifiers.

The '20 Newsgroups' data set is a collection of approximately 20,000 newsgroup documents, partitioned (nearly) evenly across 20 different newsgroups. The 20 newsgroups collection has become a popular data set for experiments in text applications of machine learning techniques, such as text classification and text clustering.

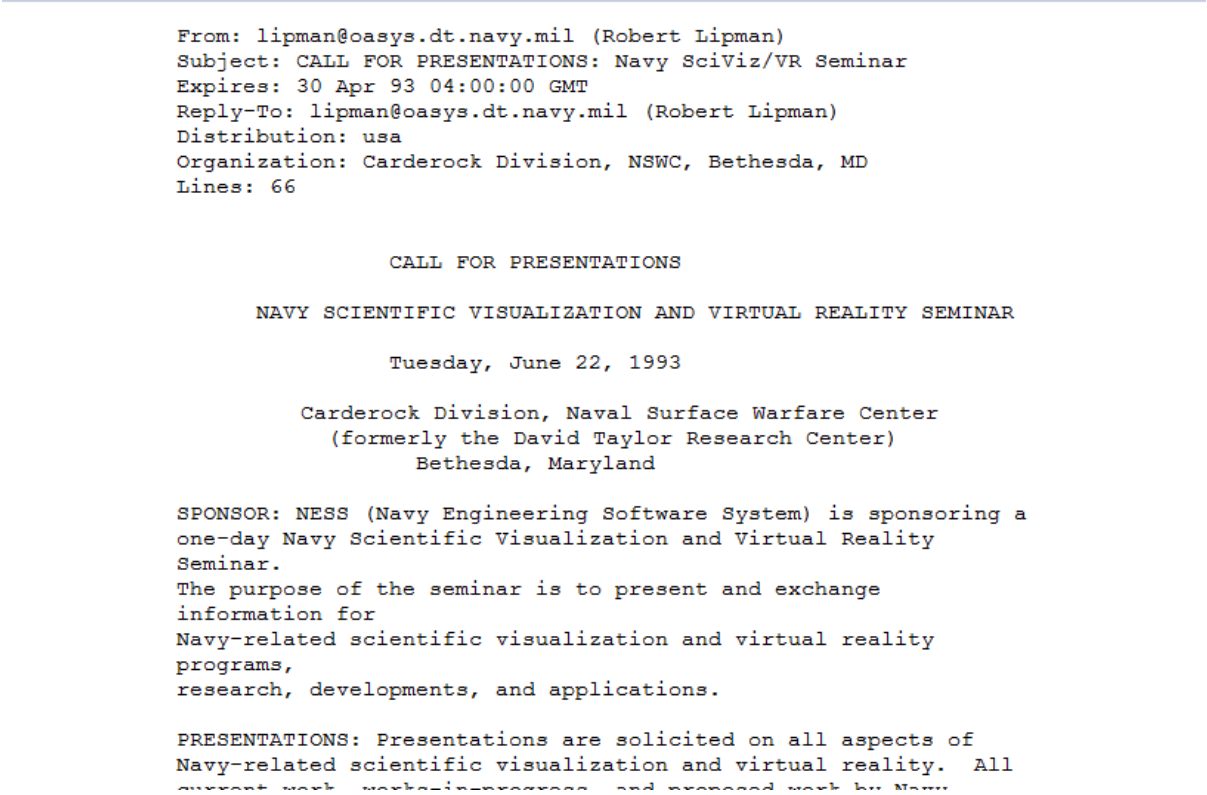
The data is organized into 20 different newsgroups, each corresponding to a different topic. Some of the newsgroups are very closely related to each other (e.g. *comp.sys.ibm.pc.hardware / comp.sys.mac.hardware*), while others are highly unrelated (e.g. *misc.forsale / soc.religion.christian*). Table 1 below shows some of the '20 newsgroups' partitioned according to subject matter:

Table 1: List of the '20 newsgroups' partitioned according to subject matter

Computers	**Science & Technology**	**Sports & Motors**	**Politics**
comp.graphics	sci.crypt	rec.autos	talk.politics.misc
comp.os.ms-windows.misc	sci.electronics	rec.motorcycles	talk.politics.guns
comp.sys.ibm.pc.hardware	sci.med	rec.sport.baseball	talk.politics.mideast
comp.sys.mac.hardware	sci.space	rec.sport.hockey	
comp.windows.x			

The data set is split into two subsets: one for training and the other one for. This is important since SVM and Naïve Bayes are supervised-learning algorithms. This means one needs to manually classify some data into the correct classes then train a SVM or Naïve Bayes model with it and eventually use it to predict unlabeled data.

