nigeria computer society ncs www.ncs.org.ng

THE JOURNAL OF COMPUTER SCIENCE AND ITS APPLICATIONS
Vol. 24, No 2, December, 2017

COMPARATIVE ANALYSIS OF TEXT CATEGORIZATION ALGORITHMS

[1]A. P. Adewole and [2]D. M. Omitiran
[1,2]Department of Computer Science, University of Lagos, Lagos, Nigeria
[1]philipwole@yahoo.com; [2]omitirand@gmail.com

ABSTRACT

Text categorization (also known as text classification) is the task of automatically assigning documents to a category (or categories) from a pre-specified set. This task has several applications, including spam filtering, identification of document genre, automated indexing of scientific articles according to a predefined thesauri of technical terms, and even the automated extraction of metadata. The importance of text categorization cannot be overemphasized due to the fact that unstructured texts are the largest readily available source of data and manual organization of this data is infeasible due to the large number of documents involved as well as time constraints. The accuracy of modern text categorization machines rivals that of trained human professionals. This study experimentally compared four machine learning classifiers used in text categorization. These algorithms are; Naïve Bayes, Decision trees, k-Nearest Neighbour (kNN) and Support Vector Machines (SVM). These classifiers were developed using Python programming language. When run on the Reuters dataset, SVM significantly outperforms Naïve Bayes, kNN and Decision Trees. Decision trees performed worst of the four algorithms considered in this study. From observations made during the course of running these experiments, there seems to be a trade-off between simplicity and effectiveness. In conclusion, the results of this comparative analysis prove that SVM is the most effective of the classifiers considered in this study.

Keywords: classifier, Decision trees, k-Nearest Neighbour (kNN), Machine learning ,Naïve Bayes, Support Vector Machines (SVM), text categorization, text classification

1.0 INTRODUCTION

The increase in the number of documents stored in electronic form and the resulting need to assess them in flexible ways has led to an increase in the significance of text mining. Over 80% of electronic data is in the form of text [6]. Thus, unstructured texts are the largest readily available sources of knowledge. If these textual data are properly organized, then retrieval as well as analysis can be greatly simplified. The Internet has led to the exponential growth of these documents, as well as the inevitable need of automatic

methods for the purpose of organizing and classifying them.

The most common theme in the analysis of complex data (structured or not) is the classification, or categorization, of elements. Defined abstractly, this is the association of a given data instance to a pre-specified set of categories. In the context of document management, categorization (more specifically *text categorization*) refers to the assignment of documents to a set of given categories based on the textual content of each of the documents. In other words, text categorization (TC- a.k.a. text classification) is the process of finding the correct category (or categories) from a given set of categories for each document in the document collection.

There are two main approaches to text categorization. The first is the *knowledge engineering* approach, in which the expert's knowledge about the categories (and the classification of documents into these categories) is manually encoded into the system either declaratively or in the form of procedural classification rules. The knowledge engineering approach was used in the development of real-world TC applications until the 1990s, a time in which this approach lost popularity (especially in the research community) to the second approach, the *machine learning* approach [2]. The machine learning approach involves building a classifier by learning from a set of pre-classified examples via a general inductive process.

The general text categorization task can be formally defined as the task of approximating an unknown category assignment function $F: D \times C \rightarrow \{0, 1\}$, where D is the set of all possible documents and C is the set of pre-defined categories [5]. The value of $F(d, c)$ for $d \in D$ and $c \in C$ is 1 if the document d belongs to the category c and 0 otherwise.

The task of the machine learning approach to text categorization is to build the approximating function $M: D \times C \rightarrow \{0, 1\}$ that produces results as "close" as possible to the true category assignment function F. This approximating function M is called the *classifier*.

The machine learning approach uses a *training set* and a *test set* for classification. The training set contains input feature vectors and their corresponding category labels. A classification model is built using the training set; the model attempts to classify the input feature vectors into corresponding category labels. Then a test set is used to validate the model by predicting the categories of feature vectors whose category labels are unseen.

This study focuses on the evaluation of different algorithms under the machine learning approach to text categorization, namely: Naïve Bayes, Decision Trees, k-Nearest Neighbour (kNN) and Support Vector Machines (SVM).

2.0 METHODOLOGY

2.1 Dataset Preparation

For this study, the Reuters-21578 dataset compiled by David Lewis was used. Lewis's standard *modApté* train/test split was also used. The documents in this dataset appeared on the Reuters newswire in 1987 and were manually classified by personnel from Reuters Ltd. The dataset consists of 12902 documents, with the *modApté* split leading to 9603 training documents and 3299 test documents. The dataset consists of 135 categories, of which only 90 possess at least one training and one test example.

