SPAM Mail Filtering

Aayush Maini
201301012
aayush.maini@students.iiit.ac.in

Adhish Singla
201403004
adhish.singla@research.iiit.ac.in

Tarun Gupta
201403002
tarun.gupta@research.iiit.ac.in

***Abstract*—Electronic Mail is used daily by millions of people to communicate around the globe and is mission-critical application for many businesses. Mailing systems have suffered from degraded quality of service due to rampant spam, phishing and fraudulent emails. This is partly because the classification speed of email filtering systems falls far behind the requirements of email service providers. In this report, we would discuss various Algorithms implemented to Classify Spam Emails. We investigate thoroughly the performance of these filters on a publicly available corpus, contributing towards standard benchmarks. At the same time, we compare the performance of these filters with each other, after introducing suitable cost-sensitive evaluation measures. All methods achieve very accurate spam filtering, outperforming clearly the keyword-based filter of a widely used e-mail reader.**

I. Introduction

Electronic mail is an efficient and increasingly popular communication medium. Like every powerful medium, however, it is prone to misuse. One such case of misuse is the blind posting of unsolicited e-mail messages, also known as spam, to very large numbers of recipients. Spam messages are typically sent using bulk-mailers and address lists harvested from web pages and newsgroup archives. They vary significantly in content, from vacation advertisements to get-rich schemes. The common feature of these messages is that they are usually of little interest to the majority of the recipients. In some cases, they may even be harmful, e.g. spam messages advertising pornographic sites may be read by children. Apart from wasting time and bandwidth, spam e-mail also costs money to users with dial-up connections. A study reported that spam messages constituted approximately 10% of the incoming messages to a corporate network. The situation seems to be worsening, and without appropriate counter-measures, spam messages could eventually undermine the usability of e-mail.

Attempts to introduce legal measures against spam mailing have had limited effect. A more effective solution is to develop tools to help recipients identify or remove automatically spam messages. Such tools, called anti-spam filters, vary in functionality from blacklists of frequent spammers to content-based filters. The latter are generally more powerful, as spammers often use fake addresses. Existing content-based filters search for particular keyword patterns in the messages. These patterns need to be crafted by hand, and to achieve better results they need to be tuned to each user and to be constantly maintained, a tedious task, requiring expertise that a user may not have.

We address the issue of anti-spam filtering with the aid of machine learning. We examine supervised learning methods, which learn to identify spam e-mail after receiving training on messages that have been manually classified as spam or non-spam (hereafter legitimate).

The remainder of this report is organized as follows: section 2 describes our benchmark dataset; section 3 discusses preprocessing steps that are needed before applying the learning algorithms; section 4 tells how the features are extracted; section 5 presents the learning algorithms that we used; section 6 introduces cost-sensitive evaluation measures; section 7 discusses our experimental results; and section 8 concludes.

II. Datasets

For training of Classifier Models, **Lingspam** dataset is used and for testing of the Classifier, Enron and Lingspam datasets are used.

A. Lingspam

There are four sub-directories, corresponding to four versions of the corpus:

- bare - Lemmatiser disabled, stop-list disabled.
- lemm - Lemmatiser enabled, stop-list disabled.
- $lemm_stop$ - Lemmatiser enabled, stop-list enabled.
- stop - Lemmatiser disabled, stop-list enabled.

Each one of these 4 directories contains 10 subdirectories (part1, ..., part10). These correspond to the 10 partitions of the corpus that were used in the 10-fold experiments. In each repetition, one part was reserved for testing and the other 9 were used for training.

Each one of the 10 subdirectories contains both spam and legitimate messages, one message in each file. Files whose names have the form spmsg*.txt are spam messages. All other files are legitimate messages.

B. Enron

- The ”preprocessed” subdirectory contains the messages in the preprocessed format that was used in the experiments of the paper. Each message is in a separate text file. The number at the beginning of each filename is the ”order of arrival”.
- The ”raw” subdirectory contains the messages in their original form. Spam messages in non-Latin encodings, ham messages sent by the owners of the mailboxes to themselves (sender in ”To:”, ”Cc:”, or ”Bcc” field), and a handful of virus-infected messages have been removed, but no other modification has been made. The messages in the ”raw” subdirectory are more than the corresponding messages in the ”preprocessed” subdirectory, because: (a)

duplicates are preserved in the "raw" form, and (b) during the preprocessing, ham and/or spam messages were randomly subsampled to obtain the desired ham:spam ratios.

