Classification of SMS as Spam or Ham

Pooja Sastry, Hera Siddiqui, Chitra Barla
San Diego State University
psastry@sdsu.edu, hsiddiqui@sdsu.edu, cbarla@sdsu.edu

***Abstract*– In the current era of mobile devices, Short Message Service (SMS) is one of the most widely used text communication platforms. However, there has been an increase in attacks like SMS Spam. In the current paper, we tackle the issue of classifying an SMS as Spam or Ham (legitimate). For this purpose, a database of real SMS Spams originally from UCI Machine Learning repository, hosted on Kaggle repository has been used. After data preprocessing, machine learning techniques like Multinomial NB, Logistic Regression, Random Forests, Decision Tree, Support Vector Machine, Ada Boost, k-NN Classifier and Neural Networks have been applied to the database. Final simulation results are done using ten-fold cross validations. Finally, the results are compared based on accuracy scores, confusion matrices and the best algorithm for spam filtering for text messaging is discussed.**

I. INTRODUCTION

With the utilization of mobile phone devices becoming commonplace, Short Message Service (SMS) has grown into a multi-billion dollars commercial industry. With increasing popularity of this communication platform, there has been a surge in the number of unsolicited commercial advertisements sent to mobile phones using text messaging. SMS Spam is more of a problem than email spam, since in some countries they contribute to a cost for the receiver as well. These factors along with limited availability of mobile phone spam-filtering software makes spam detection for text messages an interesting problem to consider.

Unlike for emails, real databases for SMS spams are very limited. Additionally, due to the small length of text messages, the number of features that can be used for their classification is far smaller. All these factors may result in serious degradation in performance of major email spam filtering algorithms applied to short text messages. In this project, the goal is to apply different machine learning algorithms like Multinomial Naïve Bayes, Logistic Regression, Random Forest, Decision Tree, SVM, Neural Network, k-NN and Ada Boost Classifiers to our dataset with the help of the Scikit learn python library [1] to SMS Spam classification problem and figure out the most efficient algorithm in classifying Spam from Ham (Legitimate) SMS.

The project report is organized as follows: Section 2 explains about the dataset used while data preprocessing and extraction of features from the main dataset is covered in Section 3. Section 4 deals with different Machine Learning models used, whose evaluation is mentioned in Section 5. Finally, Section 6 concludes the report.

II. DATASET INFORMATION

We have made use of SMS Spam collection dataset from the UCI Machine Learning repository which has been hosted on the Kaggle repository [2][3]. The problem we address is a binary classification problem to predict if a message is spam or ham based on few factors.

The SMS Spam Collection is a set of tagged messages that have been collected for SMS Spam research. The file contains a set of SMS messages in English, tagged according being ham (legitimate) or spam (illegitimate). It contains one message data per line. Each line is composed of two columns: v1 contains the label (ham or spam) and v2 contains the raw text. There are 5,574 such entries.

The main feature utilized in our model is the text message i.e. the v2 column. Each row consists of a set of words. These set of raw words must be analyzed to classify them as ham or spam. Therefore, our dataset mainly consists of categorical data. We have applied data preprocessing techniques to extract the most relevant words from the set of raw words to use in the classification models.

III. DATA PREPROCESSING

Relevant features must be selected before dealing with data splitting and its transformation. As part of data pre-processing, firstly, the unwanted rows of our dataset which had irrelevant values were dropped. Next, the remaining columns were renamed. Fig. 1. depicts the plot showing the count of ham and spam SMS before the application of any Machine Learning models.

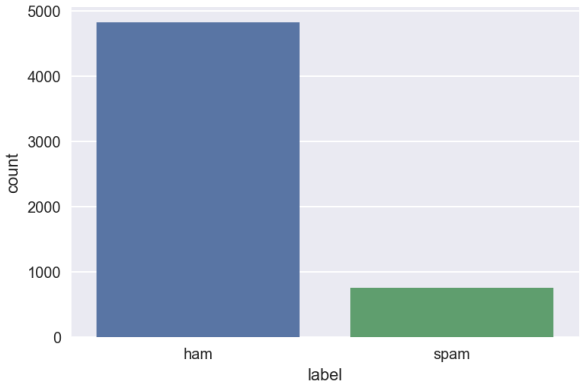

Fig. 1. Initial Ham and Spam Message Counts

A. K-Fold Cross Validation

Cross Validation is a technique which involves reserving a part of the dataset, called validation set, on which the model is not trained. After training the model, it is tested using the validation set before finalizing the model. To achieve this, an increase in size of training data could negatively affect our testing process, and vice versa.