The dataset is a multi-label dataset. This work is focused on single-label categorization; thus only documents belonging to a single category are included. Some categories have only multi-class documents belonging to them, and thus are also excluded as well. Categories with less than 30 training examples were excluded as well.

This gives us 6241 training documents and 2408 test documents, with 21 categories. The training documents are used for training the model, whereas the test set is used for evaluating the performance of the classifiers.

.

Table 1: Experimental Dataset: Reuters-21578

CATEGORY	TRAINING DOCS	TEST DOCS	TOTAL
acq	1596	696	2292
alum	31	19	50
cocoa	46	15	61
coffee	90	22	112
copper	31	13	44
cpi	54	17	71
crude	253	121	374
earn	2840	1083	3923
gnp	59	15	74
gold	70	20	90
grain	41	10	51
interest	191	81	272
ipi	34	11	45
jobs	37	12	49
money-fx	222	87	309
money-supply	123	28	151
reserves	37	12	49
rubber	31	9	40
Ship	108	36	144
Sugar	97	25	122
Trade	250	76	326
TOTAL	6241	2408	8649

The machine learning approach constructs a model from the training set through the use of the aforementioned algorithms. This model is then used to classify the documents in the test set.

The methodology for the machine learning approach used in this work can be described with the following steps:

1. Pre-processing: Before the classifiers could be trained, the documents needed to be pre-processed. This step was performed by punctuation removal, stop word removal, stemming and tokenization.
2. Feature Generation: The documents were transformed into a bag-of-words model, and the features used were the tokens gotten from pre-processing.
3. Feature Selection: A feature space of 16084 features was produced after feature generation was carried out. Dimensionality reduction had to be performed, and the best features were to be selected so as to improve classifier performance. For this reason, information gain was used. The top 200 features, ranked according to information gain, were selected.
4. Training and Testing: Each of the documents were represented by the selected features, the values of which were dependent on the algorithm used. In this format, the documents were used for classifier training and testing, and the resulting metrics were computed.

2.2 **Naïve Bayes**

The Naïve Bayes algorithm is a probabilistic classifier. In this algorithm, the categorization

status value CSV(d, c) is calculated as the probability P(c | d) that the document belongs to a category. This probability is computed by the application of Bayes' theorem:

$$P(c|d) = \frac{P(d|c)P(c)}{P(d)} \quad (1)$$

There is no need to calculate the document probability P(d) because it is constant for all categories. Thus the equation can be reduced to:

$$P(c|d) = P(d|c)P(c) \quad (2)$$

Where *P(c)* = the independent probability of *c* (also known as the *prior probability*)

P(d|c) = the conditional probability of *c* given *d* (also known as the *likelihood*)

P(c|d) = the conditional probability of *d* given *c* (also known as the *posterior probability*)

The prior probability *P(c)* is the probability that a document in the training set belongs to category *c*. Thus if N_C is the number of training documents that belong to the category *c* and *N* is the total number of training documents, then the prior probability is

$$P(c) = \frac{N_C}{N} \quad (3)$$

Since each document is represented as a set of feature vectors (w_1, w_2, … , w_k), then the *likelihood* is calculated as

$$P(d|c) = \prod_{i}^{k} P(w_i|c) \quad (4)$$

The probability $P(w_i|c)$ is the probability that a feature w_i appears in the category*c*. The multinomial Naïve Bayes model was implemented in this study. This means that

$$P(w_i|c) = \frac{count(w_i, c)}{count(w, c)} \quad (5)$$

Where *count(w_i, c)* = number of times w_i appears in all documents belonging to category *c*,

count(w, c) = total number of words in all documents belonging to category *c*.

The problem with equation (5) is that it does not consider the fact that some of these features may not occur in the documents belonging to some categories. Going by this equation, if a word from a document *d* does not appear in the documents belonging to a particular category *c*, then the document *d* does not belong to category *c*. This problem is called the *Zero Probability Problem*.

A simple way to alleviate this problem is called *add one smoothing*. As the name implies, a count of one is added to each feature count. The denominator is also increased as well so as to ensure that the probabilities are normalised.

$$P(w_i|c) = \frac{count(w_i, c) + 1}{count(w, c) + length(vocabulary)} \quad (6)$$

The training stage simply involved the calculation and storage of the prior probabilities *P(c)* and likelihoods $P(w_i|c)$for all categories and features. Classification was then performed by the equation below:

$$c_{NB} = argmax_{c \in C}\, P(c) \prod_{i}^{k} P(w_i|c) \qquad (7)$$

The multinomial model of the Naïve Bayes algorithm was implemented; the term-frequency (TF) scheme was used for assigning weights to each feature in the feature vector.