Figure 2 below shows a screen shot of a sample newsgroup message under category computers.



```
From: lipman@oasys.dt.navy.mil (Robert Lipman)
Subject: CALL FOR PRESENTATIONS: Navy SciViz/VR Seminar
Expires: 30 Apr 93 04:00:00 GMT
Reply-To: lipman@oasys.dt.navy.mil (Robert Lipman)
Distribution: usa
Organization: Carderock Division, NSWC, Bethesda, MD
Lines: 66


                CALL FOR PRESENTATIONS

      NAVY SCIENTIFIC VISUALIZATION AND VIRTUAL REALITY SEMINAR

                Tuesday, June 22, 1993

          Carderock Division, Naval Surface Warfare Center
            (formerly the David Taylor Research Center)
                  Bethesda, Maryland

SPONSOR: NESS (Navy Engineering Software System) is sponsoring a
one-day Navy Scientific Visualization and Virtual Reality
Seminar.
The purpose of the seminar is to present and exchange
information for
Navy-related scientific visualization and virtual reality
programs,
research, developments, and applications.

PRESENTATIONS: Presentations are solicited on all aspects of
Navy-related scientific visualization and virtual reality.  All
```

Figure 2: Sample newsgroup message classified under comp.os.ms-windows.misc

3.3. Requirements analysis

This section describes what the system should be in a position to do. The requirements for this study were:

i. Reading the datasets, that is loading the datasets into memory in readiness for feature extraction.

ii. Preparing the data, that is labelling the data (in this project some subset of the data is labelled), extraction of features and creation of a document-term matrix.

iii. Creation and training of SVM and Naïve Bayes models, the two algorithms falling under supervised learning algorithms, already labelled data was used to train the models in readiness for them to be fed with unlabeled/testing data.

iv. Testing of the models, in order to decide whether a classification model is accurately capturing a pattern, we must evaluate that model. Using the testing subset of the dataset, the two models were tested and evaluated based on, training time, testing time, recall, precision and F-score metrics.

3.4. Systems design

This section describes the overall architecture of text classification, and an overview of the tools used in the project.

3.4.1. Conceptual design

This section describes the framework for supervised learning, specifically in classifying text. Supervised learning is the method used in this project.

