Classification of spam comments through Text Mining & Machine Learning techniques

Vignesh Ramanathan, *Master of Science, Industrial Engineering,*
The Pennsylvania State University
vxr21@psu.edu

I. Introduction

The concept of spam originally comes about from emails. Spam email entails using electronic mailing systems to send unsolicited messages, especially advertising. Spam emails proliferated as electronic mailing systems became more prevalent in the late 1990s and early 2000s. In turn, email clients needed to find an efficient method for classifying emails as spam so that they are automatically filtered them out from useful mail.

Over time, spam began to proliferate into other forms of social media as well such as comments on social media websites. Classifying these comments and identifying nuisances has now become a more widespread problem to deal with. In particular, this is an issue on the comments section of the video sharing website YouTube. In addition to advertising and other unwanted promotions, there was an increase in proliferation of malicious links that could potentially steal valuable data from computers. This is a huge problem thanks to the popularity of YouTube, where several videos have been viewed over a billion times. Therefore, it is very important to be able to identify such comments beforehand. Being able to identify patterns in the given dataset would therefore allow us to perhaps come up with an automatic filter similar to what has been implemented for email. The key difference here is that traditional email spam filters use characteristics such as average word length, average number of characters in an email and so on to predict spam. Here, we try and look at the words themselves to see if any insights can be gained.

Text mining (according to Wikipedia) is the process of deriving high-quality information from text. This information is usually obtained through commonly used machine learning techniques such as logistic regression, classification trees and so on. While different text mining techniques exist, the method of interest here is the 'bag of words technique', where a matrix of word frequencies is obtained from a sample of the text to be analysed.

The aim here is to see whether text mining techniques can be used to successfully classify comments as either 'spam' or 'ham' (not spam). It is to be noted that these comments have already been pre-classified as either spam or ham by human judgement earlier. Hence, this entire exercise would be a form of 'supervised learning'. In addition, this project is based on previous work in 2015 that used several known algorithms to address the given problem. I will be using some additional techniques apart from what has already been performed. In the original papers, algoritms were performed separately on five different data sets and were evaluated. I will try to combine these data sets and apply the same techniques to see if algorithm performance scales up with an increase in data.

II. Literature Review

The idea of gaining insights from unstructured data in the form of text is not new. As early as 1958, a rudimentary process quite similar to modern text mining was hypothesised by HP Luhn at IBM. He proposed a system known as a 'Business Intelligence System'. Until that time, information retrieval through computers were largely restricted. For any computer to produce any meaningful output given a certain amount of input, information had to be processed extensively by humans. This was as computers then were extremely limited in function. Luhn hypothesised of Business Intelligence Systems that would be capable of automatically abstracting documents, automatically encoding these documents and the automatic creation and updating of 'action-point profiles'. 'Action point profiles' refer suitable information to support specific activities carried out by individuals, groups, divisions or organisations.

This proposed Business Intelligence System would selectively provide only information that was relevant to a given query, by means of a profile generated per individual entity that is using the system. This proposed system would take in a document as input. It would then generate an 'auto abstract' of the given data. This auto abstract would be obtained by ascertaining the frequency of word occurrences in a document. A certain portion of words with highest frequency would be given a status of 'significant words'. Using these 'significant words', sentences would be analysed and classified into 'significant sentences' according to the frequency of these significant words. These sentences, ranked by significance and separated by a threshold significance would then be extracted from the text to create an 'auto abstract'. Once this abstract had been obtained, further processing would be done on this abstract to derive an information pattern that characterises this document. While exact specifications regarding these statistical analyses were not mentioned distinctly (due to the lack of widespread statistical computing power in those days), a potential further classification of words by means of a thesaurus is mentioned. A repository of documents would be fed into this machine and auto-abstraction would be performed on all these documents. Whenever an action is desired, the system generates an information pattern regarding the client and/or action required and attempts to match it with those of the auto-abstracts. Good matches would then be retrieved for further analysis.

The paper above is one of the first records mentioning the possibility of text analytics and gaining insights through unstructured data. Several steps described in the operation of the proposed ‘Business Intelligence System’ are very similar to the underlying principles behind modern machine learning and their applications.

As years progressed, computers became much more powerful and data analytic techniques rose to prominence. This surge began in the late nineties. By the 2000s, these techniques were starting to see regular usage for different kinds of problems, as hardware capabilities could now match the computing power required to process larger data sets. A good summary of the pre-processing involved in a text mining problem and algorithms used in such problems was provided by Hotho, Nurnberger and Paas in 2005. This paper provides an exhaustive review of different aspects of text mining. I’ve discussed below only insights that were relevant to this project and not the entire paper.

This paper discusses the process of Knowledge Discovery In Databases (KDD) and differentiates it from conventional data mining. According to this ”Knowledge Discovery in Databases (KDD) is the non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data”. The main difference between KDD and normal data mining is that KDD requires several preprocessing steps to transform data to a useful form before performing the family of techniques known as data mining to gather potentially useful insights. KDD has a total of six steps – business understanding, data understanding, data preparation, modelling, evaluation and deployment.

Text Mining here is defined as Text Data Mining or the application of algorithms and methods from the fields machine learning and statistics to texts with the goal of finding useful patterns. Preprocessing is performed on data by means of information extraction methods, natural language processing or some simpler preprocessing steps in order to extract data. To this extracted dataset, data mining algorithms are applied. Natural Language Processing refers to a range of techniques from string manipulation manually to automatic processing of natural language enquiries (for more advanced or ‘intelligent’ systems). Information Extraction refers to the retrieval of specific information from texts, very similar to auto abstraction and the matching of profiles to action profiles as discussed by Luhn.

Preprocessing is usually performed by the ‘bag of words technique’. This technique observes data merely by the frequency of appearance of certain terms (usually words, or string characters preceded and succeeded by a space). Hence, ‘mary had an apple’ and ‘apple mary an had’ would be two strings that are treated identically by the bag of words techniques. A given string is separated into these words before algorithms are applied. Additional processing is performed such as filtering. This is usually in the form of removal of commonly occurring words in the English language, referred to as ‘stop words’. These words usually do not provide valuable insight when trying to analyse data and are mere noise. Libraries of such stop words are usually specified in commonly used languages such as Python or R.

Some more preprocessing techniques used are ‘lemmatization’ and ‘stemming’. ‘Lemmatization’ refers to the breaking down of all forms of tenses, parts of speech and other grammatical rules and converting these complex words to their roots. However, due to the intricacies of the English language, this would be very time consuming and cumbersome due to the many variations present in words and their spellings. An easier alternative is ‘stemming’, where certain commonly occurring characters are cut off to give an approximate root word. Hence, an example would be ‘providing’ being broken down to ‘provid’ through stemming. Lemmatization would break it to the more correct provide. However, obtaining ‘provide’ from ‘providing’ & ‘provided’ is much harder than obtaining ‘provid’ from the same. Hence, we see that stemming is preferable.

Following these steps, algorithms are then performed on the transformed data obtained by ‘bag of words’. For classification problems, it is noted that common algorithms used are the naïve Bayes classifier, k-Nearest Neighbour Classifier, Decision Trees (Usually C4.5), Support Vector Machine Methods and Trees boosted by using techniques such as adaboost. Hotho, et. al tested the performance of the above techniques to a classification problem involving Reuters20 newsgroup collection. According to their findings, the algorithms were ranked in increasing order of their performance as follows – Naïve Bayes Classifier, C4.5 Decision Tree, k-Nearest Neighbours, SVM and boosted trees. In addition, they also specify commonly used clustering techniques used for text mining problems. These include hierarchical clustering, k-means clustering, bi section k-means, Self Organizing map (SOM) and model based clustering using the Expectation Maximization (EM) Algorithm.

The most commonly used Stemming algorithm is the algorithm developed by Porter in 1980. This broadly consists of examining words for vowels and/or consonants, and breaking down these words into sub-words, eliminating parts at the end if certain conditions obtained with patterns gotten through observing the vowel/consonant combinations. A detailed explanation is provided in his 1980 paper.

The problem of interest was first examined by Alberto, Lochter and Almeida in 2015. They analysed comments taken from YouTube videos and tried to classify them as spam or ham depending on their content. According to their work, a comment was considered to be spam only if it were a potentially malicious link. Hence, traditional self-promotions were considered as ‘not spam’. These comments were manually classified as spam and ham. Several algorithms (CART, k Nearest Neighbours, Logistic Regression, Bernoulli Naïve Bayes Classifier, Gaussian Naïve Bayes Classifier, Multinomial Naïve Bayes Classifier, Random Forests, and Support Vector Machines with linear, polynomial and Gaussian kernels) were implemented to try and classify these emails. While different performance measures were implemented, it was found that the best ranking methods according to different measures were CART, Logistic Regression, Naïve Bayes-Bernoulli, Random Forests, SVM-Linear and SVM-Gaussian kernel. These results were computed using Python 2.7, implementing the scikit-learn package.