2.3 **Decision Tree Classifiers**

A decision tree (DT) classifier is a tree in which the internal nodes are labelled by the features, the edges leaving the nodes are labelled by tests on the feature's weight, and the leaves are labelled by categories. A DT categorizes a document by starting from the root of the tree and moving successively downwards via branches whose conditions are satisfied by the document until a leaf node is reached. The document is then assigned to the category that labels this leaf node. Most of the DT classifiers use a binary representation, and thus the trees are binary.

Typically, the tree is built recursively by selecting a feature *f* at each step and splitting the training collection into two sub collections, one containing *f* and the other not containing *f*, until only documents of a single category remain – at which point a leaf node is generated. The key step is the choice of a feature at each step, and this choice is made by some information-theoretic measure.

The ID3 algorithm was used for the construction of the decision tree. ID3 uses *information gain* as its splitting criterion. In other words, ID3 uses information gain to select a feature on which to split at every step while growing the tree. Information gain measures how well a feature splits the training examples according to their target category.

Information gain cannot be precisely defined without defining an information theory measure called *entropy*. Entropy is the degree of impurity of an arbitrary collection of training examples. The entropy of a collection *D* is given as:

$$H(D) = -\sum_{i}^{|C|} P(c_i) \cdot \log P(c_i) \qquad (8)$$

Where $P(c_i) = probability\ that\ a\ document\ in\ collection\ D\ belongs\ to\ category\ c_i$

Note that if $P(c_i) = 0$, then$P(c_i) \cdot \log P(c_i) = 0$. If the entropy of a document collection is zero, then it means the documents are homogeneous, that is, all the documents in the collection belong to only one category.

In terms of entropy, the information gain of a feature was calculated as:

$$IG(w) = H(D) - H(D|w) \qquad (9)$$

Where $H(D) = Entropy\ of\ document\ collection\ D, and$

$$H(D|w) = Entropy\ of\ document\ collection\ D\ after\ splitting\ on\ feature\ w.$$

A binary representation of the features were used in the feature vector space for the construction of the decision tree. This resulted in a binary tree, in which splitting on a feature *w* results in two subsets of the original collection: documents containing *w* and documents that do not contain *w*.

$$H(D|w) = -P(w)\sum_{i}^{|C|} P(c_i|w) \cdot \log P(c_i|w) - P(\overline{w})\sum_{i}^{|C|} P(c_i|\overline{w}) \cdot \log P(c_i|\overline{w}) \qquad (10)$$

Thus information gain is calculated as

$$IG(w) = -\sum_{i}^{|C|} P(c_i) \cdot \log P(c_i) + P(w) \sum_{i}^{|C|} P(c_i|w) \cdot \log P(c_i|w) + P(\overline{w}) \sum_{i}^{|C|} P(c_i|\overline{w}) \cdot \log P(c_i|\overline{w}) \quad (11)$$

The feature with the highest information gain is selected, and the collection is split on that feature. This is carried out recursively until the collection at a node is homogenous or all features have an information gain of zero. This node is then assigned the category of the homogenous documents.

2.4 K-Nearest Neighbour

The k-Nearest Neighbour algorithm (kNN) is the most prominent example of *example-based* classifiers. Example-based classifiers do not build explicit declarative representations of categories but instead rely on computing the similarity between the document to be classified and the training documents. Example-based classifiers are also called *instanced-based* or *lazy learners* because they defer the decision on how to generalize beyond the training data until each new query instance is encountered. "Training" for such classifiers consists of simply storing the representations of the training documents together with their category labels.

The kNN algorithm is a very fundamental and simple algorithm. It is often the first choice for a classification study when there is little or no prior knowledge about the distribution of data. To classify a document *d*, kNN checks for the category with the maximum number of documents in *k* training documents most similar to *d*.

The kNN algorithm checks for the category with the maximum number of documents in *k* training documents most similar to the document it wants to classify and assigns that category to it.

The measure of similarity used in this report was the Euclidean distance.

$$d(p,q) = \sqrt{(w_{p1} - w_{q1})^2 + (w_{p2} - w_{q2})^2} + \ldots + (w_{pn} - w_{qn})^2 \quad (12)$$

Where p and q are documents and $w_{pi}$and w_{qi} represent the features.