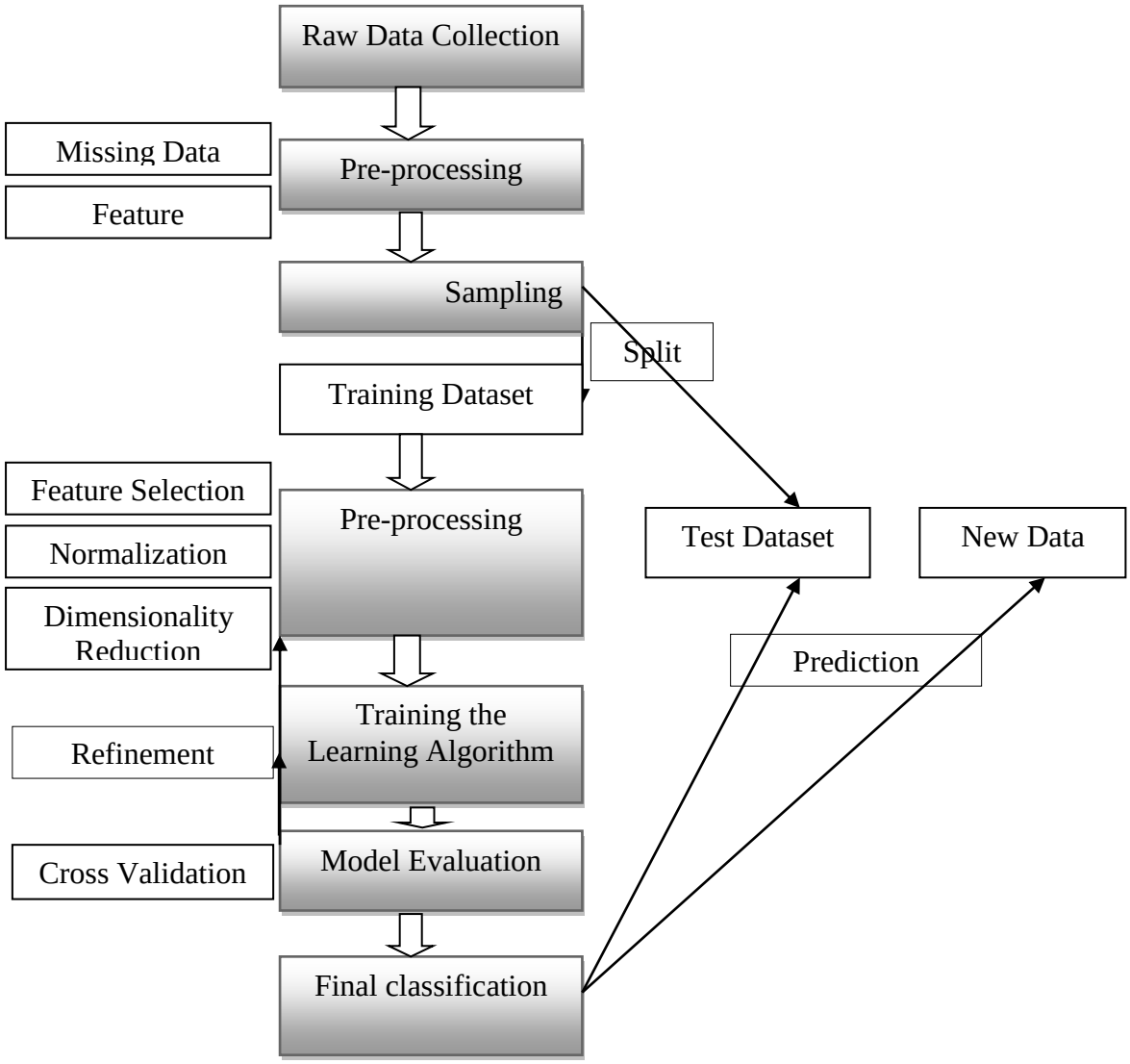


Figure 3: Text classification using supervised learning

3.4.2. Detailed design description

The design of text classifiers includes a set of steps that are universally recognized.
In this project the following steps were followed.

- **Data collection:** According to the ’20 newsgroups’ website, the ‘20 newsgroups’ dataset “was originally collected by Ken Lang”. In this project, we used the ‘20news-bydate.tar.gz’, that is ‘20 Newsgroups’ sorted by date, duplicates and some headers removed, a total of 18,846 documents. The dataset was downloaded into a local folder, from where it was loaded into memory by the python program as will be discussed in the implementation phase.
- **Pre-processing:** Because of the unstructured nature of text data, preprocessing is an essential step in text classification. There are several methods used in pre-processing text documents. Some of these are: tokenization, stop word removal and stemming. These methods allow us to transform unstructured data into a structured format that the classification algorithms can work on. In this project, this was achieved by calling Sckit-Learns ***HashingVectorizer***, which converts a collection of text documents to a matrix of token occurrences.
- **Feature weighting:** Features usually assume different roles in different documents, i.e. they can be more or less representative. To avoid these potential discrepancies it suffices to divide the number of occurrences of each word in a document by the total number of words in the document: these new features are called tf for Term Frequencies. Another refinement on top of tf is to downscale weights for words that occur in many documents in the corpus and are therefore less informative than those that occur only in a smaller portion of the corpus. This was achieved by calling Sckit-Learns ***TfidfVectorizer***, which converts a collection of raw documents to a matrix of TF-IDF features
- **Training:** In this phase, a set of training documents whose correct classifications are known is needed. The output of the learning phase is a model of one or more categories. In this project, we experimented with:
    - **Naive Bayes classifier**
      That is Naïve Bayes for multinomial models, and Naive Bayes classifier for multivariate Bernoulli models.

      In Fabian Pedregosa et al (2011) multinomial Naive Bayes classifier is suitable for classification with discrete features (e.g., word counts for text

classification). Like MultinomialNB, the BernoulliNB classifier is suitable for discrete data. The difference is that while MultinomialNB works with occurrence counts, BernoulliNB is designed for binary features.

- **Linear Support Vector Machine (SVM)**
  Which is a SVM implemented in terms of ***liblinear***. In Rong-En Fan et al (2012:1) ***LIBLINEAR*** is an open source library for large-scale linear classification. It supports logistic regression and linear support vector machines. Experiments demonstrate that LIBLINEAR is very efficient on large sparse data sets.

- **Testing:** Most evaluation techniques calculate a score for a model by comparing the labels that it generates for the inputs in a test set with the correct labels for those inputs. This test set typically has the same format as the training set. However, it is very important that the test set be distinct from the training corpus: if we simply re-used the training set as the test set, then a model that simply memorized its input, without learning how to generalize to new examples, would receive misleadingly high scores.

  The simplest metric that can be used to evaluate a classifier, accuracy, measures the percentage of inputs in the test set that the classifier correctly labeled.

  Performance evaluation of each classifier was measured by the following metrics.

  - **Training time**
    This is the time it takes to train each version of each classifier, when given different parameters, for different sizes of the dataset.

    This is the duration between the time the training of the classifiers starts, till the time the time the training ends.

  - **Testing time**
    This is the time it takes to test each version of each classifier, when given different parameters, for different sizes of the dataset.

    This is the duration between the time the testing of the classifiers starts, till the time the time the testing ends.

    Note: Both the training and testing durations did not include the time it took to extract features and the time it took to select features.

    The time to extract and select features constitutes the bulk of the processing time.

- **Precision-score**
  According to Fabian Pedregosa et al (2011) precision is the ratio **tp** / (**tp** + **fp**) where **tp** is the number of true positives and **fp** the number of false positives. The precision is intuitively the ability of the classifier not to label as positive a sample that is negative. The best value is 1 and the worst value is 0.
- **Recall-score**
  According to Fabian Pedregosa et al (2011) recall is the ratio **tp** / (**tp** + **fn**) where **tp** is the number of true positives and **fn** the number of false negatives. The recall is intuitively the ability of the classifier to find all the positive samples. The best value is 1 and the worst value is 0.
- **F1-score**
  According to Fabian Pedregosa et al (2011) F1 score can be interpreted as a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0. The relative contribution of precision and recall to the F1 score are equal. The formula for the F1 score is: F1 = 2 * (precision * recall) / (precision + recall).