With data being scarce, we address this issue by using a method called K-Fold Cross Validation. K-Fold divides the data samples in *k* groups of samples, called *folds*, of equal sizes. The prediction function is learned using *k-1* folds, and the fold left out is used for test. This is repeated until each of the k-folds has served as the test set. The average of the k recorded errors is called the cross-validation error and will serve as the performance metric for the model [4].

A lower value of k is more biased and hence undesirable whereas higher value of k is less biased, but can suffer from large variability. In this project, we have chosen k-value as 10.

B. Text Transformation

Text data requires special preparation before using it for predictive modeling. The Count Vectorizer provides a simple way to tokenize a collection of text documents and build a vocabulary of known words. It also helps to encode new documents using that vocabulary.

The *CountVectorizer()* function converts a collection of text documents to a matrix of token counts. The text must be parsed to remove words, called tokenization. Then the words need to be encoded as integers or floating-point values for use as input to a machine learning algorithm, called feature extraction (or vectorization).

In python, after creating an instance of the Count Vectorizer class, *fit()* function is called to learn a vocabulary from one or more documents and the *transform()* function is needed to encode each as a vector.

An encoded vector is returned with a length of the entire vocabulary and an integer count for the number of times each word appeared in the document. Because these vectors will contain a lot of zeros, we call them sparse. Python provides an efficient way of handling sparse vectors in the scipy.sparse package.

IV. MACHINE LEARNING MODELS

A. Multinomial Bayes Classifier

Naive Bayes classifiers are a class of simple linear classifiers which are conditional probability models based on Bayes Theorem. A naive Bayes classifier considers each of these features to contribute independently. In this section, NB algorithm is applied to the final extracted features. The speed and simplicity along with high accuracy of this algorithm makes it a desirable classifier for spam detection problems. With a multinomial event model, the feature vectors represent the frequencies with which certain events have been generated by a multinomial, $p_i = (p_1, \ldots, p_n)$ where p_i is the probability that event *i* occurs.

A feature vector $x = (x_i,.,x_n)$ is then a histogram, with x_i counting the number of times event *i* was observed in a particular instance. This is the event model used for spam classification, with events representing the occurrence of a word in a single message.

The most important parameters for the *MultinomialNB()* function are, *alpha* - additive smoothing parameter, *class_prior* - prior probabilities of the classes and *fit_prior* - whether to learn class prior probabilities or not.

B. Random Forest

Random forests are an ensemble learning method for classification, that operate by constructing a multitude of decision trees at training time and outputting the class. A random forest is a meta estimator that fits many decision tree classifiers on various sub-samples of the dataset and uses 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 if *bootstrap* is set as True.

Each tree is generated through a subset of the initial training set randomly. Each tree is also generated using just a few features. The splitting criteria on each tree node is to use a condition over typically one feature which divides the dataset in two equal parts. The criteria to select which feature to consider in each node is, by default, the *gini* criterion. This process stops whenever all the remaining data samples have the same label. The leaf node will assign labels to test samples.

Features which make predictions of the model using the function *RandomForestClassifier()* better are *Max_features* - the maximum features Random Forest can try in individual tree, *n_estimators* - the number of trees in the forest and *min_samples_leaf* - a smaller leaf makes the model more prone to capturing noise in training data.

C. Logistic Regression

Logistic regression is a generalized linear model for classification with a probabilistic interpretation that can be used to predict discrete values. The logistic function, also called the sigmoid function was developed to describe properties of population growth in ecology, rising quickly and maxing out at the carrying capacity of the environment. Below is an example logistic regression equation:

$$y = 1/ (1 + e^{(b0+b1*x)})$$

where input values *x* is combined linearly using weights or coefficient values to predict an output value *y*, *b0* is the bias or intercept term and *b1* is the coefficient for the single input value *x*. Each column in the input data has an associated *b* coefficient (a constant real value) that must be learned from the training data. Logistic regression can be seen as a special case of the generalized linear model and thus analogous to linear regression.

Logistic regression measures the relationship between the categorical dependent variable and one or more independent variables by estimating probabilities using a logistic function, which is the cumulative logistic distribution.