The optimal value of k was found to be 5 after multiple experiments. Tf-Idf scheme was used for assigning weights to each feature in the feature vector. Normalization was also carried out on each document vector.

The kNN algorithm is one the best-performing classifiers today. It is quite robust, as it does not require categories to be linearly separated. It is also very scalable to large TC applications. Its only drawback is the relatively high computational cost of classification – that is, for each test document, its similarity to all of the training documents must be computed.

2.5 Support Vector Machines

The support vector machine (SVM) algorithm is very effective for text classification.The traditional SVM algorithm is a binary linear classification algorithm that linearly separates the positive instances from the negative instances in the feature space. The SVM algorithm was introduced in TC by [3].

Geometrically speaking, the SVM algorithm attempts to find an (n-1)-dimensional hyperplane which separates the two classes (where n is the number of features). The hyperplane that separates the positive and negative instances by the widest possible margin is selected as the classifying hyperplane during training. The margin is the distance from the hyperplane to the nearest point from the positive and negative sets.

SVM is also capable of performing non-linear classifications. This is done by the use of what is called the *Kernel trick*, which involves implicitly mapping the training instances into a higher-dimensional feature spaces. Linear classification can then be (presumably) performed in this higher-dimensional feature space. Non-linear classification can still be carried out even if the two classes are linearly separable.

An interesting attribute of SVM is the fact that its hyperplanes are determined by a relatively small subset of the training instances, which are called the *support vectors*. The rest of the training data have no influence on the trained classifier.

The SVM classifier has an important advantage in its theoretically justified approach to the overfitting problem, and thus performs well even in high-dimensional feature spaces.

The SVM algorithm constructs an optimal hyperplane which separates the training set into two. Since the classifier is binary, one-vs-all (OVA) classification was used; thus an SVM classifier had to be constructed for each category in this study. 21 SVM classifiers were constructed as a result.

The constructed hyperplanes are soft margin in nature. The linear kernel was used, because of its suitability for textual data and the fact that it's less computationally expensive than other kernels.

The optimal hyperplane H_0 in the feature space is represented as:

$$\bar{w} \cdot \bar{x} + b = 0 \qquad (13)$$

[1] proved the weight vector $\bar{w}$ (which determines the optimal hyperplane) as a linear combination of vectors:

$$\bar{w} = \sum_{i}^{n} \alpha_i \, y_i \qquad (14)$$

Substitution of the value of $\bar{w}$ in the equation (14)gives:

$$\sum_{i}^{n} \alpha_i \, y_i \bar{x}_i \cdot \bar{x} + b \qquad (15)$$

In order to get the values of $\bar{w}$ and b, the quadratic optimization problem in the equations below was solved using CVXOPT.

$$\min \alpha^T 1 - \frac{1}{2} \alpha^T \boldsymbol{K} \alpha \qquad (16)$$

$$s.t. \quad 0 \leq \alpha_i \leq C \qquad (17)$$

$$and \quad \alpha^T Y = 0 \qquad (18)$$

Where $\boldsymbol{K_{ij}} = y_i y_j x_i \cdot x_j$ and C is the soft-margin constant. This optimal value of this constant was found to be 0.01 after multiple experiments.

For each category, the Lagrange multipliers for all the document vectors were calculated, as well as the bias. Classification was then performed by the equation below:

$$C_{SVM} = argmax_{c \in C} \sum_{i}^{n} \alpha_{Ci}\, y_{Ci} \bar{x}_{Ci} \cdot \bar{x} + b_C \qquad (19)$$

Tf-Idf scheme was used for assigning weights to each feature in the feature vector. Normalization was also carried out on each document vector. Feature standardization was performed on the dataset as well.

While SVM is a very effective classifier, it possesses some drawbacks as well. SVM is quite complex (relative to other classification algorithms), and parameter optimization is usually time-consuming and computationally expensive.

3.0 PERFORMANCE MEASURES

The performance measures used in this study are the classic IR (Information Retrieval) measures of accuracy, recall and precision. The accuracy is the percentage of all correctly classified documents among all the test documents. The recall for a category is defined as the percentage of correctly classified documents among all documents belonging to that category. The precision is the percentage of correctly classified documents among all documents that were assigned to the category by the classifier.