3.5. System Implementation

This section describes the system implementation. It describes the approach taken in training and testing the classifiers and highlights the general stages taken to achieve the objectives of the project.

In this project, training and testing was carried out twice for the two classifiers. First by loading and training the classifiers with 2,934 documents, and testing with 1,953 documents. Then a second run, where 10,794 documents were loaded, with 6,480 documents used to train the classifiers and 4,314 documents used to test the classifiers. For each of this run, the scores of each classifier were noted down.

To achieve the above, the following steps were undertaken:

i. **Installation of the required packages**
   This was the preliminary stage of gathering and installing the required tools. These are described under development tools in section 3.6.

ii. **Reading the data**
   This stage involved loading the filenames and data from the ‘20 newsgroups’ dataset, using ***‘fetch_20newsgroups’*** sckit-learn modules, which returns a list of the raw texts that will then be fed to text feature extractors. This list of files is stored in a local

drive specified or a default storage location. In this project, the following parameters were specified in the fetch stage:

a. Subset: 'train' or 'test'

To specify the dataset to load: 'train' for the training set, 'test' for the test set, with shuffled ordering.

b. Categories

To specify the number of documents to load. In this case, 4,887 and 10,794 documents for each run were loaded.

c. Remove

To remove 'headers', 'footers' and 'quotes'. This helps in preventing the classifiers from overfitting on data stored in those regions.

iii. Data preparation

a. Tokenizing text

Using Sckit-Learns ***HashVectorizer***, text pre-processing, tokenizing and filtering of stop words were included in a high level component that is able to build a dictionary of features and transform documents to feature vectors:

b. Getting word frequencies

To get the document frequency, ***tf*** and ***tf–idf*** were computed by calling Sckit-Learns ***TfidfTransformer***.

iv. Training the classifiers

With the features in place, the two classifiers were trained. This step made use of ***sklearn.naive_bayes.BernoulliNB*** and ***sklearn.naive_bayes.MultinomialNB*** for Naïve Bayes and ***sklearn.svm.LinearSVC*** for SVM.

v. Testing the resulting models

Testing and results are discussed in the next chapter.

3.6. Development tools

The tools used in the project were:

i. **Python programming language:** in this case version 2.7.

ii. **Scikit-learn:** an open source machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, logistic regression, naive Bayes, random forests, gradient boosting, k-means and DBSCAN, and is

designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

iii. **NumPy:** the fundamental package for scientific computing with Python.

iv. **SciPy:** an open source library of scientific tools.

v. **Matplotlib:** a python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms.

vi. **Spyder:** an interactive development environment for the Python language with advanced editing, interactive testing, debugging and introspection features.

3.7. Chapter summary

This chapter discussed the data used, the tools used, the flow of arriving at the desired results, and the evaluation metrics for the models.

4. TESTING AND RESULTS

4.1. Introduction

In order to decide whether a classification model is accurately capturing a pattern, we must evaluate that model. The result of this evaluation is important for deciding how trustworthy the model is, and for what purposes we can use it. Evaluation can also be an effective tool for guiding us in making future improvements to the model.

Most evaluation techniques calculate a score for a model by comparing the labels that it generates for the inputs in a test set with the correct labels for those inputs. This test set typically has the same format as the training set. However, it is very important that the test set be distinct from the training corpus: if we simply re-used the training set as the test set, then a model that simply memorized its input, without learning how to generalize to new examples, would receive misleadingly high scores.

The simplest metric that can be used to evaluate a classifier, accuracy, measures the percentage of inputs in the test set that the classifier correctly labeled.

Performance evaluation of each classifier was undertaken by measuring the following metrics:

- Training time
- Testing time
- Precision-score
- Recall-score
- F1-score

4.2. Dataset

Testing was carried out in two runs, in the first run, 1,953 documents (5 categories of the dataset) were loaded, and in the second run, 4,314 documents (11 categories of the dataset) were loaded, and used to test the classifiers. For each of this run, the scores of each classifier were noted down.

4.3. Setup

- **Naïve Bayes**
  For Naïve Bayes, the two versions of Naïve Bayes classifier: MultinomialNB and BernoulliNB were fed with the testing data.

- **SVM**
  For SVM, testing was done by setting different parameters of LinearSVC: ***LinearSVC*** with ***penalty 'l1'*** and ***LinearSVC*** with ***penalty 'l2'***.

4.4. Data preparation

This section highlights the time taken in data preparation, which is the time taken to extract features from the training and testing dataset. The results for the two runs are shown below.

- **Extracting features from the training dataset**
  The table below shows the time in seconds it took to extract features from the training dataset.

