SMS Spam Message Detection

Daniel McIntyre

CS 7720

Fall 2015

Table of Contents

[Introduction 3](#_Toc438074930)

[About the Data 4](#_Toc438074931)

[Data Handling & Processing 5](#_Toc438074932)

[Input & Feature Construction 5](#_Toc438074933)

[Feature Selection 5](#_Toc438074934)

[Classification 6](#_Toc438074935)

[Output & Visualization 6](#_Toc438074936)

[Notable Results 7](#_Toc438074937)

[Best J48 Results 7](#_Toc438074938)

[Best Naïve Bayes Results 8](#_Toc438074939)

[Best SMO Results 9](#_Toc438074940)

[Discussion 10](#_Toc438074941)

[Conclusion 11](#_Toc438074942)

[Additional Results 12](#_Toc438074943)

[References & Additional Resources 13](#_Toc438074944)

# Introduction

Communication is a core component in the day-to-day lives of people. In this day and age, technology plays a crucial role in communications. Email, Skype messaging, and SMS (short messaging service) are just a few types of messaging communication that are used quite regularly. Of course, with various types of messaging, there is bound to be spam (junk messages, usually from advertisers or unknown senders) coming somebody’s way every once and awhile. A lot of research has been done to combat spam emails, but spam message identification in the area of SMS has been focused on a bit less.

Spam SMS messages can be identified by using supervised learning, such as classification. After precise manipulation of SMS message data followed by training and testing on various classifiers, the results of this project prove this hypothesis true.

# About the Data

The SMS message data used in this project is from the SMS Spam Collection Data Set from the UCI Machine Learning Repository [1]. The data set contains a total of 5574 messages, with a total of 747 being spam messages. The other class of objects is considered as “ham”, which represents the positive class of non-spam messages. As the data is entirely SMS messages, the data is all text based. As such, text-based data processing and handling must take place to be able to properly classify the data. The data has a total of 6829 attributes, plus the separately annotated class type, making this a 6830-dimensional data set.

# Data Handling & Processing

There are four steps to the flow of the data in this analysis: input and feature construction, feature selection, classification, and lastly output and visualization. A Java program (SMSSpam.jar) was written to encompass all four of these tasks, which can be run to completion in under 90 seconds, along with some user input. A couple of python scripts were written and used to make the data easier to process for Java. Each of the four steps are explained in detail in the respective sections.

## Input & Feature Construction

The original SMS Spam data set came as one large txt file complete with all 5574 instances, one per line. From this original set, the data was randomly split into two files, with roughly the same amount of instances. One of the split files thus became the training data set, which contains 2822 instances. The other text file from the split became the testing data set, containing 2752 instances.

With a separate training and testing data set, the SMSSpam program can be run. The first step it takes is to construct features from the input text files. A training feature table is constructed from the training data, and a testing feature table from the testing data. Two text feature models were used for analysis in this project. The first method is simply using a raw frequency count for the attributes. In this model, each individual word in the entire document is mapped as an attribute. Each instance (SMS message) is then represented as the number of times each attribute appears in the particular instance.

The second feature model used is the TF-IDF model, a well-known model for representing text data in various fields [2]. Essentially, the TF-IDF model takes into account not only the frequency of specific word in an instance, but also the importance of that word across the entirety of the data set, which is considered as inverse document frequency. Each word in the entire document is still mapped as an attribute, but the values in the instances are represented by a number which is formulaically devised from the term frequency and the log-damped inverse document frequency.

Since the English vocabulary is so large, with SMS messages being quite short, these feature tables become a very sparse data set. It should also be noted that SMS messages contain quite a bit of slang, cultural differences with certain words, and misspellings. This is the most difficult hurdle when it comes to feature construction. In both methods, the Snowball Stemmer is used to reduce every word possible to a common word stem [3]. This isn’t a perfect solution, but it does improve matters. After construction, the data is saved into ARFF format, so it can undergo the feature selection and classification process with Weka.