In IR, a perfect precision score of 100% for a category *c* means that every document assigned to the category *c* by the classifier belongs to category *c*. This however does not mean that all documents belonging to category *c* were assigned to category *c*. Likewise, a perfect recall score of 100% for a category *c* means that all documents belonging to the category *c* were assigned to *c* by the classifier (but does not mean the absence of documents incorrectly labelled as belonging to category *c*).

These values can be estimated in terms of the contingency table provided in table 3 below for category $c_i$on a given test set.

Table 3: Contingency table for category $c_i$on a given test set.

	EXPERT YES	EXPERT NO
Classifier YES	True Positives (TP_i)	False Positives (FP_i)
Classifier NO	False Negatives (FN_i)	True Negatives (TN_i)

With respect to categoryc_i,

$$accuracy = \frac{TP_i + TN_i}{TP_i + TN_i + FP_i + FN_i} \qquad (20)$$

$$recall = \frac{TP_i}{TP_i + FN_i} \qquad (21)$$

$$precision = \frac{TP_i}{TP_i + FP_i} \qquad (22)$$

Often, there is an inverse relationship between recall and precision. Many classifiers allow trading recall for precision or vice versa by raising or lowering parameter settings or the output threshold. For such classifiers there is a convenient measure, called the *breakeven point*, which is the value of recall and precision at the point on the recall-versus-precision curve where they are equal. Alternatively, there is the F_1 measure, which combines the two measures in an ad hoc way.

$$F_1 = \frac{2 \times precision \times recall}{precision + recall} \quad (23)$$

For obtaining the values of precision and recall in this study, macroaveragingwas applied. Macroaveraging is usually the method of choice in TC, since producing classifiers that perform well on infrequent categories is the most challenging problem of TC.Precision and recall were calculated "locally" per category, and then "globally" by averaging over the results of different categories. The performance of the classifier in low-populated categories is emphasized here. Each category is given an equal weight.

$$recall_M = \frac{\sum_i^{|C|} recall_i}{|C|} \quad (24)$$

$$precision_M = \frac{\sum_i^{|C|} precision_i}{|C|} \quad (25)$$

4.0 DISCUSSION OF RESULTS

The evaluation of the effectiveness of the methodologies described in the previous chapter is carried out in this chapter. In more specific terms, the machine learning algorithms are compared using IR (Information Retrieval) performance metrics.

Each algorithm was evaluated in terms of accuracy, macro average precision, recall and F_1. The results of this experiment are shown in Tables 4, 5 and 6.

Results

1. Comparison of correctly categorized documents: As it can be seen in Table 4, the percentage of documents correctly categorized by SVM is slightly higher than kNN's, which is slightly higher than Naïve Bayes'. Decision Trees have the worst performance of the four.
2. Comparison of Accuracy: All the four algorithms have high accuracies, and this is due to the fact that the number of true negatives for each category is quite high for each algorithm. Once again, SVM has the best performance, decision trees have the worst performance, while kNN and Naïve Bayes are in between. This can be observed in Table 5.
3. Comparison of performance measures: A critical look at Table 6 shows that Decision trees' performance in the individual categories is a lot lower than that of the other three algorithms. kNN has the highest precision, while SVM has the highest recall and F_1 measure. This means that kNN is the most *exact* algorithm of the three (lowest false positives per category) and SVM is the most *complete* (lowest number of false negatives per category).

Table 4: Correctly Categorized Documents

ALGORITHM	CORRECT (%)
Naïve Bayes	91.74
kNN	91.78
Decision Trees	86.09
SVM	93.48

Table 5: Accuracy

ALGORITHM	ACCURACY
Naïve Bayes	99.21
kNN	99.22
Decision Trees	98.67
SVM	99.38

Table 6 Macro average Recall, Precision and F_1 Measure

ALGORITHM	RECALL	PRECISION	F_1 MEASURE
Naïve Bayes	84.51	81.75	83.11
Knn	83.67	88.83	86.17
Decision Trees	66.06	74.35	69.96
SVM	87.60	86.67	87.13