  Table 2: Time in seconds taken to extract features from the training dataset

5 Categories / 2,934 documents		**11 Categories / 6,480 documents**	
Time in Seconds	**MBs per second**	**Time in Seconds**	**MBs per second**
2.588000s	2.350MB/s	5.370000s	2.414MB/s

- **Extracting features from the testing dataset**
  The table below shows the time in seconds it took to extract features from the testing dataset.

  Table 3: Time in seconds taken to extract features from the testing dataset

5 Categories / 1,953 documents		**11 Categories / 4,314 documents**	
Time in Seconds	**MBs per second**	**Time in Seconds**	**MBs per second**
1.649000s	2.631MB/s	3.077000s	2.679MB/s

4.5. Results

- **4,887 documents loaded**

  The table below shows performance of each classifier, in regards to training time, testing time, F1-Score, precision and recall scores when run against 4,887 documents, i.e. 2,934 documents used for training and 1,953 documents used for testing.

Table 4: Performance for each classifier version when 4,887 documents are loaded

5 Categories / 4,887 documents				
Metric	**Linear SVC-L2 Penalty**	**Linear SVC-L1 Penalty**	**BernoulliNB**	**MultinomialNB**
Training time	0.676s	0.814s	0.034s	0.026s
Testing Time	0.006s	0.006s	0.027s	0.007s
F1-score	0.969	0.942	0.937	0.964
Precision score	0.969	0.943	0.940	0.964
Recall score	0.969	0.942	0.937	0.964

- **10,794 documents loaded**

The table below shows performance of each classifier, in regards to training time, testing time, F1-Score, precision and recall scores when run against 10,794 documents, i.e. 6,480 documents used for training and 4,314 documents used for testing.

Table 5: Performance for each classifier version when 10,794 documents are loaded

11 Categories / 10,794 documents				
Metric	**Linear SVC-L2 Penalty**	**Linear SVC-L1 Penalty**	**BernoulliNB**	**MultinomialNB**
Training time	3.733s	4.425s	0.124s	0.106s
Testing Time	0.019s	0.019s	0.108s	0.028s
F1-score	0.900	0.873	0.773	0.869
Precision score	0.901	0.876	0.822	0.872
Recall score	0.900	0.873	0.794	0.870

4.6. Comparison of the classifiers

- **4,887 documents loaded**

For 4,887 documents, as shown in table 4.4, SVM takes longer time to train than Naïve Bayes. For F1, precision, and recall scores. Linear SVM with penalty 'L2' performs better than any of the Naïve Bayes versions and better than Linear SVM with penalty 'L1'. Of note is, Multinomial Naïve Bayes has better scores than Linear SVM with penalty 'L1'.

- **10,794 documents loaded**

  For 10,794 documents, as shown in table 4.5, both versions of Naïve Bayes take much less time to train than Linear SVM. On the other hand, both versions of Naïve Bayes take longer time during testing than Linear SVM. For F1, precision, and recall scores. Linear SVM with penalty ‘L2’ performs much better than any of Naïve Bayes versions and better than Linear SVM with penalty ‘L1’. In this setup, Linear SVM with either ‘L1’ or ‘L2’ penalties has better scores than both versions of Naïve Bayes. From the results, as the number of documents increases, SVM as previous studies have shown, performs better than Naïve Bayes in classifying text.

5. DISCUSSION

5.1. Recap of the project

The project aimed at comparing the performance of Naïve Bayes and SVM classifiers as applied in the task of text classification. This was accomplished by using the ’20 Newsgroup’ dataset. The classifiers were evaluated by comparing their training times, testing times, F-score, precision and recall scores, when the two classifiers are run against different sizes of data.

5.2. Discussion

In this discussion, we’re going to highlight the performance differences between Linear SVC with penalty ‘L2’ and MultinomialNB. This is because the two versions were the better performers as per the results.

For 4,887 documents, the difference between the two classifiers was 0.005, that is, an F-score of 0.969 for Linear SVC and an F-score of 0.964 for MultinomialNB. For the same number of documents, Naïve Bayes took shorter times to train, by taking 0.026s against 0.676s for SVM. Testing time for SVM was slightly better than Naïve Bayes, with 0.006s and 0.007s respectively.

For 10,794 documents, the difference between the two classifiers was 0.031 (a figure that is six times the previous score), That is, an F-score of 0.900 for Linear SVC and an F-score of 0.869 for MultinomialNB. For the same number of documents, Naïve Bayes took shorter times to train, by taking 0.106s against 3.733s for SVM. Testing time for SVM was slightly better than that for Naïve Bayes, with 0.019s and 0.028s respectively.

From the results above, as the number of documents increased, the differences in performance between the two classifiers grew apart. This was the case too for the training times. The difference in testing times seem not to be that large, but that can probably be attributed to the relatively small differences in the number of testing documents used in each run.