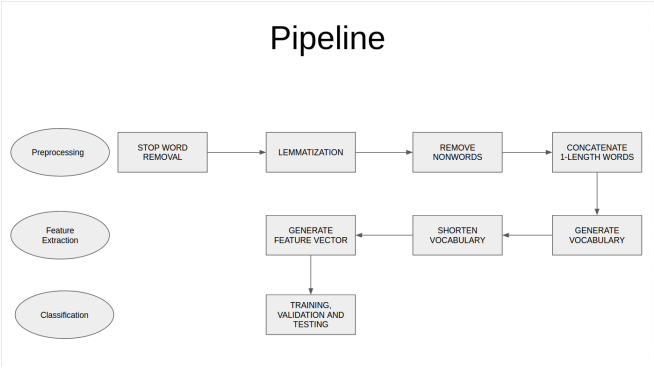

Fig. 1. Pipeline

III. Pre-Processing Details

Pre-Processing is essential for efficient Feature Extraction leading to non-redundant Feature Descriptors. The main steps involved in the Pre-Processing Stage of the Pipeline are :

- Removal of Punctuation and Special Characters
- Lemmatisation
- Stop words removal

A. Removal of Punctuation Marks

Punctuation marks and special symbols don't give any releavant information about the content. Thus, they can't be used to estimate the degree of similarity between the contents of two different mails. Thus, they are removed to remove this irrelevant information.

B. Lemmatisation and Stemming

Lemmatisation is the process of grouping together the different inflected forms of a word so they can be analysed as a single item. For example, in English, the verb 'to walk' may appear as 'walk', 'walked', 'walks', 'walking'. The base form, 'walk', that all of them are reduced to , is called the lemma for the word.

Lemmatisation is closely related to stemming. The difference is that a stemmer operates on a single word without knowledge of the context, and therefore cannot discriminate between words which have different meanings depending on part of speech. However, stemmers are typically easier to implement and run faster, and the reduced accuracy may not matter for some applications. For example, The word "better" has "good" as its lemma. This link is missed by stemming, as it requires a dictionary look-up

C. Stop Words Removal

Stop words refer to the most common words in a language. Although, there is no universal list for the Stop Words, but an exhaustive list has been used to ensure better accuracy and efficient feature extraction.

They need to be removed because they are most likely to be present in both spam and ham emails and hence, both kind of emails will share these tokens when they are converted into their equivalent feature descriptors. Thus, they are removed as a part of pre-processing giving comparably lower dimensional feature descriptors and hence, improved performance in terms of time.

IV. Feature Extraction

After the pre-processing stage, sufficient amount of redundancy gets removed from the message content. The mails now need to be converted to equivalent feature descriptors that have sufficient number of dimensions to ensure good classification and low enough to ensure that computation performed on them is tractable.

Basic Procedure for Feature Extraction :

- **Vocabulary Generation** : Obtain the vocabulary for the training data, by tokenizing the message content and finding union of all distinct words over all emails.
- **Getting Numerical Descriptor** : Tokenizing the email gives us a feature vector with string attributes. For the purpose of finding similarity between emails, we need numerical descriptors.

$W = \{$ Set of distinct words in the message $\}$
N_{w_i} = No. of times word w_i appears in the email.
F (feature vector) $= \{N_{w_i} | w_i \epsilon W\}$

V. Algorithms

A. Naive Bayes

In machine learning, naive Bayes classifiers are a family of simple probabilistic classifiers based on applying Bayes' theorem with strong (naive) independence assumptions between the features.

A naive Bayes classifier considers each of these features to contribute independently to the probability that this fruit is an apple, regardless of any possible correlations between the color, roundness and diameter features.

Abstractly, naive Bayes is a conditional probability model: given a problem instance to be classified, represented by a vector $\mathbf{x} = (x_1, \ldots, x_n)$ representing some n features (independent variables), it assigns to this instance probabilities

$$p(C_K | x_1, \ldots, x_n) \tag{1}$$

for each of K possible outcomes or classes.

The problem with the above formulation is that if the number of features n is large or if a feature can take on a large number of values, then basing such a model on probability tables is infeasible. We therefore reformulate the model to make it more tractable. Using Bayes' theorem, the conditional probability can be decomposed as

$$p(C_K | \mathbf{x}) = \frac{p(C_K) p(\mathbf{x} | C_K)}{p(\mathbf{x})} \tag{2}$$

$$p(C_K | x_1, \ldots, x_n) = \frac{1}{Z} p(C_K) \prod_{i=1}^{n} p(x_i | C_K) \tag{3}$$

where the evidence $Z = p(\mathbf{x})$ is a scaling factor dependent only on $x_1, \ldots, x_n$, that is, a constant if the values of the feature variables are known.