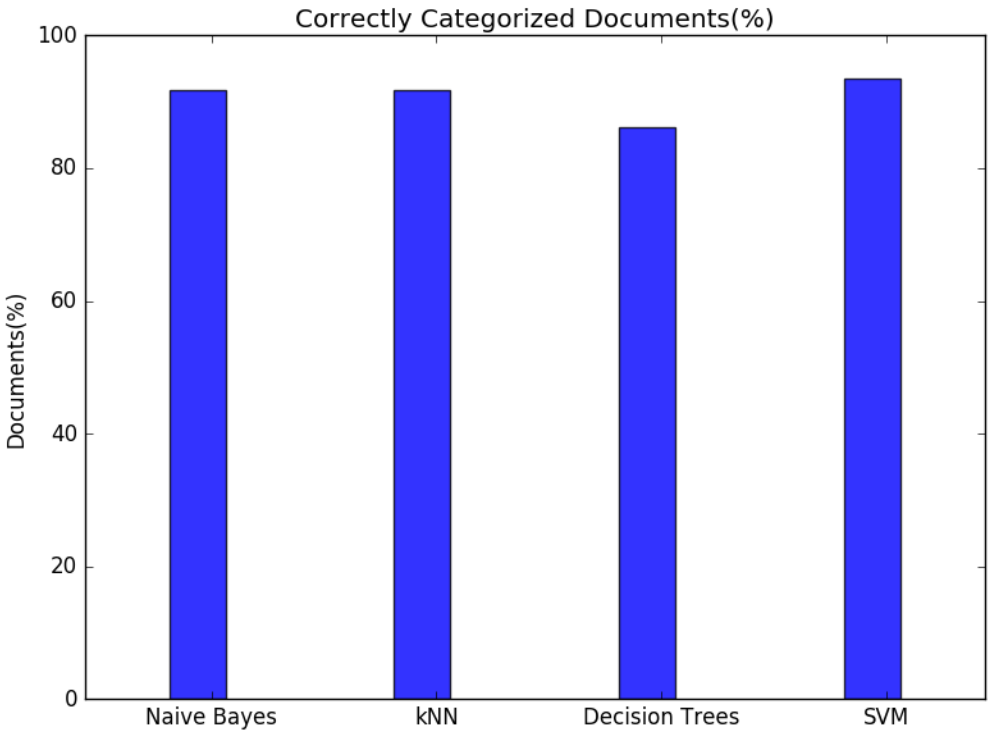

Fig .1 Correctly Categorized Documents. SVM correctly categorized more documents than the others, albeit slightly more than kNN and Naïve Bayes.

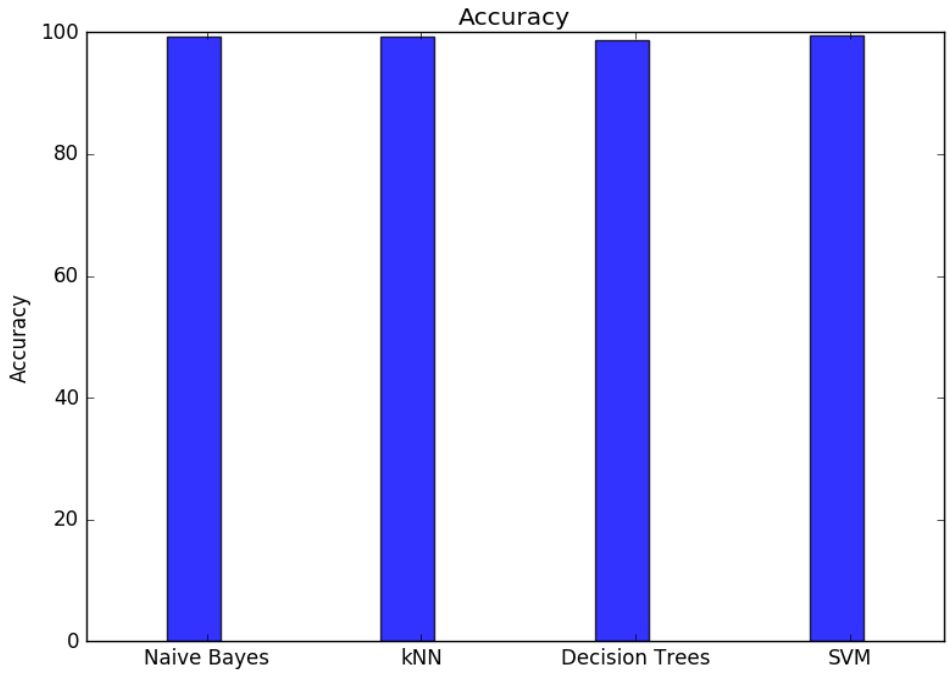

Fig 2 Accuracy of the Algorithms. All the algorithms obtained high accuracies due to the high number of true negatives in each category.