5.3. Achievements

The main objective of this research was to evaluate and compare the performance of Support Vector Machines versus Naïve Bayes in the task of text classification. This was achieved by assembling tools provided in Sckit-learn into a Python script/program that was then run

against the ’20 newsgroup dataset’ and different performance criteria (F-score, precision, and recall scores) for each classifier captured. From the results, SVM did well in classifying text than Naïve Bayes.

The other objective was to experiment with different feature sets of the two classifiers. This was achieved by passing two different parameters to Linear SVC, a linear implementation of SVM. The two parameters in this case were, ‘L1’ and ‘L2’ for the penalty parameter. From the results, with ‘L2’ penalty Linear SVC had better scores than when it was passed ‘L1’ parameter as the penalty. For Naïve Bayes, this objective was achieved by comparing different implementations of Naïve Bayes, i.e. MultinomialNB and BernoulliNB. From the results, MultinomialNB had much better scores than BernoulliNB.

The other target objective was to examine the classifier learning abilities for an increasing number of documents. This was achieved by running the algorithms twice, first against 4,887 documents and the second run against 10,794 documents. For each of the runs, the scores were output for analysis.

5.4. Limitations

One of the shortcomings of the project was the fact that we didn’t use another dataset, which would have brought a clearer picture of the performance between the two classifiers. More of a challenge was assembling the program to do the actual work of pre-processing, training and testing the classifiers. This was overcome by utilizing the libraries and documentation provided by Sckit-learn project.

5.5. Recommendations for further study

In the future, it would be interesting to investigate what makes each of the classifiers perform as they do, may it be from the scores each classifier achieves in classifying the text, or the resources it consumes; time, and computer memory during training and testing.

Investigate the impact of feature extraction and representation on the performance of each classifier. E.g. Multinomial Naïve Bayes in this study performs much better than Multivariate Naïve Bayes because of the bag of words approach used to represent the documents.

Preparation of a local dataset. This would lead to invaluable lessons and insight from the process of preparing data for machine learning tasks, which constitutes a big part of automating classification tasks. Especially an investigation of effects of pre-processing methods on classification of Swahili text.

5.6. Conclusion

From this study, we found that as previous studies have shown, SVM does very well in text classification. It should be noted that Naïve Bayes does achieve very good scores as well. In short, given a text classification problem, any of the two could be used without a big compromise on the classification accuracy. In addition, factoring in training time, in some cases it would be advised to use Naïve Bayes instead of SVM.

REFERENCES

1) Andrew McCallum and Kamal Nigam. (1998) *A comparison of event models for naive Bayes text classification*, AAAI-98 Workshop on Learning for Text Categorization.
2) C. C. Aggarwal, S. C. Gates and P. S. Yu. (2004) *On Using Partial Supervision for Text Categorization*, IEEE Transactions on Knowledge and Data Engineering, 16(2), 245–255.
3) C. Cortes and V. Vapnik. (1995) *Support-vector networks*, Machine Learning-20: pp. 273–297.
4) Charu C. Aggarwal and ChengXiang Zhai. (2012) *Mining Text Data*, Springer.
5) Chee-Hong Chan, Aixin Sun and Ee-Peng Lim. (2001) *Automated Online News Classification with Personalization*, Nanyang Technological University: Center for Advanced Information Systems.
6) D. Hardin, I. Tsamardinos and C. Aliferis. (2004) *A theoretical characterization of linear SVM-based feature selection*, ICML Conference.
7) D. Lewis. (1992) *An Evaluation of Phrasal and Clustered Representations for the Text Categorization Task*, ACM SIGIR Conference.
8) D. Mladenic, J. Brank, M. Grobelnik and N. Milic-Frayling. (2004) *Feature selection using linear classifier weights: interaction with classification models*, ACM SIGIR Conference.
9) E. Wiener, J. O. Pedersen and A. S. Weigend. (1995) *A Neural Network Approach to Topic Spotting*, SDAIR, pp. 317–332.
10) E.-H. Han and G. Karypis. (2000) *Centroid-based Document Classification: Analysis and Experimental Results*, PKDD Conference.
11) E.-H. Han, G. Karypis and V. Kumar. (2001) *Text Categorization using Weighted-Adjusted k-nearest neighbor classification*, PAKDD Conference.
12) Fabian Pedregosa et al, (2011) *Scikit-learn: Machine Learning in Python*, Journal of Machine Learning Research.
13) G. Karypis and E.-H. Han. (2000) *Fast Supervised Dimensionality Reduction with Applications to Document Categorization and Retrieval*, ACM CIKM Conference.
14) George Forman. (2003) *An Extensive Empirical Study of Feature Selection Metrics for Text Classification*, pp. 1-1.

15) H. Raghavan and J. Allan. (2007) *An interactive algorithm for asking and incorporating feature feedback into support vector machines*, ACM SIGIR Conference.

a. T. Jolliffee. (2002) *Principal Component Analysis*, Springer.

16) J.-T. Sun, Z. Chen, H.-J. Zeng, Y. Lu, C.-Y. Shi and W.-Y. Ma. (2004) *Supervised Latent Semantic Indexing for Document Categorization*, ICDM Conference.