Among the techniques used by Alberto, Lochter and Almeida, neural networks are absent. However, neural networks that update weights on the basis of backpropagation (As proposed by Rumelhart, Hinton and Williams in 1988) have been used for similar problems. This was used by Clark, Koprinska & Poon (2003) for email spam classification. In this paper, an email classifier called LINGER was built that classified emails into spam and ham through BNNs. This was done on a dataset that was extracted through a corpus and bag of words as referred to earlier. In addition, it was found that Neural Networks were either comparable to or outperformed Naïve Bayes classifiers, boosted trees and k-Nearest Neighbours for this data set.

III. Techniques to be used

With respect to the exact techniques used for text mining, I will be using the “tm” package in R. This was developed by David Meyer, Kurt Hornik and Ingo Feinerer in 2008. This contains commands to perform stemming, removal of stopwords, punctuation and other preprocessing techniques described above. Stemming is performed via Porter’s stemming algorithm, which is referenced. The creation of corpuses and ‘document term matrices’ are also performed using commands from this package.

With respect to the classification techniques, the following will be implemented -

A. Logistic Regression

It is a regression model where the dependent variable is categorical. In the case of our data, we have binary dependent variable that is, only two possible values, "0" and "1". Cases where the dependent variable has more than two outcome categories may be analyzed in multinomial logistic regression, or, if the multiple categories are ordered, in ordinal logistic regression. The binary logistic model is used to estimate the probability of a binary response based on one or more predictor (or independent) variables. The sample distribution and link function used to ensure proper probabilities is:

$$Y = 1|X \sim Bernoulli(p_1) \text{ where } p_1 = E(Y=1|X) = p(Y=1|X)$$

$$g(p_1) = \beta_0 + \beta X, \text{ a logistic function: logit } p_1 = \log_e^{\frac{p_1}{1-p_1}} = \beta_0 + \beta X$$

It allows one to say that the presence of a risk factor increases the probability of a given outcome by a specific percentage. To do that logistic regression first takes the odds of the event happening for different levels of each independent variable, then takes the ratio of those odds (which is continuous but cannot be negative) and then takes the logarithm of that ratio. This is referred to as logit or log-odds to create a continuous criterion as a transformed version of the dependent variable. Thus the logit transformation is referred to as the link function in logistic regression—although the dependent variable in logistic regression is binomial, the logit is the continuous criterion upon which linear regression is conducted.

The logit of success is then fitted to the predictors using linear regression analysis. The predicted value of the logit is converted back into predicted odds via the inverse of the natural logarithm, namely the exponential function. Thus, although the observed dependent variable in logistic regression is a zero-or-one variable, the logistic regression estimates the odds, as a continuous variable, that the dependent variable is a success (a case). Logistic regression estimates are a nonlinear function of x which is guaranteed to range from 0 to 1.

B. Linear Discriminant Analysis

Linear Discriminant Analysis (LDA) is built upon the notion of "inverse thinking" where we examine the distribution of the predictors (X) given that the observation is in class G. In other words, $f_k(X) = f(X|G=k)$.

Here, we assume the predictors (X), given being placed into group k, follow a multivariate Gaussian distribution:

$$f_h(x) = \frac{1}{(2\pi)^{p/2}|\Sigma|^{1/2}} \exp\{-\frac{1}{2}(x-\mu_k)'\Sigma^{-1}(x-\mu_k)\}$$

where μ_k is the sample mean of the predictors for group k and Σ is the common covariance. We can use the within class sample means and the within class sample covariance to estimate μ_k and Σ respectively. Note that in LDA, the Σ covariance is common among all groups. So, the predictors only differ in their mean vectors.

The linear discriminant function is defined as:

$$\delta_k(x) = x'\Sigma^{-1}\mu_k - \frac{1}{2}\mu_k'\Sigma^{-1}\mu_k + log(\pi_k)$$

Here, μ_k is the mean of class-k, $\pi_k = N_k/N$ where N_k is number of class-k and N is the total , and Σ is the common covariance matrix within a class. We can estimate these values using the samples from the training data. The decision boundary between class k and I are the points $x : \delta_k(x) = \delta_I(x)$.

The LDA method works well when we assume the predictors (X) follow a Gaussian distribution and there are common covariances within each class (G).

C. Classification and Regression Trees (CART)

A tree based method can be used to predict the expected value of either a qualitative response (classification tree) or a quantitative response(regression trees). However, as our given data set has a qualitative response, we shall focus on classification trees. We predict that each observation belongs to the most commonly occurring class of training observations in the region to which it belongs. In interpreting the results of a classification tree, we are often interested not only in the class prediction corresponding to a particular terminal node region, but also in the class proportions among the training observations that fall into that region and we use recursive binary splitting to grow a classification tree. Since we plan to assign an observation in a given region to the most commonly occurring class of training observations in that region, the classification error rate is simply the fraction of the training observations in that region that do not belong to the most common class:

$$E = 1 - \max_k(\hat{p}_{mk})$$

Here $\hat{p}_{mk}$ represents the proportion of training observations in the mth region that are from the kth class.

The Gini index is a measure of total variance across the K classes. The RandomForest and tree functions in R use the Gini Index as a means of determining node purity. We can see that the Gini index takes on a small value if all of the $\hat{p}_{mk}$'s are close to zero or one. Hence, this index is a means of measuring node purity — a small value indicates that a node contains predominantly observations from a single class:

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

D. Random Forest Classification

Random Forests grow many classification (or regression) trees and combine them together for an aggregated result. To classify a new object from an input vector, we place the input vector down each of the trees in the forest. Each tree gives a certain classification as output, and we say the tree "votes" for that class. The forest then chooses the classification having the most votes (over all the trees in the forest). Therefore, the random forest algorithm is an ensemble learning method for classification, regression and other tasks, that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of over-fitting to their training set. Each tree is grown as follows:

1) If the number of cases in the training set is N, sample N cases at random - but with replacement, from the original data. This sample will be the training set for growing the tree.
2) If there are M input variables, a number m«M is specified such that at each node, m variables are selected at random out of the M and the best split on these m is used to split the node. The value of m is held constant during the forest growing. The R function RandomForest by default uses this value m as N/3, where N is the total number of potential predictor variables in the input data.
3) Each tree is grown to the largest extent possible. There is no pruning.

E. K-Nearest Neighbours

This is a classification technique directly dependent on the nearest neighbours of a new item entering a 'feature space' where all the data points lie. We define distance on input x, such as Euclidian distance. Assume that x* is a new object entering our feature space waiting to be classified. We then find K training samples $\mathrm{x_r}$, r = 1,...,K closest in distance to x$*$ and then classify using majority vote among the k neighbors.

$$Y* = \frac{1}{K}\sum_{r=1}^{K} y_r$$

It is to be noted that this technique is extremely sensitive to K. A low value of K means that a high likelihood of overfitting occurring is possible. On the other hand, a high value of K will make classification smoother. However, we may lose out classifying power in this case. Also, the computational power required will now increase. Also, it is advisable to normalize the data before performing KNN classification, as this technique is quite sensitive to the scale of data.

F. Boosting

Similar to random forest classification, boosting captures local features of the data; thereby making it a a very flexible classification method. Boosting employs an iterative approach where multiple decision trees are grown *sequentially*. Moreover, each decision tree is fitted to a modified (or re-weighted) version of the original dataset. The main idea underlying the boosting method is that each observation is initially assigned a weight. After each iteration, we assign larger weights to the more difficult-to-classify observations. This method is a "slow" learning process where we begin with a "weak" classifying algorithm for the first decision tree, and then sequentially improve (or "boost") the algorithm until it evolves into a "stronger" classier.

Here, we implement the adaptive boosting (adaboost) method originally proposed in Freund and Schapire (1996). Initially, the observations are assigned equal weights $w_i =$

$\frac{1}{N}, i = 1, 2, ..., N$. For each iteration $m = 1, ..., M$, we fit a classifier $G_m(x)$ to the training data with weight w_i. Then, we calculate the *combined weight* α_m:

$$\alpha_m = \frac{\sum_{i=1}^{N} w_i I[y_i \neq G_m(x_i)]}{\sum_{i=1}^{N} w_i}$$

Note that the indicator function ($I[y_i \neq G_m(x_i)]$) is true when misclassification occurs, so the more difficult-to-classify observations will have larger weights in the following iterations.

Next, the incorrectly classified observation are assigned a larger weight from the previous observation. The new weight is $w_i^{new} = w_i \frac{1-\alpha_m}{\alpha_m}$. After running M iterations (with M classifiers), we create the final classifier using the fitted values and combined weights α_m from each iteration. The final classifier is:

$$G(x) = sign[\sum_{m=1}^{M} \log(\frac{1-\alpha_m}{\alpha_m} G_m(x)]$$

Advantages of boosting are (1) its flexibility to pinpoint local features of the data; (2) there is built-in dimension reduction because it only selects features known to improve prediction; and (3) smaller number of trees are required because each tree is built upon the results of the previous tree. Drawbacks of boosting include (1) overfitting because there is no formal stopping rule; and (2) sensitivity to "noisy" data where the observational data are incorrect. In the latter case, the "noisy" observations drive overfitting.