## Feature Selection

After feature construction occurs, the next step is to select the most important features, so that the classification process runs as clean and efficiently as possible. The feature selection process serves to remove any sort of outliers, noisy, and other poor quality attributes. Attributes that are not inherently “good” may also be removed in the feature selection process, because they may not positively contribute to classification process.

The first step taken in feature selection pipeline is a simple one. As previously indicated, there are over 6000 attributes that are inherently constructed from each of the training and testing data. This is far too much to efficiently handle, so the top 3000 attributes in terms of frequency are chosen right off the bat. This is done for the training data, and the testing data follows suite. The rest not considered in the top 3000 are pruned from the set and are discarded. The ARFF files are subsequently updated to represent this pruning.

The next step in the feature selection process is entirely optional, and serves as another testable variable in the project to see how it can affect the results. The feature selection algorithm used here is known as Information Gain. It is one of the many feature selection algorithms that can be found within the Weka package. The information gain algorithm is a good general purpose selection algorithm for classification cases. Essentially, in information theory, information gain can be used to determine which data will contribute the most information to the data set [4]. In this part of the process, 1 to 3000 features can be selected (3000 meaning no information gain is run, as 3000 are left after the prune). For example, if 500 features are to be selected out of the remaining 3000, these 500 features will be the top 500 attributes that contribute information based on all their values. Using this selection method can be very beneficial, but can also be a double edged sword. Selecting too many features is considered as “overfitting” the data. In the case of overfitting, the data set is represented too precisely, and the classifier in the testing phase has a difficult time of making generalizations that it needs to generate models.

## Classification

The third phase in this project is the actual classification phase. Three different classifiers were experimented with and used to classify the data. All three classifiers are part of the Weka package, and are available through the Weka Java API. The first classifier used is J48, a version of the C4.5 decision tree algorithm for Java [5]. Another classifier used is the classic statistical classifier, Naïve Bayes [6]. SMO (Sequential Minimal Optimization), an optimization on the SVM (support vector machine) classifier, is the last classification option. One of these three classifiers can be chosen to classify the data after it has gone through the feature selection process. The classification process consists of two steps, the training and testing phases. The classifier learns via the training data, and then take the models it has learned from the training data and apply them to the testing data.

## Output & Visualization

After classification, statistics are saved to a file for viewing at leisure. There is quite the variety of statistics provided from the classification process, but the most important are the precision, recall, and f-measure, as well as the confusion matrix showing the number of correctly and incorrectly classified instances [7]. Precision represents the accuracy of the classifier returning relevant results versus irrelevant results. Recall represents the ability of the classifier to return relevant results from the total pool of relevant results. The F-measure is the harmonic mean between the precision and recall values. The larger the values, the better.

Once output is saved to the file, a GUI window pops up showing the basic ROC curve for the data. The ROC curve is a great way to visualize how a classifier performs [8]. In short, the larger the area under the “curve”, the better the accuracy of the classifier. The text output from the classifier also shows the area of the ROC curve in numerical format.