17) K. Dalal and A. Zaveri. (2011) *Automatic Text Classification: A Technical Review*.

18) Kamal Nigam, Andrew McCallum, Sebastian Thrun, and Tom Mitchell. (1999) *Text classification from labeled and unlabeled documents using EM*.

19) L. Baker and A. McCallum. (1998) *Distributional Clustering of Words for Text Classification,* ACM SIGIR Conference.

20) L. Cai and T. Hofmann. (2003) *Text categorization by boosting automatically extracted concepts,* ACM SIGIR Conference.

21) N. Slonim and N. Tishby. (2001) *The power of word clusters for text classification*, European Colloquium on Information Retrieval Research (ECIR).

22) R. Bekkerman, R. El-Yaniv, Y. Winter and N. Tishby. (2001) *On Feature Distributional Clustering for Text Categorization*, ACM SIGIR Conference.

23) Rong-En Fan et al, (2012) *LIBLINEAR: A Library for Large Linear Classification.*

24) S. Chakrabarti, S. Roy and M. Soundalgekar. (2003) *Fast and Accurate Text Classification via Multiple Linear Discriminant Projections*, VLDB Journal, 12(2), pp. 172–185.

25) S. Chakraborti, R. Mukras, R. Lothian, N. Wiratunga, S. Watt and D. Harper. (2007) *Supervised Latent Semantic Indexing using Adaptive Sprinkling*, IJCAI.

26) S. Deerwester, S. Dumais, T. Landauer, G. Furnas and R. Harshman. (1990) *Indexing by Latent Semantic Analysis*, JASIS, pp. 391–407.

27) S. Dumais and H. Chen. (2000) *Hierarchical Classification of Web Content*, ACM SIGIR Conference.

28) S. Dumais, J. Platt, D. Heckerman and M. Sahami. (1998) *Inductive learning algorithms and representations for text categorization*, CIKM Conference.

29) T. Hofmann. (1999) *Probabilistic latent semantic indexing*, ACM SIGIR Conference.

30) T. Salles, L. Rocha, G. Pappa, G. Mourao, W. Meira Jr. and M.Goncalves. (2010) *Temporally-aware algorithms for document classification*, ACM SIGIR Conference.

31) Thorsten Joachims. (1998) *Text categorization with Support Vector Machines: Learning with many relevant features*, Tenth European Conference on Machine Learning.
32) V. Sindhwani and S. S. Keerthi. (2006) *Large scale semi-supervised linear SVMs*, ACM SIGIR Conference.
33) W. Lam and C. Y. Ho. (1998) *Using a generalized instance set for automatic text categorization*, ACM SIGIR Conference.
34) Y. Li and A. Jain. (1998) *Classification of text documents*, The Computer Journal, 41(8), pp. 537–546.
35) Y. Yang and L. Liu. (1999) *A re-examination of text categorization methods*, ACM SIGIR Conference.
36) Y. Yang. (1995) *Noise Reduction in a Statistical Approach to Text Categorization*, ACM SIGIR Conference.
37) Y. Yang. (2001) *A Study on Thresholding Strategies for Text Categorization*, ACM SIGIR Conference.
38) Yiming Yang. (1997) *An Evaluation of Statistical Approaches to Text Categorization*, Journal of Information Retrieval.