The discussion so far has derived the independent feature model, that is, the naive Bayes probability model. The naive Bayes classifier combines this model with a decision rule. One common rule is to pick the hypothesis that is most probable; this is known as the maximum a posteriori or MAP decision rule. The corresponding classifier, a Bayes classifier, is the function that assigns a class label $\hat{y} = C_k$ for some k as follows:

$$\hat{y} = \underset{k \in \{1,\ldots,K\}}{\operatorname{argmax}} \; p(C_k) \prod_{i=1}^{n} p(x_i|C_k). \tag{4}$$

B. K-Nearest Neighbour

In pattern recognition, the k-Nearest Neighbors algorithm (or k-NN for short) is a non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space.

The training examples are vectors in a multidimensional feature space, each with a class label. The training phase of the algorithm consists only of storing the feature vectors and class labels of the training samples.

In the classification phase, k is a user-defined constant, and an unlabeled vector (a query or test point) is classified by assigning the label which is most frequent among the k training samples nearest to that query point.

A commonly used distance metric for continuous variables is Euclidean distance. For discrete variables, such as for text classification, another metric can be used, such as the overlap metric (or Hamming distance). In the context of gene expression microarray data, for example, k-NN has also been employed with correlation coefficients such as Pearson and Spearman. Often, the classification accuracy of k-NN can be improved significantly if the distance metric is learned with specialized algorithms such as Large Margin Nearest Neighbor or Neighbourhood components analysis.

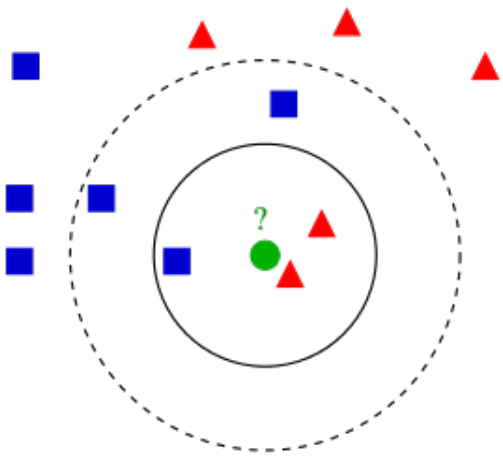

Fig. 2. Example of k-NN classification.

A drawback of the basic ”majority voting” classification occurs when the class distribution is skewed. That is, examples of a more frequent class tend to dominate the prediction of the new example, because they tend to be common among the k nearest neighbors due to their large number. One way to overcome this problem is to weigh the classification, taking into account the distance from the test point to each of its k nearest neighbors. The class (or value, in regression problems) of each of the k nearest points is multiplied by a weight proportional to the inverse of the distance from that point to the test point. Another way to overcome skew is by abstraction in data representation.

C. Artificial Neural Network

In machine learning and cognitive science, artificial neural networks (ANNs) are a family of models inspired by biological neural networks (the central nervous systems of animals, in particular the brain) and are used to estimate or approximate functions that can depend on a large number of inputs and are generally unknown. Artificial neural networks are generally presented as systems of interconnected ”neurons” which exchange messages between each other. The connections have numeric weights that can be tuned based on experience, making neural nets adaptive to inputs and capable of learning.

We are given a set of example pairs $(x, y), x \in X, y \in Y$ and the aim is to find a function $f : X \to Y$ in the allowed class of functions that matches the examples. In other words, we wish to infer the mapping implied by the data; the cost function is related to the mismatch between our mapping and the data and it implicitly contains prior knowledge about the problem domain.

A commonly used cost is the mean-squared error, which tries to minimize the average squared error between the network’s output, $f(x)$, and the target value y over all the example pairs. When one tries to minimize this cost using gradient descent for the class of neural networks called multilayer perceptrons, one obtains the common and well-known backpropagation algorithm for training neural networks.

Tasks that fall within the paradigm of supervised learning are pattern recognition (also known as classification) and regression (also known as function approximation). The supervised learning paradigm is also applicable to sequential data (e.g., for speech and gesture recognition). This can be thought of as learning with a ”teacher”, in the form of a function that provides continuous feedback on the quality of solutions obtained thus far.