# Notable Results

## Best J48 Results

J48 Classification  
TF-IDF Construction  
3000 Features

Correctly Classified Instances 2610 94.8401 %

Incorrectly Classified Instances 142 5.1599 %

Kappa statistic 0.7488

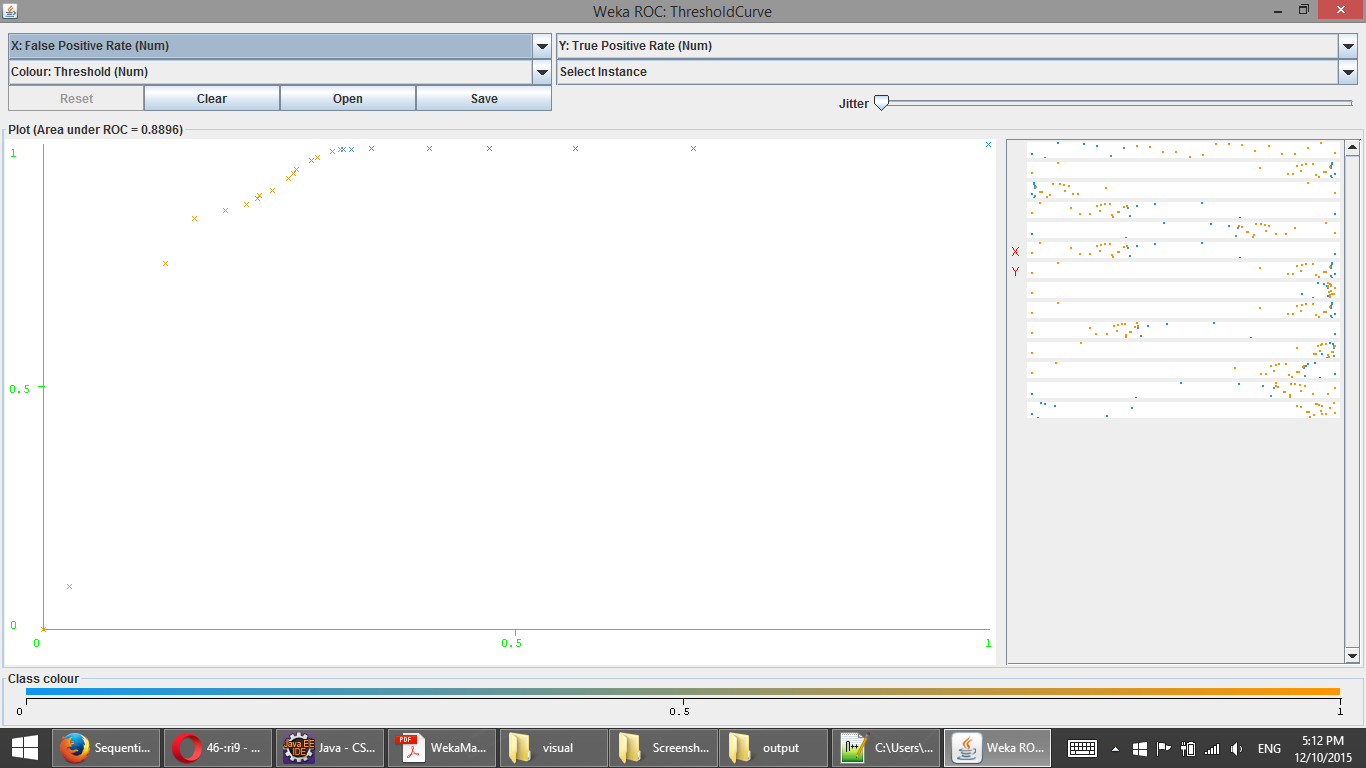
Mean absolute error 0.071

Root mean squared error 0.221

Total Number of Instances 2752

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Detailed Accuracy By Class | | | | | | | | |
| Class | TP Rate | FP Rate | Precision | Recall | F-Measure | MCC | ROC Area | PRC Area |
| Ham | 0.988 | 0.315 | 0.954 | 0.988 | 0.971 | 0.758 | 0.890 | 0.970 |
| Spam | 0.685 | 0.012 | 0.899 | 0.685 | 0.777 | 0.758 | 0.890 | 0.717 |
| Weighted Avg. | 0.948 | 0.275 | 0.947 | 0.948 | 0.945 | 0.758 | 0.890 | 0.936 |

|  |  |  |
| --- | --- | --- |
| Confusion Matrix | | |
| Classified As | A | B |
| a = Ham | 2362 | 28 |
| b = Spam | 114 | 248 |