G. Back propagated Neural Networks

The derivation for this is quite complex and is described in the paper by Rumelhart, Williams and Hinton in 1986 (see references). To give a superficial explanation, a neural network is composed of several 'threshold logic units', which perform some sort of rule based classification based on a certain threshold. They take in as input a linear combination of input values, along with certain factors known as 'weights'. The result of this is compared against a threshold value and classification is accordingly performed. For the next iteration of this technique, the weights and threshold value are updated. The updation of these values is directly proportional to the 'error' or the difference between our desired and achieved values. The exact expression is obtained through a gradient descent technique. This algorithm continues until we see a convergence in the weights and threshold values.

Let j denote output layers and k denote input layers. Assuming that there are n input nodes, for each neuron j, output is defined as -

$$o_j = \phi(\sum_{k=1}^{n} w_{ij} o_k)$$

ϕ here represents an activation function, which is necessarily non-linear and differentiable. A commonly used function for this purpose is the logistic function (The same logistic function used in logistic regression). In multiple layer networks, the outputs of one layer are used as input for the new layer. For classification purposes, this o_j is compared against a certain threshold value θ. If o_j exceeds θ, a certain class is assigned. Else, another class is assigned. (The same logic can be extended for multiple classes). This process is repeated iteratively until convergence is reached (Or for a fixed number of iterations). In every iteration, the values of θ and the weights (w_{ij}) for every i and j are updated. Backpropagation refers to the method by which they are updated, which is through gradient descent. Assume that t is the target response to be achieved and y is the actual response achieved. We then define a squared error E such that -

$$E = \frac{1}{2}(t - y)^2$$

Initial values for w_{ij} and θ are chosen randomly and one iteration of the neural network occurs. For the next iteration, we update weights by the following rule -

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}$$

η is a learning rate parameter that varies between zero and one and is set by the user. Conventionally, a value of 0.3 is used to not make the system too nervous. I will be using the infrastructure available in R's "RSNNS" package. This uses backpropagation algorithm to update TLUs in every iteration.

IV. Data set description

The dataset chosen for this project is the YouTube Spam data set. This dataset is publicly available for usage through the UCI Machine Learning Repository (Link - https://archive.ics.uci.edu/ml/datasets/YouTube+Spam+Collection) This contains a sample of the list of video comments from the five most popular videos on YouTube at the time of which this data set was mined. All five of these videos are music videos. These comments were mined using the YouTube Data API, v3. The five videos in question are as follows –

1) PSY - GANGNAM STYLE M/V
2) Katy Perry – Roar (Official)
3) LMFAO - Party Rock Anthem ft. Lauren Bennett, GoonRock
4) Eminem - Love The Way You Lie ft. Rihanna
5) Shakira - Waka Waka (This Time for Africa) (The Official 2010 FIFA World Cup^{TM} Song)

Each dataset contains five attributes – A unique comment ID (A character string), The Author of the Comment (A character string), The Date of the comment (A character string), The Content of the comment (A character string) and finally a classifier category denoting as to whether the given comment is spam or not. This is a 0-1 binary integer variable with '0' defined as 'not spam' and 1 as 'spam'. This classification has been performed manually during the preparation of the data set. The fact that this classification has already taken place means that it is possible to implement Supervised Machine Learning techniques on these data sets.

As previous manual classification has already taken place, we have an idea of the composition of each data set with respect to spam and ham (not-spam). With respect to particular methodology behind what is considered spam or not spam for this study, please refer the earlier section. The five individual datasets can be summarised by the following table –

Data set	Number of Comments	Spam	Ham
PSY	350	175	175
Katy Perry	350	175	175
LMFAO	438	236	202
Eminem	448	245	203
Shakira	370	174	196

Table I: Description of given data sets (%)

As we can see, all the given data sets are very balanced. Therefore, a naive prediction of all items being classified as spam or ham would yield a maximum accuracy of only 54.69% (In the case of the Eminem data, classifying every single comment as 'spam'). This means that there's not too high a threshold required for any algorithm to work better than a naive guess.

V. Analysis and Results

As is the case with any text mining problem, extensive preprocessing is required before any techniques can be applied on the data set. Each data set's source file comprises of a unique comment ID, the author of the comment, the time at which this comment was made, the comment as text and the classification of this comment as either spam or ham. As we are concerned only with content analysis for this project (As was the case with the analysis of , we extract only the content from this source data frame and the response variable and perform analysis.

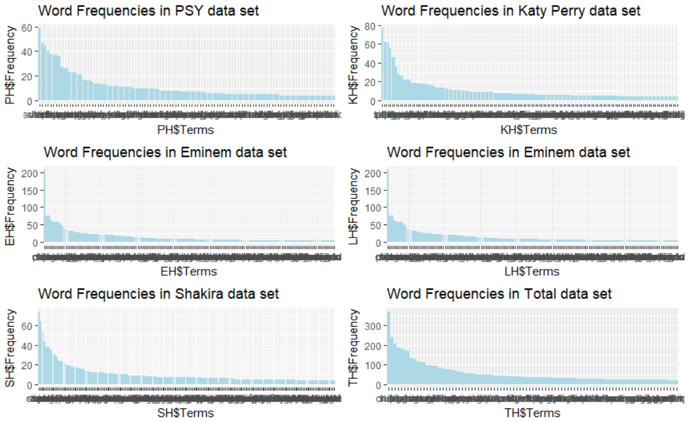
After extracting text, we prepare a 'corpus', which is a data structure specifically design to handle large amounts of text data. This text is usually filled with a lot of junk which is not helpful towards classification. The following preprocessing steps were to be performed on all of the data sets (by means of the tm_map function from the tm package) -

1) Conversion of all characters to lowercase, in order to avoid unnecessary duplication
2) Removal of punctuation characters as these do not aid analysis
3) Removal of numbers from the data set
4) Stemming the document through Porter's stemming algorithm to ensure that all different forms of the same word are classified together
5) Removal of stopwords (Commonly occurring words such as 'as', 'in', 'with', etc), using the stopwords repository available in the tm package

It is to be noted that all of these steps were not successfully implemented on every data set. I did see the occurrence of special characters in different data sets when inspecting word clouds and the document term matrices. This is possibly due to some bugs with the R function that is to handle such tasks. In addition, removal of Punctuation caused errors particularly with the Shakira data set. My resultant corpus after the step of removal of punctuations could not be converted to a document term matrix. A similar problem existed with the aggregated data set. As a result, some of the elements have special characters that led to errors with implementing some of the techniques proposed.

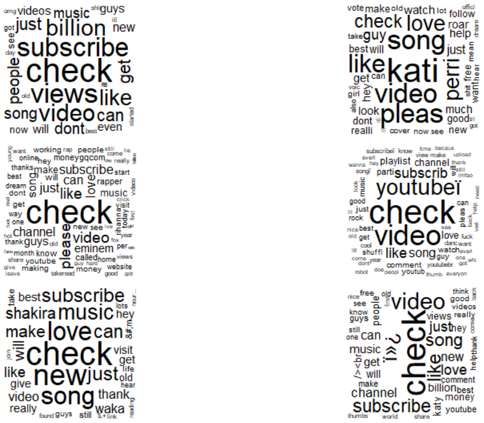
Having obtained corpuses, we see that all of them are very sparse matrices (About 99% sparse). This is usually the case with text mining problems. Hence each data set was reduced by 99% and columns with respective sparse terms were removed. After these steps have been performed, a 'document term matrix' is created from the resultant corpus. This document term matrix is a frequency matrix with columns corresponding to the frequency of words occurring for that particular comment. Even after reducing for 99% sparsity, the resultant matrix is still very sparse. To illustrate are histograms with word frequencies for each of the 6 data sets -

Figure 1: Histograms depicting word frequencies for each data set, with terms arranged in descending order



To have an idea of the words that are present in the resultant document term matrices, word clouds are constructed, which are images that depict the frequency of the occurrence of a particular word in a matrix through its size. The larger the word, the higher the frequency of that word's occurrence in all of the comments in a particular data set. Here are resultant word clouds for all 6 data sets -

Figure 2: Wordclouds from each data set. (Clockwise from top right) PSY, Katy Perry, LMFAO, Total data set, Shakira, Eminem



As we can see, many comments seem to be 'self-promoting', asking users to subscribe to their own channels. A quick look at the classification already provided manually shows that such self promoting comments are usually classified as 'spam'. The high presence of such words indicates that there would be quite a lot of spam in these data sets.