D. SVM

In machine learning, support vector machines (SVMs, also support vector networks) are supervised learning models with associated learning algorithms that analyze data and recognize patterns, used for classification and regression analysis. Given a set of training examples, each marked for belonging to one of two categories, an SVM training algorithm builds a model that assigns new examples into one category or the other, making it a non-probabilistic binary linear classifier. An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided

by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall on.

Writing the classification rule in its unconstrained dual form reveals that the maximum-margin hyperplane and therefore the classification task is only a function of the support vectors, the subset of the training data that lie on the margin.

Maximize (in α_i)

$$\tilde{L}(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j) \tag{5}$$

E. Decision Trees

A decision tree is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm.

Decision trees are commonly used in operations research, specifically in decision analysis, to help identify a strategy most likely to reach a goal.

Decision tree learning uses a decision tree as a predictive model which maps observations about an item to conclusions about the item's target value. It is one of the predictive modelling approaches used in statistics, data mining and machine learning. Tree models where the target variable can take a finite set of values are called classification trees. In these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. We have used the following Decision Tree Algorithms for Classification:

1) CART: Classification and regression trees (CART) are a non-parametric decision tree learning technique that produces either classification or regression trees, depending on whether the dependent variable is categorical or numeric, respectively.

Decision trees are formed by a collection of rules based on variables in the modeling data set:

- Rules based on variables' values are selected to get the best split to differentiate observations based on the dependent variable.
- Once a rule is selected and splits a node into two, the same process is applied to each "child" node (i.e. it is a recursive procedure).
- Splitting stops when CART detects no further gain can be made, or some pre-set stopping rules are met. (Alternatively, the data are split as much as possible and then the tree is later pruned.).

Each branch of the tree ends in a terminal node. Each observation falls into one and exactly one terminal node, and each terminal node is uniquely defined by a set of rules.

2) Random Forest: A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is always the same as the original input sample size but the samples are drawn with replacement.

F. Rough Sets

Rough set (RS) theory was developed by Pawlak. Rough set has a great ability to compute the reductions of information systems. Information system might has some attributes that are irrelevant to the target concept (i.e. decision attribute), and some redundant attributes. Reduction is needed to generate simple useful knowledge from it. It is a minimal subset of condition attributes with respect to decision attributes. The Rough set scheme is provided as follows:

Step-1: With the the incoming emails, first thing we need to do is to select the most appropriate attributes to use for classification. Then the input dataset is transformed into a decision system , which is then split into the training dataset and the testing dataset. A classifier will be induced from the training dataset and applied to the testing dataset to obtain performance estimation. For training dataset, do Step 2 and Step 3.

Step-2: Because the decision system has real values attributes, Boolean reasoning algorithm should be used to finish the discretization strategies. **Step-3:** Genetic algorithms should be used to get the decision rules. Then For testing dataset, continue to Step 4. **Step-4:** First, discretizes the testing dataset employing the same cuts computed from step 2. Then the rules generated in Step 3 are used to match every new object in testing dataset to make decision. Let $b = 0.15 \in (0, 21)$ be the threshold for positive region, therefore, these b - lower and b - upper approximations divide the whole emails in tree regions, called 0.15-positive, 0.15-boundary and 0.15-negative region

G. AdaBoosting

An AdaBoost classifier is a meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases. It, thus is an AdaBoost, Adaptive Boosting algorithm.

AdaBoost is adaptive in the sense that subsequent weak learners are tweaked in favor of those instances misclassified by previous classifiers. AdaBoost is sensitive to noisy data and outliers. In some problems, however, it can be less susceptible to the overfitting problem than other learning algorithms. The individual learners can be weak, but as long as the performance of each one is slightly better than random guessing (i.e., their error rate is smaller than 0.5 for binary classification), the final model can be proven to converge to a strong learner.

We have used AdaBoost with decision trees as the weak learners. When used with decision tree learning, information gathered at each stage of the AdaBoost algorithm about the relative 'hardness' of each training sample is fed into the tree growing algorithm such that later trees tend to focus on harder-to-classify examples.

1) Training: AdaBoost adopts a particular method of training a boosted classifier. A boost classifier is a classifier in the form

$$F_T(x) = \sum_{t=1}^{T} f_t(x)$$

where each f_t is a weak learner that takes an object x as input and returns a real valued result indicating the class of the object. The sign of the weak learner output identifies the predicted object class and the absolute value gives the confidence in that classification. Similarly, the T-layer classifier will be positive if the sample is believed to be in the positive class and negative otherwise.