## Best Naïve Bayes Results

Naïve Bayes Classification  
Raw Count Feature Construction  
100 Features

Correctly Classified Instances 2622 95.2762 %

Incorrectly Classified Instances 130 4.7238 %

Kappa statistic 0.7913

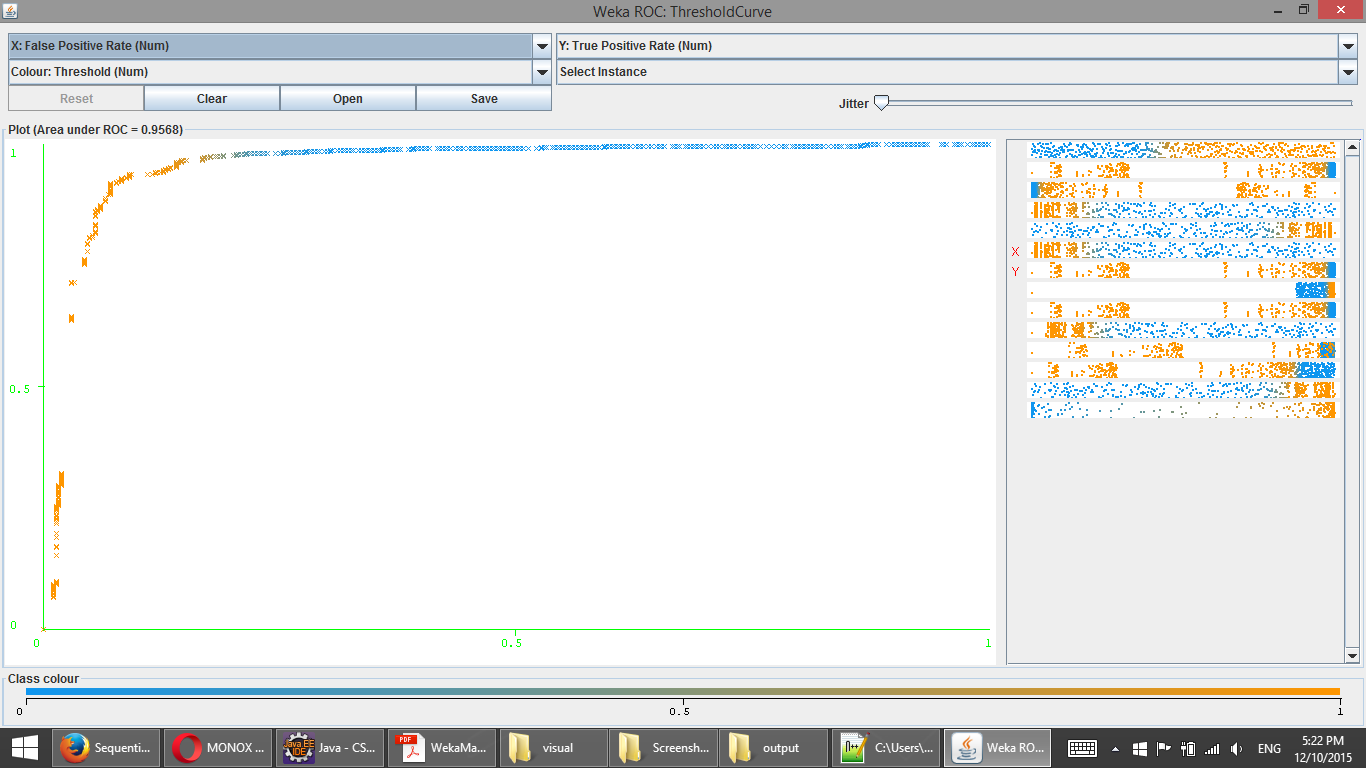
Mean absolute error 0.0509

Root mean squared error 0.2036

Total Number of Instances 2752

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Detailed Accuracy By Class | | | | | | | | |
| Class | TP Rate | FP Rate | Precision | Recall | F-Measure | MCC | ROC Area | PRC Area |
| Ham | 0.974 | 0.191 | 0.971 | 0.974 | 0.973 | 0.791 | 0.957 | 0.988 |
| Spam | 0.809 | 0.026 | 0.828 | 0.809 | 0.818 | 0.791 | 0.957 | 0.825 |
| Weighted Avg. | 0.953 | 0.169 | 0.952 | 0.953 | 0.953 | 0.791 | 0.957 | 0.966 |

|  |  |  |
| --- | --- | --- |
| Confusion Matrix | | |
| Classified As | A | B |
| a = Ham | 2329 | 61 |
| b = Spam | 69 | 293 |