For the function *LogisticRegression()*, the parameter *solver* has been set to 'liblinear'. Parameter *tol* gives the tolerance for the stopping criteria and is set to default value of '1e-4'.

D. Support Vector Machine

Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression, and outliers' detection. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to 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. The SVM algorithm uses a flexible representation of the class boundaries and implements automatic complexity control to reduce overfitting. It is popular since it often has good generalization performance. Also, the same algorithm can solve a variety of problems with little tuning.

In this project, we are using Scikit-learn package for implementing SVM algorithm [5]. The support vector machines in Scikit-learn support both dense (numpy.ndarray and convertible to that by numpy.asarray) and sparse (any scipy.sparse) sample vectors as input. The SVM algorithm is implemented in practice using a kernel. A kernel is a similarity function which is provided to the machine learning algorithm. It takes two inputs and tells how similar they are. Kernels are easier to compute than the feature vector for the corresponding kernel. We can work with highly complex, efficient-to-compute, and yet high performing kernels without ever having to write down the huge dimensional feature vector. This is called as the kernel trick.

In this project, the below-mentioned three different kernel types were used and the one giving the best accuracy score was used for final comparisons:

1) Linear Kernel SVM

This is often recommended for text classification problems since they are linearly separable. The dot-product is called the kernel and can be written as:

$$K(x, xi) = sum(x * xi)$$

The kernel defines the similarity or a distance measure between new data and the support vectors. The dot product is the similarity measure used for linear SVM or a linear kernel because the distance is a linear combination of the inputs.

2) Polynomial Kernel SVM

The polynomial kernel looks not only at the given features of input samples to determine their similarity, but also combinations of these. That is, if we want to be able to model occurrences of pairs of words, which give distinctive information about topic classification, not given by the individual words alone.

Instead of the dot-product, we can use a polynomial kernel, for example:

$$K(x,xi) = 1 + sum(x * xi)^d$$

where the degree of the polynomial must be specified by hand to the learning algorithm. When $d = 1$ this is the same as the linear kernel. The polynomial kernel allows for curved lines in the input space. This is quite popular in natural language processing.

3) Radial Kernel SVM

A radial basis function (RBF) is equivalent to mapping the data into an infinite dimensional Hilbert space. The most common form of radial basis function is a Gaussian distribution, calculated as:

$$K(x,xi) = exp(-gamma * sum((x - xi^2))$$

where gamma is a parameter that must be specified to the learning algorithm. A good default value for *gamma* is 0.1, where gamma is often $0 < gamma < 1$. The radial kernel is very local and can create complex regions within the feature space, like closed polygons in two-dimensional space.

We calculated the accuracy using the above three kernels using a constant $C = 1.25$, and found that SVM Linear gives the best accuracy score. Fig. 2 gives the plot of accuracy score against different SVM kernels. Linear kernel is very well suited for text-categorization which in turn helps in Spam SMS classification.

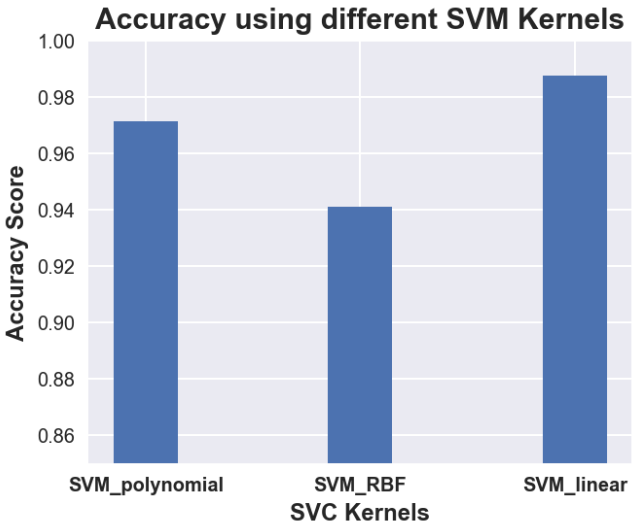

Fig. 2. Accuracy Scores using different Kernel Types

E. K-Nearest Neighbors

K-nearest neighbor can be applied to the classification problems as a simple instance-based learning algorithm. In this method, the label for a test sample is predicted based on the majority vote of its k nearest neighbors [6]. In K-NN classification, the output is a class membership. An object is assigned to the class which is most common among its *k* nearest neighbors.