APPENDICES

Source Code

```
#Classification of text documents#
#Authour: Muchai, with references from Sckit-Learn#

from __future__ import print_function

import logging
import numpy as np
from optparse import OptionParser
import sys
from time import time
import matplotlib.pyplot as plt

from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import HashingVectorizer
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.svm import LinearSVC
#from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import BernoulliNB,MultinomialNB
from sklearn.utils.extmath import density
from sklearn import metrics


################################################################################
#Load some categories from the training set
if opts.all_categories:
categories = None
else:
categories = [
'comp.graphics',
'rec.autos',
'sci.space',
'soc.religion.christian',
'talk.politics.mideast',
'sci.med',
'comp.os.ms-windows.misc',
'comp.sys.ibm.pc.hardware',
'comp.sys.mac.hardware',
'comp.windows.x',
'rec.sport.hockey',
]
if opts.filtered:
remove = ('headers', 'footers', 'quotes')
else:
remove = ()
print("Loading 20 newsgroups dataset for categories:")
```

```
print(categories if categories else "all")

data_train = fetch_20newsgroups(subset='train', categories=categories,
shuffle=True, random_state=42,
remove=remove)
data_test = fetch_20newsgroups(subset='test', categories=categories,
shuffle=True, random_state=42,
remove=remove)
print('data loaded')

categories = data_train.target_names #for case categories == None

def size_mb(docs):
return sum(len(s.encode('utf-8')) for s in docs) /1e6
data_train_size_mb = size_mb(data_train.data)
data_test_size_mb = size_mb(data_test.data)

print("%d documents - %0.3fMB (training set)" % (
len(data_train.data), data_train_size_mb))
print("%d documents - %0.3fMB (test set)" % (
len(data_test.data), data_test_size_mb))
print("%d categories" % len(categories))
print()

# split a training set and a test set
y_train, y_test = data_train.target, data_test.target

print("Extracting features from the training dataset using a sparse vectorizer")
t0 = time()
if opts.use_hashing:
vectorizer = HashingVectorizer(stop_words='english', non_negative=True,
n_features=opts.n_features)
X_train = vectorizer.transform(data_train.data)
else:
vectorizer = TfidfVectorizer(sublinear_tf=True, max_df=0.5,
stop_words='english')
X_train = vectorizer.fit_transform(data_train.data)
duration = time() - t0
print("done in %fs at %0.3fMB/s" % (duration, data_train_size_mb / duration))
print("n_samples: %d, n_features: %d" % X_train.shape)
print()

print("Extracting features from the test dataset using the same vectorizer")
t0 = time()
X_test = vectorizer.transform(data_test.data)
duration = time() - t0
print("done in %fs at %0.3fMB/s" % (duration, data_test_size_mb / duration))
print("n_samples: %d, n_features: %d" % X_test.shape)
print()
```

```
if opts.select_chi2:
print("Extracting %d best features by a chi-squared test" %
opts.select_chi2)
t0 = time()
ch2 = SelectKBest(chi2, k=opts.select_chi2)
X_train = ch2.fit_transform(X_train, y_train)
X_test = ch2.transform(X_test)
print("done in %fs" % (time() - t0))
print()
def trim(s):
"""Trim string to fit on terminal (assuming 80-column display)"""
return s if len(s) <= 80 else s[:77] + "..."
# mapping from integer feature name to original token string
if opts.use_hashing:
feature_names = None
else:
feature_names = np.asarray(vectorizer.get_feature_names())

###############################################################################
#Benchmark classifiers
def benchmark(clf):
print('_' * 80)
print("Training: ")
print(clf)
t0 = time()
clf.fit(X_train, y_train)
train_time = time() - t0
print("train time: %0.3fs" % train_time)
t0 = time()
pred = clf.predict(X_test)
test_time = time() - t0
print("test time: %0.3fs" % test_time)
#F-score
score = metrics.f1_score(y_test, pred)
print("f1-score: %0.3f" % score)
#Precision score
pscore = metrics.precision_score(y_test, pred)
print("precision-score: %0.3f" % pscore)
#Recall
rscore = metrics.recall_score(y_test, pred)
print("recall-score: %0.3f" % rscore)
#Accuracy
#ascore = metrics.accuracy_score(y_test, pred)
#print("accuracy-score: %0.3f" % ascore)
if hasattr(clf, 'coef_'):
print("dimensionality: %d" % clf.coef_.shape[1])
print("density: %f" % density(clf.coef_))
if opts.print_top10 and feature_names is not None:
print("top 10 keywords per class:")
for i, category in enumerate(categories):
```

```
top10 = np.argsort(clf.coef_[i])[-10:]
print(trim("%s: %s"
% (category, " ".join(feature_names[top10]))))
print()
if opts.print_report:
print("classification report:")
print(metrics.classification_report(y_test, pred,
target_names=categories))
if opts.print_cm:
print("confusion matrix:")
print(metrics.confusion_matrix(y_test, pred))
print()
clf_descr = str(clf).split('(')[0]
return clf_descr, score, pscore, rscore, train_time, test_time
results = []
for penalty in ["l2", "l1"]:
print('=' * 80)
print("%s penalty" % penalty.upper())
# Train Liblinear model
results.append(benchmark(LinearSVC(loss='l2', penalty=penalty,
dual=False, tol=1e-3)))
# Train sparse Naive Bayes classifiers
print('=' * 80)
print("Naive Bayes")
results.append(benchmark(MultinomialNB(alpha=.01)))
results.append(benchmark(BernoulliNB(alpha=.01)))

# make some plots

indices = np.arange(len(results))

results = [[x[i] for x in results] for i in range(6)]

clf_names, score, pscore, rscore, training_time, test_time = results
training_time = np.array(training_time) / np.max(training_time)
test_time = np.array(test_time) / np.max(test_time)

plt.figure(figsize=(12, 9))
plt.title("Scores, comparison of SVM and Naive Bayes Classifiers using 20 Newsgroups Dataset")
plt.barh(indices, score, .15, label="f1-score", color='r')
plt.barh(indices + .15, pscore, .15, label="precision score", color='y')
plt.barh(indices + .3, rscore, .15, label="recall score", color='c')
plt.barh(indices + .45, training_time, .15, label="training time", color='g')
plt.barh(indices + .6, test_time, .15, label="test time", color='b')
plt.yticks(())
plt.legend(loc='best')
plt.subplots_adjust(left=.25)
plt.subplots_adjust(top=.95)
plt.subplots_adjust(bottom=.05)
```

```
for i, c in zip(indices, clf_names):
plt.text(-.15, i, c)
plt.show()
```