## Best SMO Results

SMO Classification  
TF-IDF Feature Construction  
3000 Features

Correctly Classified Instances 2697 98.0015 %

Incorrectly Classified Instances 55 1.9985 %

Kappa statistic 0.9097

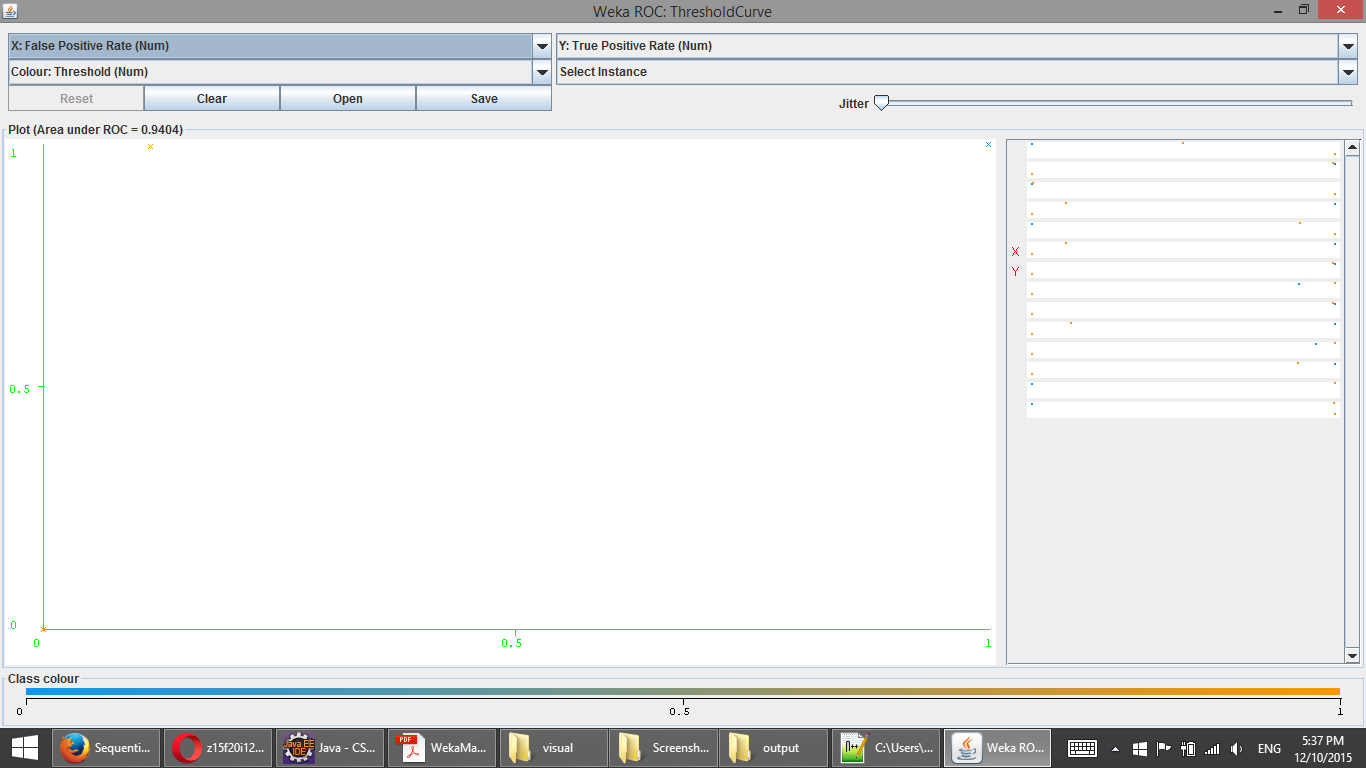
Mean absolute error 0.02

Root mean squared error 0.1414

Total Number of Instances 2752

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Detailed Accuracy By Class | | | | | | | | |
| Class | TP Rate | FP Rate | Precision | Recall | F-Measure | MCC | ROC Area | PRC Area |
| Ham | 0.994 | 0.113 | 0.983 | 0.994 | 0.989 | 0.911 | 0.940 | 0.982 |
| Spam | 0.887 | 0.006 | 0.958 | 0.887 | 0.921 | 0.911 | 0.940 | 0.865 |
| Weighted Avg. | 0.980 | 0.099 | 0.980 | 0.980 | 0.980 | 0.911 | 0.940 | 0.967 |

|  |  |  |
| --- | --- | --- |
| Confusion Matrix | | |
| Classified As | A | B |
| a = Ham | 2376 | 14 |
| b = Spam | 41 | 321 |