Weights are assigned to the contributions of the neighbors so that the nearer neighbors contribute more to the average than the more distant ones. A common weighting scheme consists in giving each neighbor a weight of 1/d, where d is the distance to the neighbor. The training phase of the algorithm consists only of storing the feature vectors and class labels of the training samples. Commonly used distance metric is the overlap metric or the Hamming Distance.

K is a user-defined constant, and an unlabeled test vector is classified by assigning the label which is most frequent among the *K* training samples nearest to that query point. 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. This implies that the parameter for setting the number of neighbors i.e., *n_neighbors* is the most important parameter in the *KNeighborsClassifier()* function. To improve the performance of K-NN method, parameter tuning can be done by applying the *GridSearchCV()* function on the existing model.

F. Ada Boost Classifier

This is another ensemble learning algorithm. Boosting methods build models sequentially and generate a powerful ensemble, which is the combination of several weak models. The core principle of AdaBoost is to fit a sequence of weak learners i.e., models that are only slightly better than random guessing, such as small decision trees on repeatedly modified versions of the data.

The predictions from all of them are then combined through a weighted majority vote to produce the final prediction. The data modifications at each boosting iteration consist of applying weights $w_1, w_2 \ldots . w_N$ to each of the training samples. Initially, those weights are all set to $w_i = 1/N$, so that the first step simply trains a weak learner on the original data.

For each successive iteration, the sample weights are individually modified, and the learning algorithm is reapplied to the reweighted data. At a given step, those training examples that were incorrectly predicted by the boosted model induced at the previous step have their weights increased, whereas the weights are decreased for those that were predicted correctly. As iterations proceed, examples that are difficult to predict receive ever-increasing influence. Each subsequent weak learner is thereby forced to concentrate on the examples that are missed by the previous ones in the sequence [7].

An *AdaBoostClassifier()* function 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. Some of its important parameters are *N_estimators* - the maximum number of estimators at which boosting is terminated, *base_estimator* - The base estimator from which the boosted ensemble is built and *learning_rate* – shrinks the contribution of each classifier by its value.

G. Decision Tree

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. A decision tree is a flow-chart-like structure, where each internal (non-leaf) node denotes a test on an attribute, each branch represents the outcome of a test, and each leaf (or terminal) node holds a class label.

The general motive of using Decision Tree is to create a training model which can be used to predict class or value of target variables by learning decision rules inferred from training data. Algorithms for constructing decision trees usually work top-down, by choosing a variable at each step that best splits the set of items.

DecisionTreeClassifier() is a class capable of performing multi-class classification on a dataset [8]. As with other classifiers, DecisionTreeClassifier takes as input two arrays, an array to hold the training data and another array holding the class labels for the training samples. It has many parameters which affect the accuracy of the model, some of which are *min_samples_split* - the minimum number of samples required to split an internal node, *criterion* - the function to measure the quality of a split, and *class_weight* - weights associated with classes in the form class_label: weight.

H. Neural Network

An artificial neural network learning algorithm, usually called Neural Network (NN), is a learning algorithm that is inspired by the structural and functional aspects of biological neural networks [9]. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process.

Neural networks are considered as non-linear statistical data modeling tools. They are usually used to model complex relationships between inputs and outputs, to find patterns in data, or to capture the statistical structure in an unknown joint probability distribution between observed variables. A Neural network can learn from observing data sets. In this way, a NN is used as a random function approximation tool. These types of tools help estimate the most cost-effective and ideal methods for arriving at solutions while defining computing functions or distributions.

NNs have several layers that are interconnected. The first layer consists of input neurons. Those neurons send data on to the second layer, which in turn sends the output neurons to the next layer and so on. NNs use sigmoid function like logistic regression in each layer.

Here we have used a Multi-layer Perceptron classifier or the *MLPClassifier()* function to implement neural networks. It is a supervised learning algorithm that learns a function *f(.):* R_m -> R_o by training on a dataset, where *m* is the number of dimensions for input and *o* is the number of dimensions for output. *Hidden_layer_sizes* - the *i*th element represents the number of neurons in the *i*th hidden layer, *activation* – activation function for the hidden layer and *solver* – the solver for weight optimization, are some of the main parameters of the function.