Having prepared out data sets, we can now apply commonly used algorithms to build models to predict spam. The algorithms that I'll be using are -

1) Logistic Regression
2) Linear Discriminant Analysis
3) K-Nearest Neighbours
4) Classification Trees (CART)
5) Random Forests
6) Boosted Trees (Using Adaptive Boost)
7) Backpropagated Neural Networks (Multi-layer perceptrons)

It is worth mentioning that Logistic Regression, CART, K-Nearest Neighbours and Random Forests have already been tested out on this data set in the 2015 paper by Alberto, Almeida & Lochter. Hence I'm trying out these techniques. As Random Forests are being implemented, it would be a natural extension to try out boosted trees, as these techniques are similar in principle (where Boosting involves learning through the residuals of the model rather than the outcome of the model itself, and resampling from trees without substitution). While it usually is the case that Random Forests perform better than Boosting or single trees, this difference in performance is traded off for an increase in time taken to compute this model. Finally, while Neural Networks have not been used in the original paper, they have been used for email spam classification by Clark, et. al (2003) and were found to outperform other algorithms. I would like to see if the same applies here.

Logistic Regression usually is inferior to tree based methods and usually has a problem with overfitting data to the training sample. It requires less computation usually. LDA is usually very light computationally, but would work well only if the given items to be classified lie along a linear decision boundary in our feature space. This may not be guaranteed. K-Nearest neighbours are heavily biased by existing members in the existing feature space. In addition, with a very large data set, their computation becomes quite expensive (Which is thankfully not the case here). They do not scale up due to the large number of distance computations and comparisons involved. However, they are very intuitive and easy to understand as a technique.

CART is one of the most intuitive methods to understand, but can usually suffer from overfitting as it is sensitive to biases in the data. It is usually not expensive computationally either. This bias is reduced in Random Forests and Boosting, although they still can sometimes overfit data. These are much more computationally intensive techniques. Finally, neural networks usually have been seen to perform well on a wide variety of classification programs, even if the underlying data does not make too much sense. However, neural networks can be prohibitively expensive with an increase in nodes and hidden layers.

With respect to implementation, the following parameters were used -

1) K = 3 and K = 5 for Nearest Neighbours, as in the original paper.
2) CART - default values with rpart function, implying a minimum of 20 items before attempting a split, a minimum 'bucket size' of 7 (Nuber of entitites in a terminal leaf node), Cp = 0.01, 10 cross-validations, maximum tree depth of 30.
3) Random Forests - Default parameters for the randomForest function in R, with 500 trees grown, the number of predictors randomly sampled at each split being one-third of the total number of predictors and sampling occurring with replacement
4) Adaptive Boost - Default parameters in the ada function in R, which is 50 iterations, shrinkage parameter of 1.
5) Backpropagated Neural Networks (Multi-layer perceptrons) - The number of nodes in each layer varied. Every neural network had 2 hidden layers. I roughly assigned the number of nodes per hidden layer depending on the number of predictor variables. I found through trial and error that roughly two thirds of the number of input variables taken as number of nodes for first hidden layer and half to two thirds of this as input for the next hidden layer yielded good results. Learning rate = 0.3 and number of iterations = 100.

For every algorithm used, testing and training sets were the same, with a random split of data into 75% for training and 25% for testing.All computations were performed on a system with an intel i5 processor and 8GB of RAM.

To evaluate algorithm performance, I used three parameters used in the original paper - Prediction Accuracy (Proportion of total comments classified correctly as either spam or ham), Blocked Ham (BH) Rate, which would be the proportion of non-spam emails that were blocked (Or False Positive Rate) and Spam Caught (SC) Rate, which would be the proportion of spam mails in the data set that were caught (Or True Positive Rate). Accuracy, TPR and FPR are standard metrics for any classification problem. For email classification, BH rate is much more important as the relative cost of missing out on an important mail is higher than seeing unwanted spam in inbox. For this problem though, I would say that they are both important metrics. In addition, I recorded system time elapsed for each algorithm and data set combination. **Hence for an ideal case, one would want very high accuracy, low BH rate, high SC rate and low system time elapsed.**

Results after performing these algorithms on respective data sets are summarized by the tables below -

Algorithm	Accuracy (%)	BH Rate (%)	SC Rate (%)	Time (s)
LR	76.14	31.82	84.09	0.21
LDA	69.32	43.18	81.82	0.05
CART	72.72	9.09	54.55	0.08
AdaBoost	77.27	6.82	61.36	2.21
RF	85.23	25.00	95.45	1.13
3NN	71.59	54.55	97.73	0.03
5NN	65.91	68.18	100.00	0.01
Neural Net	82.95	26.83	91.49	3.17

Table II: Performance of Algorithms on PSY dataset

Algorithm	Accuracy (%)	BH Rate (%)	SC Rate (%)	Time (s)
LR	79.46	21.57	80.33	0.31
LDA	82.14	13.73	78.69	0.05
CART	91.96	0.00	85.25	0.09
AdaBoost	88.39	7.84	85.25	3.65
RF	92.86	5.88	91.80	0.92
3NN	84.82	0.00	72.13	0.03
5NN	83.93	0.00	70.49	0.03
Neural Net	93.75	0.00	89.06	16.08

Table III: Performance of Algorithms on Eminem dataset

Algorithm	Accuracy (%)	BH Rate (%)	SC Rate (%)	Time (s)
LR	82.80	8.16	72.72	0.41
LDA	84.95	4.08	72.72	0.05
CART	89.25	0.00	77.27	0.11
AdaBoost	87.10	0.00	72.72	3.65
RF	-	-	-	-
3NN	-	-	-	-
5NN	-	-	-	-
Neural Net	85.87	5.88	73.80	7.06

Table IV: Performance of Algorithms on Shakira dataset

Algorithm	Accuracy (%)	BH Rate (%)	SC Rate (%)	Time (s)
LR	90.83	6.00	88.14	0.11
LDA	98.27	0.00	96.61	0.03
CART	96.33	0.00	93.22	0.05
AdaBoost	94.50	0.00	89.83	1.47
RF	97.25	0.00	94.92	0.70
3NN	90.83	0.00	89.83	0.02
5NN	94.50	0.00	83.05	0.02
Neural Net	91.74	4.35	85.94	1.87

Table V: Performance of Algorithms on LMFAO dataset

Algorithm	Accuracy (%)	BH Rate (%)	SC Rate (%)	Time (s)
LR	60.23	27.27	47.72	0.19
LDA	68.18	31.82	68.18	0.10
CART	70.45	4.55	45.45	0.11
AdaBoost	69.32	9.09	47.72	3.33
RF	-	-	-	-
3NN	-	-	-	-
5NN	-	-	-	-
Neural Net	84.09	4.35	90.24	6.67

Table VI: Performance of Algorithms on Katy Perry dataset

Algorithm	Accuracy (%)	BH Rate (%)	SC Rate (%)	Time (s)
LR	83.44	5.63	72.65	0.32
LDA	82.15	4.33	68.80	0.13
CART	82.15	3.46	67.95	0.42
AdaBoost	82.37	3.03	67.95	8.93
RF	-	-	-	-
3NN	-	-	-	-
5NN	-	-	-	-
Neural Net	84.30	9.33	78.33	11.95

Table VII: Performance of Algorithms on Total aggregated dataset

VI. Inferences and Insights

Irrespective of the method used and the data set, every algorithm does seem to have some predictive power and is more powerful than a naive classification. Predictive power of all these techniques seems very good (varying between 70-90%), but inferior to the results obtained in the original paper. This is probably thanks to better hyperparameter tuning in the work of the TubeSpam paper.

Irrespective of the algorithm that is used, the most important factor is the data itself. If no patterns can be seen in the data given, algorithm performance is not good by any metric for any technqiue. This can be seen in the differences between the Katy Perry data set (where all algorithms did not perform as well) and the LMFAO data set (where all algorithms performed really well), implying that the LMFAO data had underlying patterns that were easy to detect, unlike the KP data set. This same phenomenon was observed in the original TubeSpam paper too. The original TubeSpam paper had algorithms performing very well on the Shakira data set, which can be seen to some extent here (With respect to the algorithms that were successfully implementable).

No one technique completely dominates the rest in terms of any performance metric. Some broad trends however can be observed. Logistic Regression is quite fast as an algorithm, but generally has relatively poor to average accuracy. In addition, it seems particularly susceptible to a higher BH rate than other techniques that have been used. It seems to perform poorly to averagely with respect to SC rate across all data sets, particularly poor with the Katy Perry data set where less than half the spam was detected. In addition, we see that for no data set was it ever the best technique with respect to any of the chosen metrics.