Each weak learner produces an output, hypothesis $h(x_i)$, for each sample in the training set. At each iteration t, a weak learner is selected and assigned a coefficient α_t such that the sum training error E_t of the resulting t-stage boost classifier is minimized.

$$E_t = \sum_i E[F_{t-1}(x_i) + \alpha_t h(x_i)]$$

Here $F_{t-1}(x)$ is the boosted classifier that has been built up to the previous stage of training, E(F) is some error function and $f_t(x) = \alpha_t$ h(x) is the weak learner that is being considered for addition to the final classifier.

2) Weighting: At each iteration of the training process, a weight is assigned to each sample in the training set equal to the current error

$$E(F_{t-1}(x_i))$$

on that sample. These weights can be used to inform the training of the weak learner, for instance, decision trees can be grown that favor splitting sets of samples with high weights.

H. Online Classification

The Classification of emails can be improved further, if the model of classification available can be trained again using the test data at fly.

There are 2 possible cases :

- Prior Information about labels of Test Data available In this case, after classification, since label of test input is available, we can simply train the new training data which now includes the test input along with its label(priori).

$$T_{old} = OldTrainingData$$
$$Model_{old} = Train(T_{old})$$
$$t = TestInput$$
$$T_{new} = T_{old} \cup t$$
$$Model_{new} = Train(T_{new})$$

- No Prior Information about label of test data available In this case, we have no prior information, however, we can still guess close to ground truth by relying on natural structuring of data. That is, we cluster the data, assign clusters the labels depending on the clusters that had existed earlier.

$$T_{old} = OldTrainingData$$
$$Model_{old} = Train(T_{old})$$
$$t = TestInput$$
$$T_{new} = T_{old} \bigcup t$$
$$C_1, ..., C_k = Cluster(T_{new})$$

Practical Aspects :

- **Computational Overload** : Clustering is a computationally expensive procedure, and it depends on number of samples being clustered. Therefore, we should try to minimize the number of samples being clustered.
  This can be overcome using the Fuzzy K-Means Clustering. For each class, we can find out the points that characterize it to a greater extent than the others. These points are those, for which value of membership corresponding to that class is high (higher than a threshold T). For each of the clusters, we find out these characteristic points and then, try to cluster the test input using them. This can significantly reduce computational expense.

  D_i = Set of Characteristic Points for Cluster C_i

  $mem(x, i)$ = Membership of point x in Cluster C_i

  D_i = Set of Characteristic Points for Cluster C_i

  $D_i = \{x | mem(x, i) > Threshold\}$

  $C_1, ..., Ck = Cluster(\bigcup_i^{D_i} \bigcup t)$

- **When to Cluster again** : Training/Clustering again for each test input can prove to very expensive because either of the procedures are computationally expensive. Therefore, the training/clustering step should be repeated after receiving a fixed amount of test inputs, without affecting accuracy.

VI. Evaluation Measures

In order to test the performance of above mentioned methods, we used the most popular evaluation methods used by the spam filtering researchers.

A. Spam Precision

Spam Precision (SP) is the number of relevant documents identified as a percentage of all documents identified; this shows the noise that filter presents to the user (i.e. how many of the messages classified as spam will actually be spam).

$$SP = \frac{NumberofSpamCorrectlyClassified}{TotalNumberofmessagesclassifiesasspam} \quad (6)$$

$$SP = \frac{N_{spam \to spam}}{N_{spam \to spam} + N_{ham \to spam}} \quad (7)$$

B. Spam Recall

Spam Recall (SR) is the percentage of all spam emails that are correctly classified as spam.

$$SR = \frac{NumberofSpamCorrectlyClassified}{TotalNumberofmessages} \tag{8}$$

$$SR = \frac{N_{spam \to spam}}{N_{spam \to spam} + N_{spam \to ham}} \tag{9}$$

C. Accuracy

Accuracy (A) is the percentage of all emails that are correctly categorized.

$$A = \frac{Numberofmailscorrectlycategorized}{TotalNumberofEmails} \tag{10}$$

$$A = \frac{N_{spam \to spam} + N_{ham \to ham}}{N_{spam} + N_{ham}} \tag{11}$$

D. F1-Score

The F1 score (also F-score or F-measure) is a measure of a testś accuracy. It considers both the precision p and the recall r of the test to compute the score: p is the number of correct positive results divided by the number of all positive results, and r is the number of correct positive results divided by the number of positive results that should have been returned. The 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 at 0.

$$F1 = 2 * \frac{precision.recall}{precision + recall} \tag{12}$$

VII. Results

A. Naive Bayes

Parametrized By : Model to represent Data
Accuracy: 99.89%

	precision	recall	f1-score	support
NonSpam	1.00	1.00	1.00	4824
Spam	0.99	1.00	1.00	962
Avg/total	1.00	1.00	1.00	5786