V. PERFORMANCE EVALUATION

In this section, we will evaluate the results of different classification models in terms of Accuracy Score, Confusion Matrix and Area Under ROC curve to see how well they classify an SMS as Spam or Ham. We have used eight different classification models - Logistic Regression, Support Vector machine, Neural Network, Random Forest, Decision Tree, Multinomial Naïve Bayes, KNN Classifier, Ada Boost Classifier. We have also plotted the confusion matrix and tabulated the accuracy for each model.

A. Environmental Setup

The models have been implemented in Python Language which was run on the Jupyter Notebook platform. We have used few python libraries for achieving different tasks as part of our implementation. Pandas library has been used for loading and handling the data from input csv data file. Scikit Learner library has been used for pre-processing modules and implementing classification model. NumPy and MatPyPlot libraries have been used for plotting the heat map and confusion matrix for each learning model. For visualizing the spam and ham words we have installed WordCloud.

Parameter	Values
Input data file	spam.csv
Number of columns	2
Number of Records	5574
Classification Models	Multinomial NB, Random Forest, Logistic Regression, SVM, KNN, Ada Boost, Decision Tree, Neural Network

Table 1. Systems Parameters

As Table 1 depicts, SMS Spam Collection dataset is used as input file for all the classification models. It is a set of SMSs tagged messages that have been collected for SMS Spam research. It contains one set of SMS messages in English of 5,574 messages, tagged as being ham or spam. The files contain one message per line. Each line is composed of two columns: v1 contains the label (ham or spam) and v2 contains the raw text.

B. Analysis of Classification Models based on Accuracy

Table 2 shows the accuracy of each classification model for the predictions. Though most of the models gives accuracy scores greater than 90%, Multinomial Bayes classifier and Neural Network give the best accuracy. K-NN model has the least accuracy of 95.33% which was improved to 96.94% by applying parameter tuning using *GridSearchCV* method.

Classification Model	Accuracy(Percent)
Multinomial NB	99.28%
Random Forest	97.66%
Logistic Regression	98.74%
SVM	98.74%
KNN	96.94%
Ada Boost	98.56%
Decision Tree	96.95%
Neural Network	98.92%

Table 2. Models and their Accuracy

Fig. 3 shows a bar graph comparing the accuracies of different models.

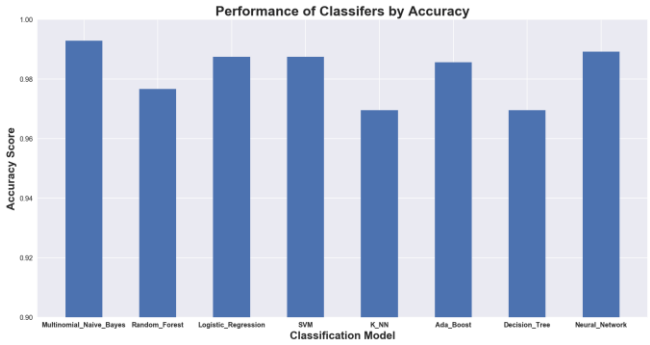

Fig. 3. Comparison of Accuracy Scores

C. Analysis of Models based on Confusion Matrices

In a Confusion matrix, each row of the matrix represents the instances in an actual class while each column represents the instances in a predicted class. The table reports the number of *True positives*, *False Positives*, *True Negatives* and *False Negatives*. Using these values, the *Precision* which is the fraction of relevant instances among the retrieved instances can be obtained [10]. Also as the name suggests, confusion matrix makes it easy to see if the system is confusing two classes.