Linear Discriminant Analysis is a very fast technique and does not really require much tuning. I was surprised at the performance yielded and did not expect as good results. It gives comparable or better results to Logistic Regression across every data set, which I found surprising as Logistic Regression is a popular technique for classification. It has relatively average to good accuracy, but average to poor BH rate. This is with the exception of the LMFAO data where it did not block out any ham. However, the LMFAO data had several algorithms perform with 0 BH rate. With respect to SC rate, it performs averagely on the PSY data set, poorly on the Eminem data set, poorly on the Shakira data set, well on the LMFAO data set, well on the Katy Perry data set and poorly on the total data set. There does not seem to be any consistency here.

CART does not take much time to compute. One big positive of CART is that we see a definite model with respect to each implementation, making it very easy for somebody to gain a superficial understanding of the given data. (Trees attached in appendix). With respect to performance, it has relatively average to good (Eminem) accuracy and actually has the best performance in the Shakira data set. It consistently seems to have low BH rates across all data sets, but has a tendency to suffer from a low SC rate, performing the worst in three data sets. This is especially seen in the PSY data set with a very poor SC rate. Oddly enough, it has the best SC rate with the Shakira data set.

Boosting does not seem to offer an advantage over single trees here. This is quite a surprising result. It is computationally quite expensive too, consistently being the second worst technique in terms of computational time elapsed.Classification accuracy, while never consistently the best, seems to range from average to good across all data sets. It consistently performs very well with respect to BH rate, ranging from a good to the best performance in this metric across all data sets. With respect to SC rate, it performs averagely to poorly across all data sets.

Random Forests was implementable successfully on only 3 of the data sets, so insights are limited. It is more computationally intensive than other techniques used, but not detrimentally so. Random Forests are usually a very robust algorithm for most problems. This is seen here as it gives good to excellent accuracy for the data sets it worked on. BH rate seems to be average to good, while the SC rate exhibited for random forest models seem excellent across all data sets.

Nearest neighbours give quite varying results. As with random forests, these were implementable on only 3 of the 6 data sets. Nearest neighbours are very fast, having quickest computation for nearly every data set. However, this may be due to a small number of comments in each of the data sets. There does not seem to be an appreciable difference between using 3 nearest and 5 nearest neighbours for any of the data sets. Accuracy is relatively poor compared to other methods. BH rate performance is highly inconsistent, with excellent performance in 2 data sets and the worst performance in the third data set. The same is true for SC rate as well.

Neural networks are highly intensive computationally, taking the maximum time to train models across every data set. With respect to performance though, they would be the closest to being called the best algorithm amongst the algorithms evaluated here. They have consistently excellent accuracy and SC rate. They do not perform very well with respect to BH rate though. It is to be noted that in the case of the Katy Perry data set, the improvement that neural networks have above the other algorithms is very significant. This is particularly relevant as the KP data set seems to be poor compared to the others with respect to the information contained. Hence, it seems that neural networks are capable of detecting patterns that other algorithms cannot here.

We see that for the combined data set, the algorithms that did work managed to scale up appropriately. This suggests that all spam, irrespective of the source video, has certain common features. We see that all algorithms perform on par with their performance on individual data sets across accuracy and BH rate. SC rate seems to suffer, but not noticeable so. We do see an increase in computational time across almost every technique as would be expected.

If one were to recommend an algorithm to use, it would be dependent on the priorities that one is looking at. Irrespective of priorities, I think that a good initial diagnostic would be to obtain a CART model. This would not be hard to compute and would give us an idea of important variables and factors, as well as a metric for performance. Following this, it is to be mentioned that one really cannot go wrong with most of the techniques listed. If one had very limited computing power at hand (meaning that computational time was priority), a good model for this data would be an LDA model. If one was very concerned that no legitimate comments should be removed and had good computing power at hand, a Boosted Tree model would be the best fit for this data. If one were very particular that all spam has to be captured by any means possible, a random forest or neural network model would work very well for the given data, provided that one has adequate computing power at hand.

VII. Caveats and Issues Encountered

The inferences noted above have some caveats to be addressed. Firstly, as is with any classification problem, there will always be a bias dependent on the manner of previous classification. Given that whether an item is spam or not would be slightly subjective, it is an important issue to be considered. Hence, it is imperative that prior to this classification, 'ground truth' as to whether a comment or not spam must be specified clearly. There should be a definite logic and set of rules determining as to whether a comment is spam or not.

Secondly, it was seen that there were other variables apart from just tect in our intial data set. Some of these were the username and date of posting. While the scope of this project was to evaluate algorithms only on the text mined data, useful insights may be gleaned from the given two factors. There may be certain times of the day when spam frequency is high, and certain 'repeat offenders'. Hence, these would be important factors to consider for any future analysis.

In addition, several of the algorithms used were not optimally tuned and default parameters were used. Tuning would undoubtedly give results closer to optimality as is evidenced in the original paper. Tuning of hyperparameters is usually done through grid searches. Functions for this are available on the caret package, but were not implemented due to a lack of knowledge. In addition, tuning can take significant amounts of computing power. I did submit a request to use the Penn State ACI cluster, but did not receive approval from them for this.

While the number of predictors is quite large, the number of comments themselves that are classified are quite low (About 350 per video). While this is enough to gain useful information, I still feel that this might be too small a sample to get optimal results. Assuming 25% testing samples, we often see only 90 test comments which I feel is a bit low. A larger sample of comments I think would offer better insights into relative algorithm performance, as well as to see if algorithms can scale up efficiently. Considering the volumes of data that would have to be handled in any commercial spam detection system, one would need an algorithm whose performance does not suffer much with an upscaling in input data.

Lastly, it was seen that cleaning and preprocessing the data itself seems to have much more importance as compared to different algorithms that could possibly be used. Data sets with better data had even supposedly 'inferior' algorithms perform very well on them. Preprocessing was not always accurate, as special characters and numbers were present even after they were supposedly removed. Hence, any analysis will be dependent on how well the source functions in R or Python are written. The fact that punctuation removal in the Shakira data set was not achievable meant that every algorithm could not be implemented. Therefore, better pre-processing and cleaning would greatly improve spam classification.

VIII. Future Work

In the original 2015 paper, one technique that performed very well was the Support Vector Machine. This was not implemented as I am not very familiar with most of its properties. In additions, SVMs perform exceptionally well only after extensive parameter tuning. Hence, seeing if SVM scales up appropriately would be of interest.

As mentioned, hyper parameter tuning was not performed much on the given techniques. Appropriate hyper parameter tuning would certainly yield better results and might even guide us to alternative techniques to be used when looking at different criteria.

All analysis was conducted on data sets that were reduced by 99% for sparsity. This did ensure that computation was not cumbersome. However, one might obtain even better results. In addition, almost all pre-processing steps that are performed (such as removal of stopWords) are performed not keeping junk classification in mind. These steps are usually performed with the aim of classifying input data into different categories, almost all of which are meaningful. One example is movie classification. Hence, one might find interesting results if analysis were performed without all the preprocessing steps. Two steps that come into mind would be the removal of numbers and the removal of stopwords, as I think these could possibly be spam indicators.

Finally, if one were to obtain a larger data set, we could possibly get better algorithm performance. However, the problem would be that the manual classification required to generate target response variables would be a very cumbersome task, limiting potential data set size.

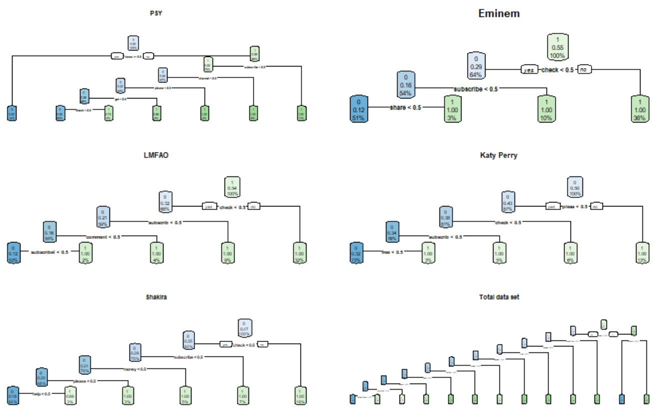
I would like to thank Dr. Soundar Kumara for his teaching throughout the course, as well as for guiding me through this project. It has certainly been a fruitful experience and I have gained a lot from it. In addition, I would also like to thank Dr. Le Bao from the Penn State department of Statistics who was my instructor for my STAT 557 (Data Mining Course) as well as Ben Lee, Ana Kenney and Rajat Sanklecha (Graduate students from the department of Statistics & IE, project partners for STAT 557 projects). I learned valuable skills thanks to everyone mentioned here.