B. K Nearest Neighbour

Parametrized By : K (No. of Neighbors), Distance Metric
Accuracy: 99.4%

	precision	recall	f1-score	support
NonSpam	1.00	1.00	1.00	4824
Spam	0.99	1.00	1.00	962
Avg/total	1.00	1.00	1.00	5786

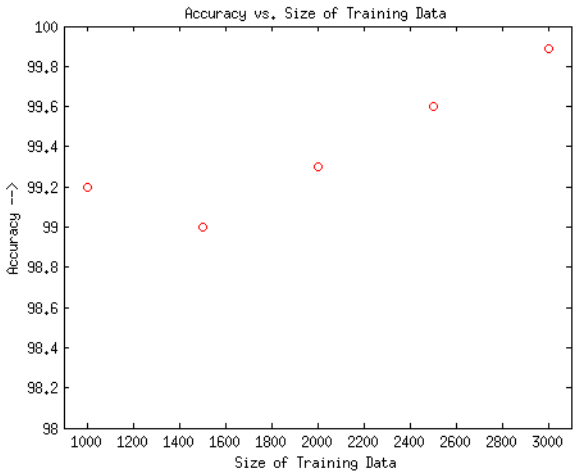

Fig. 3. Naive Bayes classification.

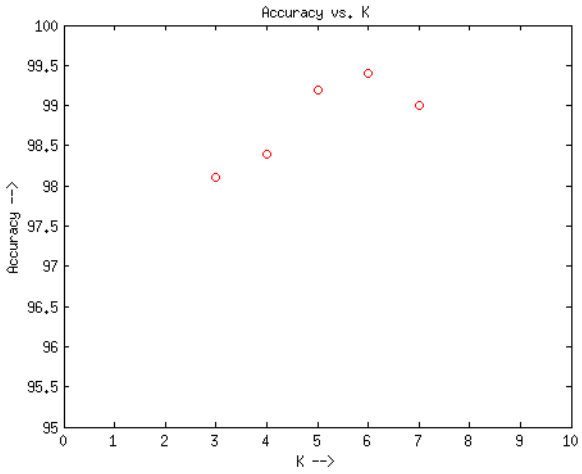

Fig. 4. KNN Mahalanobis classification.

C. Artificial Neural Network

Parametrized By : No. of Hidden Units
Accuracy: 94.12%

	precision	recall	f1-score	support
NonSpam	1.00	1.00	1.00	4824
Spam	0.99	1.00	1.00	962
Avg/total	1.00	1.00	1.00	5786

D. Support Vector Machine

Parametrized By : Choice of Kernel

1) RBF Kernel:

$$K(i,j) = e^{\frac{||y_i - y_j||^2}{2\sigma^2}}$$

Accuracy: 87.97%

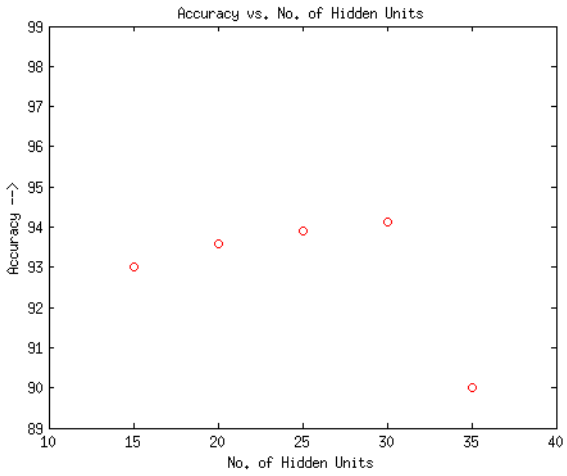

Fig. 5. Artificial Neural Network classification.

	precision	recall	f1-score	support
NonSpam	0.87	1.00	0.93	4824
Spam	1.00	0.28	0.43	962
Avg/total	0.89	0.88	0.85	5786

Number of support vectors for spam are 430 and for non-spam were 483.

2) Polynomial Kernel:

$$K(i,j) = (1 + y_i^T yj)^n$$

Accuracy: 83.40%

	precision	recall	f1-score	support
NonSpam	0.83	1.00	0.91	4824
Spam	1.00	0.00	0.00	962
Avg/total	0.86	0.83	0.76	5786

Number of support vectors for spam are 430 and for non-spam were 483.