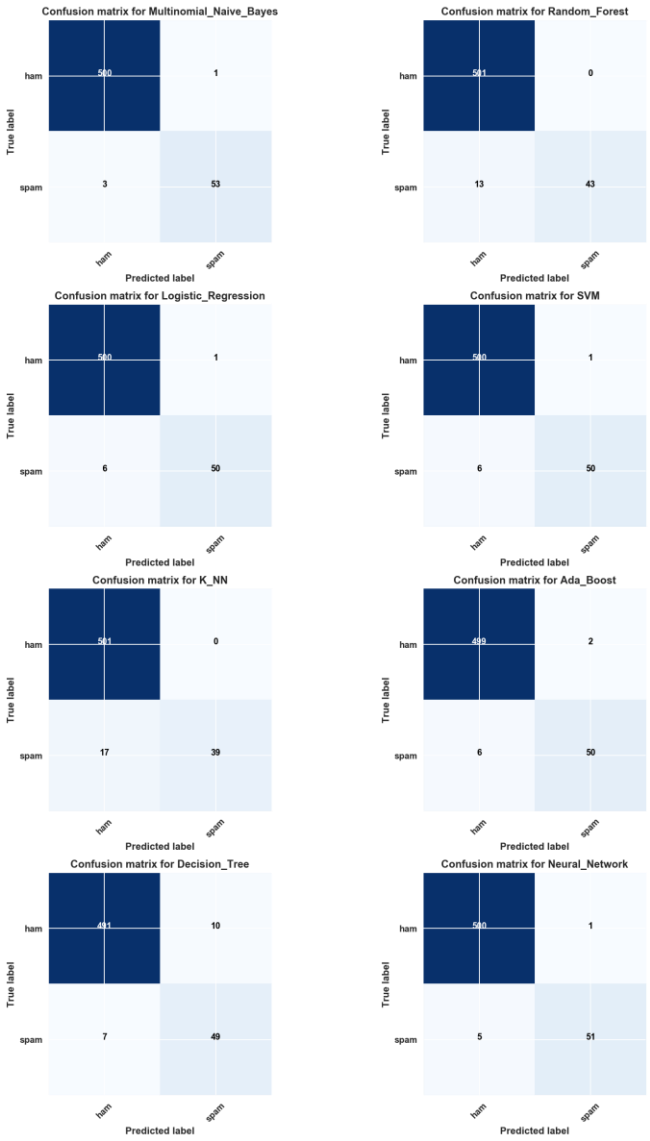

Fig. 4. Confusion Matrix for each model

Fig. 4 shows the Confusion Matrices generated by each classification model. Each model takes some unique parameters along the with the input data. The process of using K-fold cross validation to split the dataset leads to more accurate estimate of the model's performance.

D. Analysis of Models based on Area under ROC Curve

The AUC under ROC (Receiver Operating Characteristic) curve shows the tradeoff between sensitivity and specificity. It is one of the best depiction of a binary classifier. Fig. 5 shows the ROC curve for every model. The closer the curve follows the left-hand border first and then the top border of the ROC space, the more accurate the test is. The closer the curve comes to the 45-degree diagonal of the ROC space, the less accurate the test. The area under the ROC curve is a measure of text accuracy [11].