IX. References

1) A Business Intelligence System (HP Luhn – October 1958, IBM Journal of Research and Development)
2) A Brief Survey of Text Mining (Andreas Hotho, Andreas Nurnberger & Gerhard Paas – 2005)
3) An algorithm for suffix stripping (MF Porter - July 1980, Program)
4) TubeSpam: Comment Spam Filtering on YouTube (TA Almeida, JV Lochter, TC Alberto - December 2015, Proceedings of the 14th IEEE International Conference on Machine Learning and Applications)
5) Text Mining Infrastructure in R (Ingo Feinerer, Kurt Hornik, David Herner – March 2008, Journal of Statistical Software)
6) A neural network based approach to automated e-mail classification (J Clark, I Koprinska, J Poon - October 2003, Proceedings of the IEEE International Conference on Web Intelligence)
7) Learning Representations by back-propagating errors (DE Rumelhart, GE Hinton, RJ Williams - October 1986, Nature)
8) Introduction To Statistical Learning - 7th Edition (G James, J Witten, T Hastie, R Tibshirani)

X. Appendix

Figure 3: CART Trees constructed from different data sets



Code

```
##Load required packages ##
install.packages("SnowballC")
install.packages("caTools")
install.packages("ada")
library("ada")
library("RSNNS")
library("xgboost")
library("tm")
library("wordcloud")
library("SnowballC")
library(caTools)
library(ROCR)
library(caret)
library(ggplot2)
library(MASS)
library(rpart)
library(rpart.plot)
library(neuralnet)
library(igraph)
library(Rgraphviz)
library(randomForest)
library(gridExtra)
library(markdown)

## Read data frames ##
setwd("C:/Users/vigne/Desktop/IE␣Stuff/IE␣582␣-␣Engineering␣Analytics/Project/Dataset")
PSY=read.csv("Youtube␣01-comments␣Psy.csv",stringsAsFactors=FALSE)
KP=read.csv("Youtube␣04-comments␣KatyPerry.csv",stringsAsFactors=FALSE)
LMFAO=read.csv("Youtube␣07-comments␣LMFAO.csv",stringsAsFactors=FALSE)
Eminem=read.csv("Youtube␣08-comments␣Eminem.csv",stringsAsFactors=FALSE)
Shakira=read.csv("Youtube␣09-comments␣Shakira.csv",stringsAsFactors=FALSE)
Total=rbind.data.frame(PSY,KP,LMFAO,PSY, Shakira) ##Incorporate total data frame
par(mfrow=c(3,2))

View(PSY)

## Preprocess PSY data set by extracting content, making lowercase, removing punctuation,
corpus1=VCorpus(VectorSource(PSY$CONTENT))
```

```
corpus1[[1]]$content
corpus1=tm_map(corpus1,content_transformer(tolower
corpus1=tm_map(corpus1,removePunctuation)
corpus1=tm_map(corpus1,removeNumbers)
corpus1=tm_map(corpus1,removeWords,stopwords("engl
dtm1=DocumentTermMatrix(corpus1)
Sdtm1=removeSparseTerms(dtm1,0.99) ##Remove 99% sp

##Create WordCloud from PSY data set and histogram
WPSY=as.data.frame(as.matrix(Sdtm1))
wordcloud(colnames(WPSY),colSums(WPSY),random.orde
PH=data.frame(Terms=colnames(WPSY),Frequency=colSur
PH$Terms=factor(PH$Terms, levels = PH$Terms[order(PI
PH=PH[order(PH$Frequency, decreasing = T),]
P1=ggplot(PH, aes(y=PH$Frequency, x=PH$Terms))+geom_

## Preprocess Katy Perry data set by extracting co
corpus2=VCorpus(VectorSource(KP$CONTENT))
corpus2[[2]]$content
corpus2=tm_map(corpus2,content_transformer(tolower
corpus2=tm_map(corpus2,removePunctuation)
corpus2=tm_map(corpus2,removeNumbers)
corpus2=tm_map(corpus2,stemDocument)
corpus2=tm_map(corpus2,removeWords,stopwords("engl
dtm2=DocumentTermMatrix(corpus2)
Sdtm2=removeSparseTerms(dtm2,0.99) ##Remove 99% sp

##Create WordCloud from Katy Perry data
set
WKP=as.data.frame(as.matrix(Sdtm2))
wordcloud(colnames(WKP),colSums(WKP),random.order =
KH=data.frame(Terms=colnames(WKP),Frequency=colSum
KH$Terms=factor(KH$Terms, levels = KH$Terms[order(K
KH=KH[order(KH$Frequency, decreasing = T),]
P2=ggplot(KH, aes(y=KH$Frequency, x=KH$Terms))+geom_

## Preprocess Eminem data set by extracting conten
corpus3=VCorpus(VectorSource(Eminem$CONTENT))
corpus3[[3]]$content
corpus3=tm_map(corpus3,content_transformer(tolower
corpus3=tm_map(corpus3,removePunctuation)
corpus3=tm_map(corpus3,removeNumbers)
corpus3=tm_map(corpus3,removeWords,stopwords("engl
dtm3=DocumentTermMatrix(corpus3)
Sdtm3=removeSparseTerms(dtm3,0.99) ##Remove 99% sp

##Create WordCloud from Eminem data set
WEm=as.data.frame(as.matrix(Sdtm3))
wordcloud(colnames(WEm),colSums(WEm),random.order =
EH=data.frame(Terms=colnames(WEm),Frequency=colSum
EH$Terms=factor(EH$Terms, levels = EH$Terms[order(E
EH=EH[order(EH$Frequency, decreasing = T),]
P3=ggplot(EH, aes(y=EH$Frequency, x=EH$Terms))+geom_

## Preprocess LMFAO data set by extracting content
corpus4=VCorpus(VectorSource(LMFAO$CONTENT))
corpus4[[4]]$content
corpus4=tm_map(corpus4,content_transformer(tolower
corpus4=tm_map(corpus4,removePunctuation)
corpus4=tm_map(corpus4,removeNumbers)
corpus4=tm_map(corpus4,stemDocument)
corpus4=tm_map(corpus4,removeWords,stopwords("english"))
dtm4=DocumentTermMatrix(corpus4)
Sdtm4=removeSparseTerms(dtm4,0.99) ##Remove 99% sparse terms due to inflated sparsity

##Create WordCloud from LMFAO data  set
WLM=as.data.frame(as.matrix(Sdtm4))
wordcloud(colnames(WLM),colSums(WLM),random.order=FALSE, min.freq = 5,rot.per = .1)
LH=data.frame(Terms=colnames(WEm),Frequency=colSums(WEm))
LH$Terms=factor(LH$Terms, levels = LH$Terms[order(LH$Frequency, decreasing = T)])
LH=LH[order(LH$Frequency, decreasing = T),]
P4=ggplot(LH, aes(y=LH$Frequency,x=LH$Terms))+geom_histogram(stat="identity",fill="light␣blue")+

## Preprocess Shakira data set by extracting content, making all lower case, removing punctuatic
corpus5=VCorpus(VectorSource(Shakira$CONTENT))
corpus5[[5]]$content
corpus5=tm_map(corpus5,removePunctuation)
corpus5=tm_map(corpus5,content_transformer(tolower))
corpus5=tm_map(corpus5,removeNumbers)
corpus5=tm_map(corpus5,removeWords,stopwords("english"))
corpus5=VCorpus(VectorSource(corpus5))
dtm5=DocumentTermMatrix(corpus5)
Sdtm5=removeSparseTerms(dtm5,0.99) ##Remove 99% sparse terms due to inflated sparsity

##Create WordCloud from Shakira data set
WSha=as.data.frame(as.matrix(Sdtm5))
wordcloud(colnames(WSha),colSums(WSha),random.order=FALSE, min.freq = 5,rot.per = .1)
SH=data.frame(Terms=colnames(WSha),Frequency=colSums(WSha))
SH$Terms=factor(SH$Terms, levels = SH$Terms[order(SH$Frequency, decreasing = T)])
SH=SH[order(SH$Frequency, decreasing = T),]
P5=ggplot(SH, aes(y=SH$Frequency,x=SH$Terms))+geom_histogram(stat="identity",fill="light␣blue")+

## Preprocess Total data set by extracting content, making all lower case, removing punctuation"
corpusT=VCorpus(VectorSource(Total$CONTENT))
corpusT[[5]]$content
corpusT=tm_map(corpusT,content_transformer(tolower))
corpusT=tm_map(corpusT,removeNumbers)