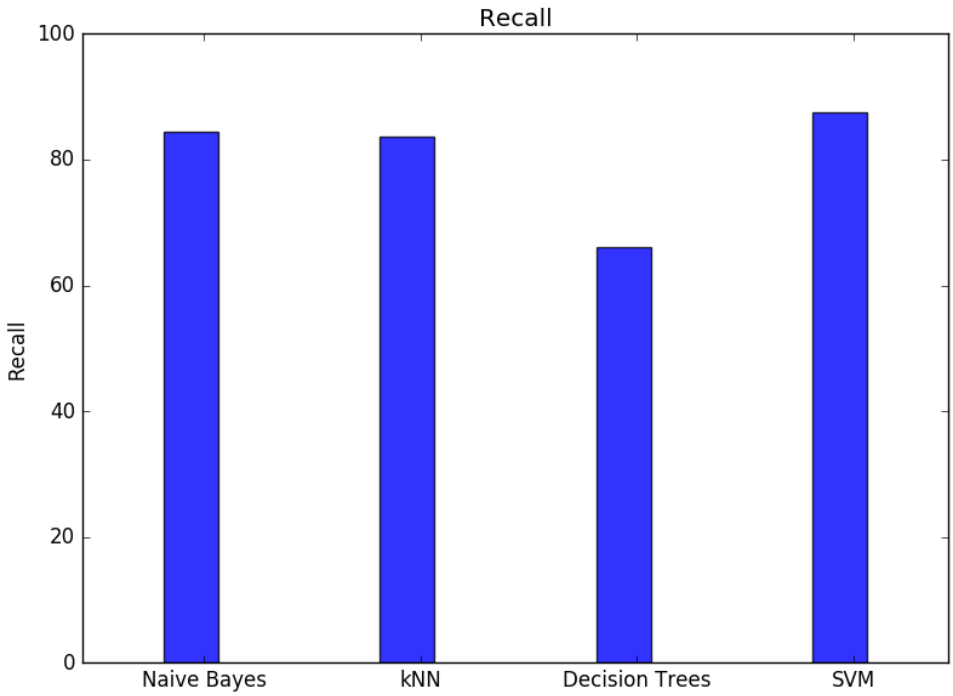

Fig. 3 Recall of the Algorithms. SVM has the highest recall. The recall of Decision Trees is significantly lower than the other algorithms.

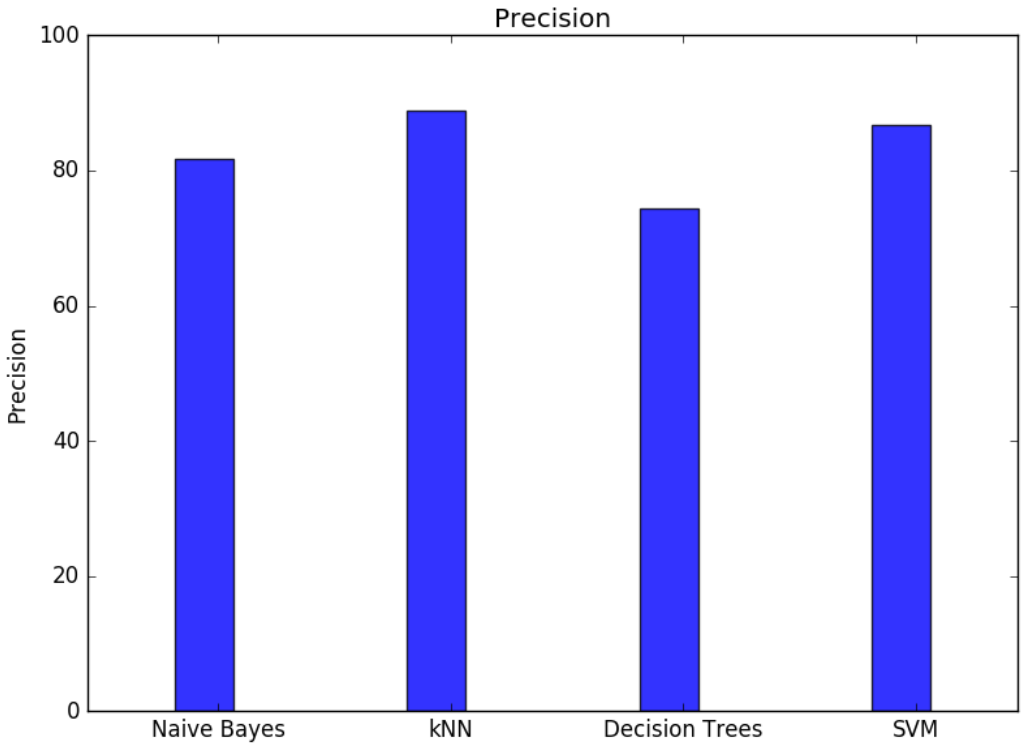

Fig. 4 Precision of the Algorithms. kNN obtained the highest precision.