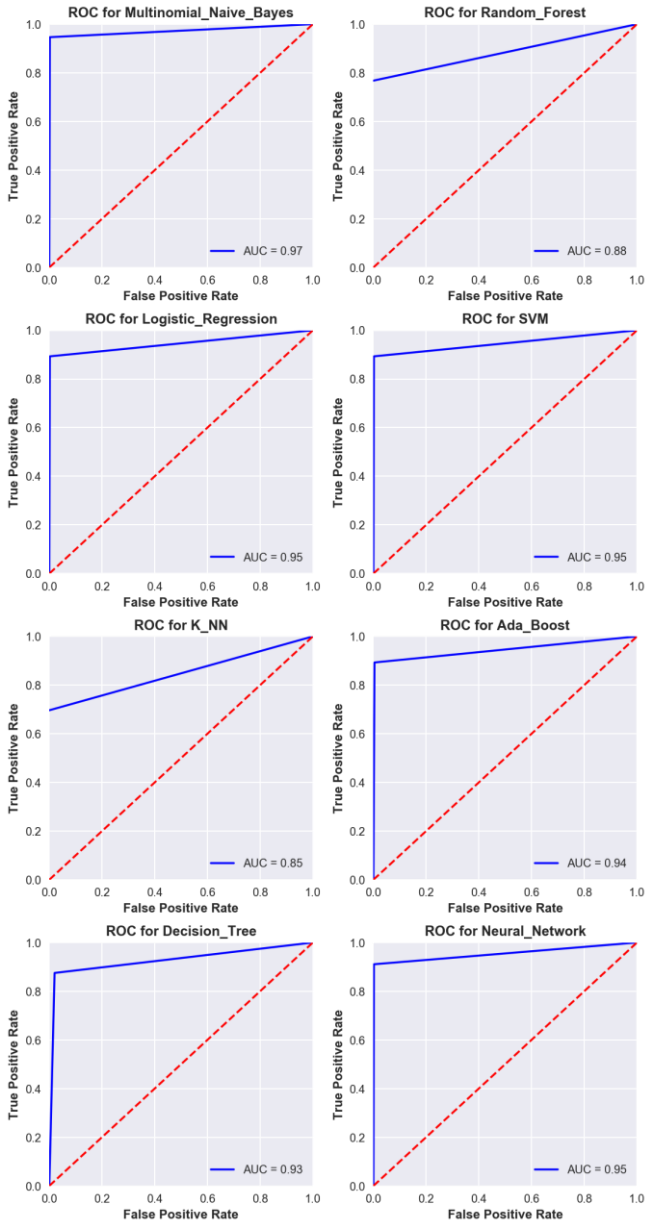

Fig. 5. AUC under ROC curve for each model

Table 3 shows the ROC scores of each classification model. Most of the models give a good ROC score but Multinomial Bayes classifier and Neural Network give the best scores.

Classification Model	ROC Score
Multinomial NB	0.97
Random Forest	0.88
Logistic Regression	0.95
SVM	0.95
KNN	0.85
Ada Boost	0.94
Decision Tree	0.93
Neural Network	0.95

Table 3. Models based on ROC score

E. Analysis based on Training/Testing Time

Two models that give a good accuracy and AUC are Multinomial Naive Bayes and Neural Network. On comparing the training/testing time of these two models we see that time taken by Multinomial Naive Bayes is much lesser than the time taken by Neural Network. Fig. 6 shows the same.

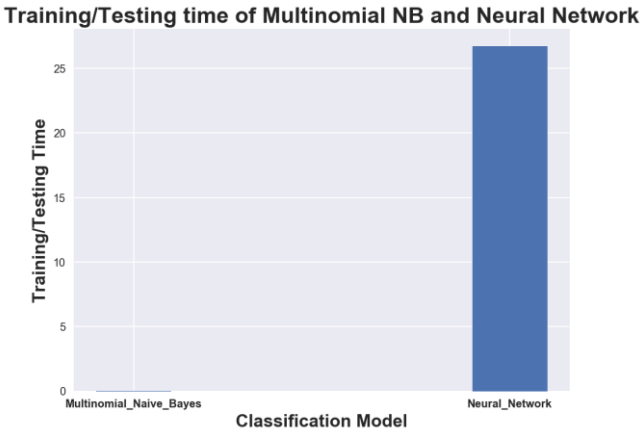

Fig. 6. Execution times for Multinomial NB and NN

VI. CONCLUSION

In this project, we experimented different classification models on SMS Spam Collection dataset and evaluated their results to see which model performs well to predict whether an SMS is Spam or Ham. Input dataset was preprocessed, split to training, validation and testing sets using K-fold Cross Validation techniques. Count Vectorization method was applied to transform the text of the SMS messages.

We employed eight different classification models for the prediction model. After comparing all the algorithms, we found that Multinomial Naive Bayes and Neural Network gave the best accuracies and AUC. However, on comparing these two models in terms of their execution times, we found that the time taken by Neural Network was much larger than the time taken by Multinomial Naive Bayes. Hence, we can say that the most suitable model for predicting whether an SMS is Spam or Ham is Multinomial Naive Bayes.

VII. REFERENCES

[1] Scikit-Learn Package Python - http://scikit-learn.org/stable/

[2] https://www.kaggle.com/datasets

[3] http://www.dt.fee.unicamp.br/~tiago/smsspamcollection/

[4] https://www.analyticsvidhya.com/blog/2015/11/improve-model-performance-cross-validation-in-python-r/

[5] S. V. N. Vishwanathan, and M. N. Murty. "SSVM: a simple SVM algorithm". In Neural Networks, 2002. IJCNN '02. Proceedings of the 2002 International Joint Conference on, Vol.3, 2393-2398, Honolulu,HI, USA, 2002

[6] http://cs229.stanford.edu/proj2013/ShiraniMehr-SMSSpamDetectionUsingMachineLearningApproach.pdf

[7] T. Hastie, R. Tibshirani and J. Friedman, “Elements of Statistical Learning Ed. 2”, Springer, 2009

[8] http://scikit-learn.org/stable/modules/tree.html

[9]https://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html

[10] https://en.wikipedia.org/wiki/Precision_and_recall

[11] http://gim.unmc.edu/dxtests/roc2.htm