corpusT=tm_map(corpusT,removeWords,stopwords("english"))
dtmT=DocumentTermMatrix(corpusT)
SdtmT=removeSparseTerms(dtmT,0.99) ##Remove 99% sparse terms due to inflated sparsity

##Create WordCloud from Total data set
WT=as.data.frame(as.matrix(SdtmT))
wordcloud(colnames(WT),colSums(WT),random.order=FALSE, min.freq = 5,rot.per = .1)
TH=data.frame(Terms=colnames(WT),Frequency=colSums(WT))
TH$Terms=factor(TH$Terms, levels = TH$Terms[order(TH$Frequency, decreasing = T)])
TH=TH[order(TH$Frequency, decreasing = T),]
P6=ggplot(TH, aes(y=TH$Frequency,x=TH$Terms))+geom_histogram(stat="identity",fill="light␣blue")+
grid.arrange(P1,P2,P3,P4,P5,P6,nrow=3,ncol=2)

##Attach response variables to respective data frames and convert to Factor var
WPSY$Class=as.factor(PSY$CLASS)
WEm$Class = as.factor(Eminem$CLASS)
WSha$Class= as.factor(Shakira$CLASS)
WLM$Class= as.factor(LMFAO$CLASS)
WKP$Class=as.factor(KP$CLASS)
```

```
WT$Class=as.factor(Total$CLASS)

##Split datasets into 75:25 Training:Testing sets
set.seed(2017)
PSplit=sample.split(WPSY$Class,0.75) ##PSY
PTrain=subset(WPSY, PSplit==T)
PTest=subset(WPSY, PSplit==F)

set.seed(2017)
ESplit=sample.split(WEm$Class,0.75) ##Eminem
ETrain=subset(WEm, ESplit==T)
ETest=subset(WEm, ESplit==F)

set.seed(2017)
SSplit=sample.split(WSha$Class,0.75) ##Shakira
STrain=subset(WSha, SSplit==T)
STest=subset(WSha, SSplit==F)

set.seed(2017)
LSplit=sample.split(WLM$Class,0.75) ##LMFAO
LTrain=subset(WLM, LSplit==T)
LTest=subset(WLM, LSplit==F)

set.seed(2017)
KSplit=sample.split(WKP$Class,0.75) ##Katy Perry
KTrain=subset(WKP, KSplit==T)
KTest=subset(WKP, KSplit==F)

set.seed(2017)
TSplit=sample.split(WT$Class,0.75) ##Total
TTrain=subset(WT, TSplit==T)
TTest=subset(WT, TSplit==F)

##Logistic regression models for the 6 datasets –
PLog=glm(formula=Class~., data = PTrain, family = "
system.time(glm(formula=Class~., data = PTrain,fam
PLogPred=predict(PLog,newdata=PTest)
table(PTest$Class,PLogPred>0.5) ##PSY

ELog=glm(formula=Class~., data = ETrain,family = "
system.time(glm(formula=Class~., data = ETrain,fam
ELogPred=predict(ELog,newdata=ETest)
table(ETest$Class,ELogPred>0.5) ##Eminem

SLog=glm(formula=Class~., data = STrain,family = "
system.time(glm(formula=Class~., data = STrain,fam
SLogPred=predict(SLog,newdata=STest)
table(STest$Class,SLogPred>0.5) ##Shakira

LLog=glm(formula=Class~., data = LTrain,family = "
system.time(glm(formula=Class~., data = LTrain,fam
LLogPred=predict(LLog,newdata=LTest)
table(LTest$Class,LLogPred>0.5) ##LMFAO

KLog=glm(formula=Class~., data = KTrain,family = "
system.time(glm(formula=Class~., data = KTrain,fam
KLogPred=predict(KLog,newdata=KTest)
table(KTest$Class,KLogPred>0.5) ##Katy Perry
```

```
TLog=glm(formula=Class~., data = TTrain, family = "binomial")
system.time(glm(formula=Class~., data = TTrain, family = "binomial"))
TLogPred=predict(TLog, newdata=TTest)
table(TTest$Class, TLogPred>0.5) ##Total dataset

TLogPredR=ROCR::prediction(TLogPred, TTest$Class)
TLogPerf=performance(TLogPredR, "tpr", "fpr")
plot(TLogPerf, main = "Total␣Data␣Set,␣Logistic␣Regression␣ROC␣Plot")

##LDA models for the 6 datasets – Build model, fit using data, measure system time elapsed,
PLDA=lda(Class~., data = PTrain)
system.time(lda(Class~., data = PTrain))
PLDAPred=predict(PLDA, newdata=PTest)
table(PTest$Class, PLDAPred$class) ##PSY

ELDA=lda(Class~., data = ETrain)
system.time(lda(Class~., data = ETrain))
ELDAPred=predict(ELDA, newdata=ETest)
table(ETest$Class, ELDAPred$class) ##Eminem

SLDA=lda(Class~., data = STrain)
system.time(lda(Class~., data = STrain))
SLDAPred=predict(SLDA, newdata=STest)
table(STest$Class, SLDAPred$class) ##Shakira

LLDA=lda(Class~., data = LTrain)
system.time(lda(Class~., data = LTrain))
LLDAPred=predict(LLDA, newdata=LTest)
table(LTest$Class, LLDAPred$class) ##LMFAO

KLDA=lda(Class~., data = KTrain)
system.time(lda(Class~., data = KTrain))
KLDAPred=predict(KLDA, newdata=KTest)
table(KTest$Class, KLDAPred$class) ##Katy Perry

TLDA=lda(Class~., data = TTrain)
system.time(lda(Class~., data = TTrain))
TLDAPred=predict(TLDA, newdata=TTest)
table(TTest$Class, TLDAPred$class) ##Total DataSet

TLDAPredR=ROCR::prediction(TLDAPred$class, TTest$Class)
TLDAPerf=performance(TLDAPredR, "tpr", "fpr")
plot(TLDAPerf, main = "Total␣Data␣Set,␣LDA␣ROC␣Plot")

##QDA – Rank deficiency, likely indication that not enough data available

##CART models for the 6 datasets – Build model, fit using data, measure system time elapsed,
PTree=rpart(Class~., data=PTrain, method = "class")
system.time(rpart(Class~., data=PTrain, method = "class"))
rpart.plot(PTree, main = "PSY")
PTreePred=predict(PTree, newdata = PTest)
table(PTest$Class, PTreePred[,2]>0.5) ##PSY

ETree=rpart(Class~., data=ETrain, method = "class")
system.time(rpart(Class~., data=ETrain, method = "class"))
rpart.plot(ETree, main="Eminem")
```

```
ETreePred=predict(ETree, newdata = ETest)
table(ETest$Class, ETreePred[,2]>0.5) ##Eminem

LTree=rpart(Class~., data=LTrain, method = "class")
system.time(rpart(Class~., data=LTrain, method = "c
rpart.plot(LTree, main="LMFAO")
LTreePred=predict(LTree, newdata = LTest)
table(LTest$Class, LTreePred[,2]>0.5) ##LMFAO

KTree=rpart(Class~., data=KTrain, method = "class")
system.time(rpart(Class~., data=KTrain, method = "c
rpart.plot(KTree, main="Katy␣Perry")
KTreePred=predict(KTree, newdata = KTest)
table(KTest$Class, KTreePred[,2]>0.5) ##Katy Perry

STree=rpart(Class~., data=STrain, method = "class")
system.time(rpart(Class~., data=STrain, method = "c
rpart.plot(STree, main="Shakira")
STreePred=predict(STree, newdata = STest)
table(STest$Class, STreePred[,2]>0.5) ##Shakira

TTree=rpart(Class~., data=TTrain, method = "class")
system.time(rpart(Class~., data=TTrain, method = "c
rpart.plot(TTree, main="Total␣data␣set")
TTreePred=predict(TTree, newdata = TTest)
table(TTest$Class, TTreePred[,2]>0.5) ##Total data

TTreePredR=ROCR::prediction(TTreePred[,2], TTest$Cl
TTreePerf=performance(TTreePredR, "tpr", "fpr")
plot(TTreePerf, main = "Total␣Data␣Set,␣CART␣ROC␣Pl

##Boosted Tree models for the 6 datasets (Adaboost
PAda=ada(Class~., data=PTrain)
system.time(ada(Class~., data=PTrain))
PAdaPred=predict(PAda, newdata = PTest)
table(PTest$Class, PAdaPred)

EAda=ada(Class~., data=ETrain)
system.time(ada(Class~., data=ETrain))
EAdaPred=predict(EAda, newdata = ETest)
table(ETest$Class, EAdaPred)

LAda=ada(Class~., data=LTrain)
system.time(ada(Class~., data=LTrain))
LAdaPred=predict(LAda, newdata = LTest)
table(LTest$Class, LAdaPred)