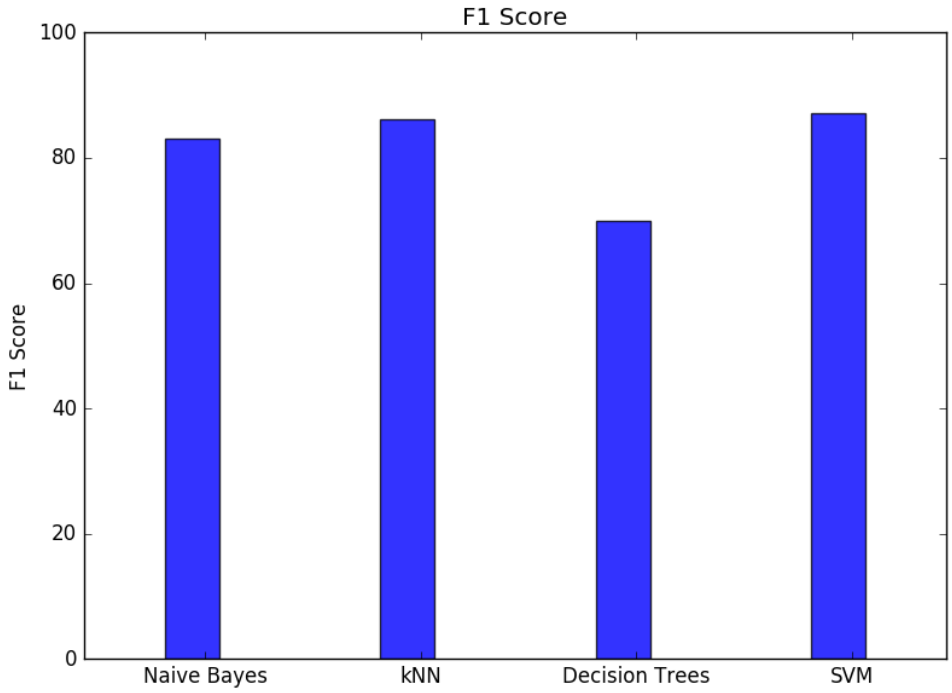

Fig. 5 F1 Score of the Algorithms. SVM has the highest F1 Score.

5.0 CONCLUSION AND FUTURE WORK

4.1 Conclusion

This study provided a thorough comparison of four machine learning text categorization algorithms namely; Naïve Bayes, k-Nearest Neighbour, Decision Trees and Support Vector Machines. Different heuristics were proposed to highlight their differences, strength and weaknesses, as well as to compare the results. The evaluation of the results showed that SVM outperformed the remaining three algorithms. The overall effectiveness of Decision Trees was significantly lower than the other algorithms, thus justifying why decision trees are majorly used as baseline classifiers today.

4.2 Future Work

There are a number of ways in which this work can be expanded. An improvement could involve the use of Latent Semantic Indexing for feature extraction instead of information gain. Another way in which this study could be improved is to compare the individual algorithms with a classifier committee consisting of these algorithms.

REFERENCES

[1]. Cortes and Vapnik (1995). *Support Vector Networks. Journal of Machine Learning, 20(1),* pp. 277-284. Publisher: Kluwer Academic Publishers, Boston.

[2]. Fabrizio Sebastiani(2002). Machine Learning in Automated Text Categorization. ACM Computing Surveys, 34(1), pp. 2-43.

[3]. Joachims, T. (1998). Text categorization with support vector machines: learning with many relevant features. *In Proceedings of ECML-98, 10th European Conference on Machine Learning (Chemnitz, DE, 1998),* pp. 137–142.

[4]. Lewis, D. D. (1992a). An evaluation of phrasal and clustered representations on a text categorization task. *In Proceedings of SIGIR-92, 15th ACM International Conference on*

Researchand Development in Information Retrieval (Kobenhavn, DK, 1992), pp. 37–50.

[5]. Ronen Feldman, and James Sanger(2007). *The Text Mining HandBook*, Cambridge University Press, pp. 64-80.

[6]. Vishal, G. and Gurpreet, S. L.(2009). *A Survey of Text Mining Techniques and Applications Journal of Emerging Technologies in Web Intelligence, (1)* pp.1.