# Discussion

The three results sets above detail the three cases of a) J48 being run on a set of 3000 features constructed using the TF-IDF model, b) Naïve Bayes being run on a set of 100 features constructed using the raw frequency model, and c) SMO being run on a set of 3000 features constructed using the TF-IDF model. Overall, SMO outperformed Naïve Bayes, which in turn outperformed J48.

The first thing that jumps out from the notable results is how well SMO performed. SVM classifiers can usually handle the text classification challenge very well, and these results are pretty solid evidence. Text data is often easily linearly separable due to its sparseness, which is why SVM does so well with text. SVM classifiers also tend to perform better with more features, due to kernels and optimizations such as SMO being able to handle high dimensional data. As seen, the 3000 features performed best for SMO.

Another interesting finding to note is that both of the notable results that had 3000 features also had the TF-IDF feature model. One theory for this is that since the TF-IDF model represents the data more richly, using information gain feature selection isn’t all that helpful, and just serves as a method of overfitting the data.

An interesting surprise is how well J48 performed. Decision tree classifiers tend to do poorly on text classification. Decision trees perform poorly in high dimensional spaces because there is not “core” or very common data that it can latch onto to create key nodes in the tree. J48 still performed the worst of the three classifiers, but it held its own quite well, especially when paired with the stronger TF-IDF model.

Lastly, why did the Naïve Bayes classifier do the best with only 100 features and from using the basic raw frequency count model of construction? This is because the Baysian statistics makes some big generalizations about data. The Naïve Bayes classifier assumes all data is independent, and therefore can be very insensitive to overfit data. Since Naïve Bayes makes these types of generalizations, it can actually do more with less. The classifier can learn with less data and lower quality data, which can be a plus in terms of efficiency and data collection. Of course, because of these caveats, Naïve Bayes is a more simplistic classifier, and usually never provides the best results.

# Conclusion

As seen in the findings presented, classification, paired with proper data pre-processing, is quite the viable option when it comes to detecting spam messages. This can especially be seen in the top results of the SMO classification approach. Of course, improvement is always welcome, and can certainly be made with enough time, money, and effort. No machine will ever be perfect in detecting every single spam SMS message, but the gap towards one hundred percent can continually be narrowed over time as new technologies and methodologies are discovered.

# Additional Results

Additional text output results from other classification runs with different feature construction and feature selection methods can be found in the “results” directory of the project.

Additional visual ROC curves from other classification runs with different feature construction and feature selection methods can be found in the “visuals” directory of the project.

# References & Additional Resources

1. SMS Spam Collection Data Set. UCI Machine Learning Repository. <http://archive.ics.uci.edu/ml/datasets/SMS+Spam+Collection>

2. TF-IDF, A Single Page Tutorial. <http://www.tfidf.com/>

3. Snowball Stemmer. <http://snowball.tartarus.org/>

4. Data Mining – Information Gain. <http://gerardnico.com/wiki/data_mining/information_gain>

5. C.45 Algorithm. Wikipedia. <https://en.wikipedia.org/wiki/C4.5_algorithm>

6. Naïve Bayes Classifier. Wikipedia. <https://en.wikipedia.org/wiki/Naive_Bayes_classifier>

7. Powers, David M W (2011). "Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness & Correlation”. <http://www.bioinfopublication.org/files/articles/2_1_1_JMLT.pdf>

8. The Area Under an ROC Curve. <http://gim.unmc.edu/dxtests/roc3.htm>

9. Weka. The University of Waikato. <http://www.cs.waikato.ac.nz/ml/weka/>