KAda=ada(Class~., data=KTrain)
system.time(ada(Class~., data=KTrain))
KAdaPred=predict(KAda, newdata = KTest)
table(KTest$Class, KAdaPred)

SAda=ada(Class~., data=STrain)
system.time(ada(Class~., data=STrain))
SAdaPred=predict(SAda, newdata = STest)
table(STest$Class, SAdaPred)

TAda=ada(Class~., data=TTrain)
```

```
system.time(ada(Class~.,data=TTrain))
TAdaPred=predict(TAda,newdata = TTest)
table(TTest$Class,TAdaPred)

##Random Forest models for the 6 datasets – Build model, measure system time
set.seed(2017)
PRF=randomForest(Class~., data=PTrain)
system.time(randomForest(Class~., data=PTrain))
PRFPred=predict(PRF,newdata = PTest)
table(PTest$Class,PRFPred) ##PSY

set.seed(2017)
ERF=randomForest(Class~.,data=ETrain)
system.time(randomForest(Class~., data=ETrain))
ERFPred=predict(ERF,newdata = ETest)
table(ETest$Class,ERFPred) ##Eminem

set.seed(2017) ##Resolve
SRF=randomForest(Class~., data=STrain)
SRFPred=predict(SRF,newdata = STest)
table(SRFPred,STest$Class) ##Shakira

set.seed(2017)
LRF=randomForest(Class~., data=LTrain)
system.time(randomForest(Class~., data=LTrain))
LRFPred=predict(LRF,newdata = LTest)
table(LTest$Class,LRFPred) ##LMFAO


set.seed(2017)  ##Resolve
KRF=randomForest(Class~., data=KTrain)
KRFPred=predict(KRF,newdata = KTest)
table(KRFPred,KTest$Class)

set.seed(2017) ##Resolve
TRF=randomForest(Class~., data=TTrain)
TRFPred=predict(TRF,newdata = TTest)
table(TRFPred,TTest$Class)

##5–Nearest Neighbour models for the 6 datasets – Build model, measure system time
P5NN=knn3(Class~., data=PTrain, k=5)
system.time(knn3(Class~., data=PTrain, k=5))
P5NNPred=predict(P5NN,newdata = PTest)
table(PTest$Class,P5NNPred[,2]>0.5)##PSY

E5NN=knn3(Class~., data=ETrain, k=5)
system.time(knn3(Class~., data=ETrain, k=5))
E5NNPred=predict(E5NN,newdata = ETest)
table(ETest$Class,E5NNPred[,2]>0.5)##Eminem

K5NN=knn3(Class~., data=KTrain, k=5) ##Resolve
system.time(knn3(Class~., data=KTrain, k=5))
K5NNPred=predict(K5NN,newdata = KTest)
table(K5NNPred[,2]>0.5,KTest$Class)

L5NN=knn3(Class~., data=LTrain, k=5)
system.time(knn3(Class~., data=LTrain, k=5))
L5NNPred=predict(L5NN,newdata = LTest)
```

```
table(LTest$Class,L5NNPred[,2]>0.5)##LMFAO

S5NN=knn3(Class~., data=STrain, k=5) ##Resolve
system.time(knn3(Class~., data=STrain, k=5))
S5NNPred=predict(S5NN,newdata = STest)
table(S5NNPred[,2]>0.5,STest$Class)##Shakira

T5NN=knn3(Class~., data=TTrain, k=5) ##Resolve
system.time(knn3(Class~., data=TTrain, k=5))
T5NNPred=predict(T5NN,newdata = TTest)
table(T5NNPred[,2]>0.5,TTest$Class)##Total Data Se

##3–Nearest Neighbour models for the 6 datasets –
P3NN=knn3(Class~., data=PTrain, k=3)
system.time(knn3(Class~., data=PTrain, k=3))
P3NNPred=predict(P3NN,newdata = PTest)
table(PTest$Class,P3NNPred[,2]>0.5)##PSY

E3NN=knn3(Class~., data=ETrain, k=3)
system.time(knn3(Class~., data=ETrain, k=3))
E3NNPred=predict(E3NN,newdata = ETest)
table(ETest$Class,E3NNPred[,2]>0.5)##Eminem

K3NN=knn3(Class~., data=KTrain, k=3) ##Resolve
system.time(knn3(Class~., data=KTrain, k=3))
K3NNPred=predict(K3NN,newdata = KTest)
table(K3NNPred[,2]>0.5,KTest$Class)

L3NN=knn3(Class~., data=LTrain, k=3)
system.time(knn3(Class~., data=LTrain, k=3))
L3NNPred=predict(L3NN,newdata = LTest)
table(LTest$Class,L3NNPred[,2]>0.5)##LMFAO

S3NN=knn3(Class~., data=STrain, k=3) ##Resolve
system.time(knn3(Class~., data=STrain, k=3))
S3NNPred=predict(S3NN,newdata = STest)
table(S3NNPred[,2]>0.5,STest$Class)##Shakira

T3NN=knn3(Class~., data=TTrain, k=3) ##Resolve
system.time(knn3(Class~., data=TTrain, k=3))
T3NNPred=predict(T3NN,newdata = TTest)
table(T3NNPred[,2]>0.3,TTest$Class)##Total Data Se

##Neural network models for the 6 datasets – Build

WPSY2=WPSY[sample(1:nrow(WPSY),length(1:nrow(WPSY)
WPSYValues=WPSY2[,1:107]
WPSYTargets=decodeClassLabels(WPSY2[,108])
WPSY2=splitForTrainingAndTest(WPSYValues,WPSYTarge
PNNModel=mlp(WPSY2$inputsTrain,WPSY2$targetsTrain,
system.time(mlp(WPSY2$inputsTrain,WPSY2$targetsTrai
PNNPred=predict(PNNModel,WPSY2$inputsTest)
PTarget=WPSY2$targetsTest[,2]
table(PTarget,PNNPred[,2]>0.5) ##PSY

WLM2=WLM[sample(1:nrow(WLM),length(1:nrow(WLM))),]
WLMValues=WLM2[,1:67]
WLMTargets=decodeClassLabels(WLM2[,68])
```

```
WLM2=splitForTrainingAndTest(WLMValues,WLMTargets,ratio = 0.25)
LNNModel=mlp(WLM2$inputsTrain,WLM2$targetsTrain,size=c(40,30,2),learnFuncParams = 0.3,maxit =
system.time(mlp(WLM2$inputsTrain,WLM2$targetsTrain,size=c(40,30,2),learnFuncParams = 0.3,maxit
LNNPred=predict(LNNModel,WLM2$inputsTest)
LTarget=WLM2$targetsTest[,2]
table(LTarget,LNNPred[,2]>0.5) ##LMFAO

WSha2=WSha[sample(1:nrow(WSha),length(1:nrow(WSha))),]
WShaValues=WSha2[,1:199]
WShaTargets=decodeClassLabels(WSha2[,200])
WSha2=splitForTrainingAndTest(WShaValues,WShaTargets,ratio = 0.25)
SNNModel=mlp(WSha2$inputsTrain,WSha2$targetsTrain,size=c(80,50,2),learnFuncParams = 0.3,maxit =
system.time(mlp(WSha2$inputsTrain,WSha2$targetsTrain,size=c(80,50,2),learnFuncParams = 0.3,max
SNNPred=predict(SNNModel,WSha2$inputsTest)
STarget=WSha2$targetsTest[,2]
table(STarget,SNNPred[,2]>0.5) ##Shakira

WKP2=WKP[sample(1:nrow(WKP),length(1:nrow(WKP))),]
WKPValues=WKP2[,1:149]
WKPTargets=decodeClassLabels(WKP2[,150])
WKP2=splitForTrainingAndTest(WKPValues,WKPTargets,ratio = 0.25)
KNNModel=mlp(WKP2$inputsTrain,WKP2$targetsTrain,size=c(100,50,2),learnFuncParams = 0.3,maxit =
system.time(mlp(WKP2$inputsTrain,WKP2$targetsTrain,size=c(100,50,2),learnFuncParams = 0.3,maxi
KNNPred=predict(KNNModel,WKP2$inputsTest)
KTarget=WKP2$targetsTest[,2]
table(KTarget,KNNPred[,2]>0.5) ##Katy Perry

WEm2=WEm[sample(1:nrow(WEm),length(1:nrow(WEm))),]
WEmValues=WEm2[,1:178]
WEmTargets=decodeClassLabels(WEm2[,179])
WEm2=splitForTrainingAndTest(WEmValues,WEmTargets,ratio = 0.25)
ENNModel=mlp(WEm2$inputsTrain,WEm2$targetsTrain,size=c(150,100,2),learnFuncParams = 0.3,maxit =
system.time(mlp(WEm2$inputsTrain,WEm2$targetsTrain,size=c(150,100,2),learnFuncParams = 0.3,max
ENNPred=predict(ENNModel,WEm2$inputsTest)
ETarget=WEm2$targetsTest[,2]
table(ETarget,ENNPred[,2]>0.5) ##Eminem

WT2=WT[sample(1:nrow(WT),length(1:nrow(WT))),]
WTValues=WT2[,1:91]
WTTargets=decodeClassLabels(WT2[,92])
WT2=splitForTrainingAndTest(WTValues,WTTargets,ratio = 0.25)
TNNModel=mlp(WT2$inputsTrain,WT2$targetsTrain,size=c(60,40,2),learnFuncParams = 0.3,maxit = 10
system.time(mlp(WT2$inputsTrain,WT2$targetsTrain,size=c(60,40,2),learnFuncParams = 0.3,maxit =
TNNPred=predict(TNNModel,WT2$inputsTest)
TTarget=WT2$targetsTest[,2]
table(TTarget,TNNPred[,2]>0.5) ##Total Data Set
```