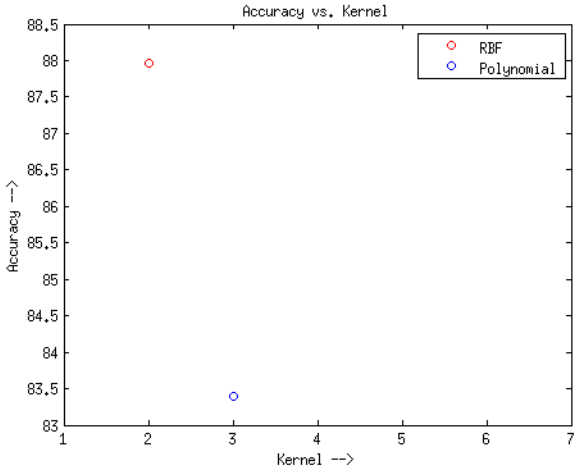

Fig. 6. Support Vector Machine classification.

E. Decision Trees

1) CART: Parametrized By : Structure of Tree
Accuracy: 100%

	precision	recall	f1-score	support
NonSpam	1.00	1.00	1.00	4824
Spam	1.00	1.00	1.00	962
Avg/total	1.00	1.00	1.00	5786

2) Random Forests: Parametrized By : Structure of Tree
Accuracy: 99.82%

	precision	recall	f1-score	support
NonSpam	1.00	1.00	1.00	4824
Spam	1.00	0.99	0.99	962
Avg/total	1.00	1.00	1.00	5786

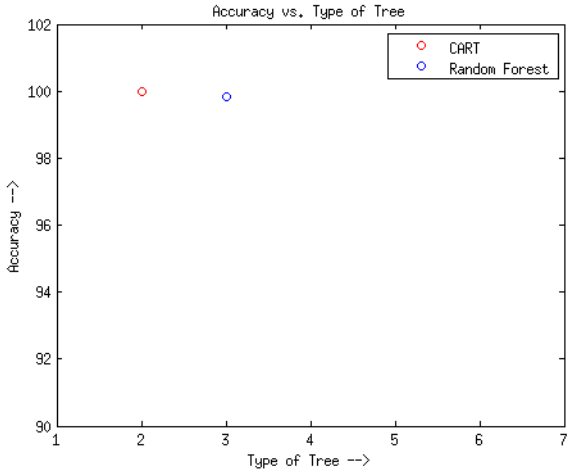

Fig. 7. Decision Tree classification.

F. Rough Sets Classification

Accuracy: 95.12%

	precision	recall	f1-score	support
NonSpam	1.00	1.00	1.00	4824
Spam	0.98	1.00	1.00	962
Avg/total	1.00	1.00	1.00	5786

G. Adaboosting

Parametrized By : Choice of Weak Classifier, No. of Weak Classifiers
Accuracy: 100%

	precision	recall	f1-score	support
NonSpam	1.00	1.00	1.00	4824
Spam	1.00	1.00	1.00	962
Avg/total	1.00	1.00	1.00	5786

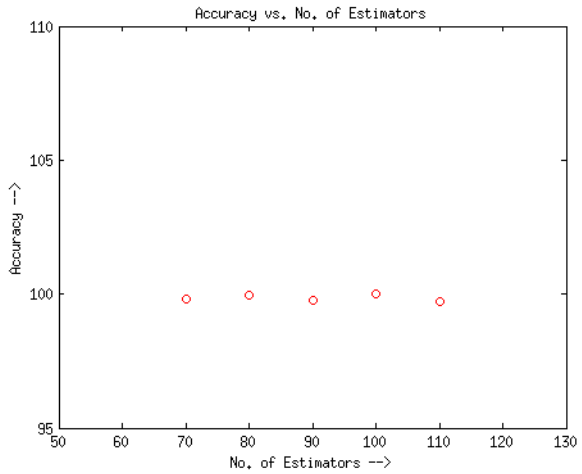

Fig. 8. Adaboost classification.

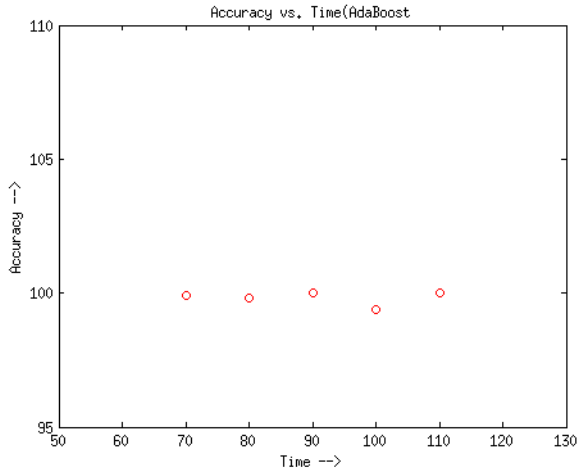

Fig. 9. Online Ada classification.

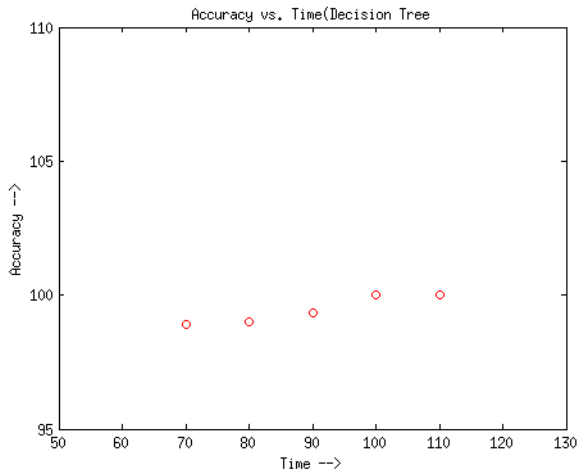

Fig. 10. Online Decision Tree classification.

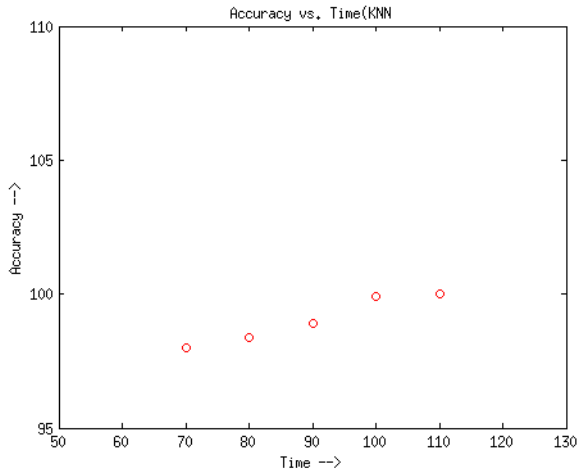

Fig. 11. Online KNN classification.

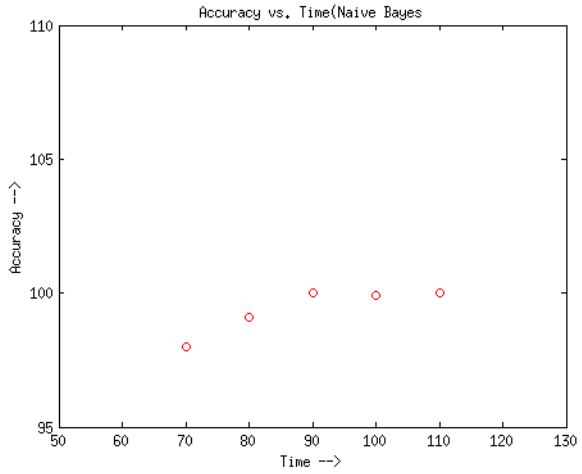

Fig. 12. Online Naive Bayes classification.

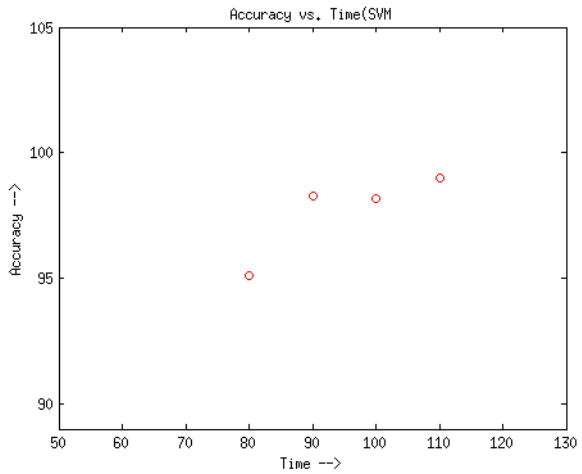

Fig. 13. Online SVM classification.

REFERENCES

[1] Awad, W. A., and S. M. ELseuofi. "Machine Learning methods for E-mail Classification." International Journal of Computer Applications (09758887) 16.1 (2011).

[2] http://scikit-learn.org/stable/

[3] Porter Stemmer. http://tartarus.org/martin/PorterStemmer/matlab.txt