SMS Spam Detection Using Machine Learning and Text Mining

Term Project Report

for

CSC 869 – Data Mining

Submitted to

Prof. Hui Yang

by

Rohan Samir Patel

SFSU ID: 917583698

Spring 2018

1. Introduction

SMS Spam is defined as unwanted text messages that are often used for spreading unwanted ads. Spam messages often contain malicious links and are notoriously used by hackers to steal user information. To add to this, spam messages in some countries prove to be a major problem because they often incur an additional cost. This, along with the lack of major mobile phone spam filtering software is my motivation behind exploring the problem of SMS spam detection through the use of data mining and machine learning techniques.

Throughout the course of this project, my main goal has been to find knowledge in data through the KDD process by first collecting, analysing and understanding the data, then applying pre-processing techniques to transform the data in a way that can be understood by a machine learning algorithm, then applying classification algorithms to make predictions on the data, and finally using sound evaluation strategies to evaluate and further improve my results.

2. Data

For this project two different text message datasets have been used. The first dataset is the SMS Spam Collection Data Set that has been collected from the popular UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/sms+spam+collection). This dataset contains a total of 5572 SMS messages out of which 4825 are non-spam (ham) messages and 747 are spam messages. The dataset file contains one message per line. Each line is composed of two columns: one with label (ham or spam) and other with the raw text. Here are some examples:

ham What you doing? how are you?

ham Cos i was out shopping wif darren jus now n i called him 2 ask wat present he wan lor. Then he started guessing who i was wif n he finally guessed darren lor.

spam FreeMsg: Txt: CALL to No: 86888 & claim your reward of 3 hours talk time to use from your phone now! ubscribe6GBP/ mnth inc 3hrs 16 stop?txtStop

The second dataset is a spam message dataset that has been scraped from Dublin Institute of Technology's website. This dataset contains only spam messages and has a total of 1353 spam messages.

The primary dataset of the project is the UCI dataset which has been used for the most part of the project for model building and evaluation. The Dublin dataset is simply used in the final stages when we use it to test the bias in our learned classifiers.

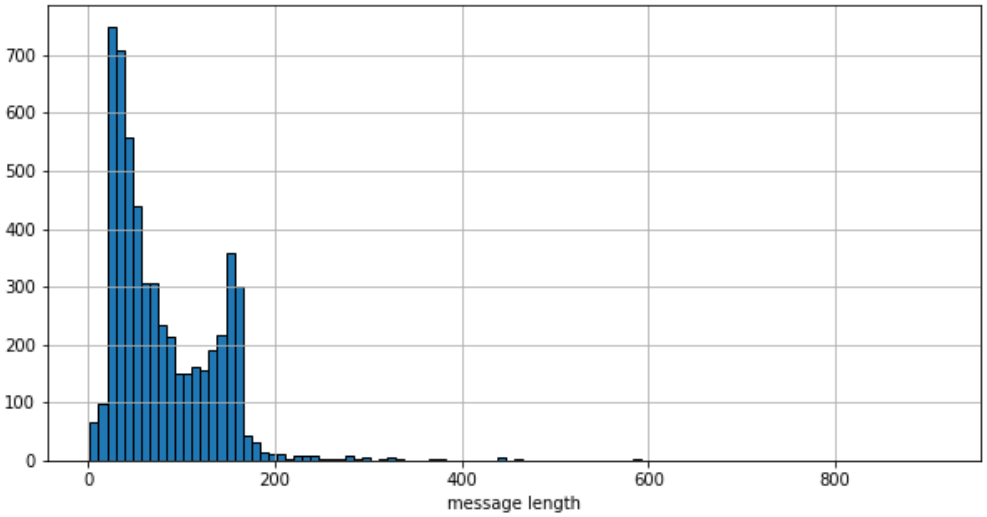
3. Methodology / Main Steps

The following section gives a brief description of the methodology and main steps that have been followed in this project.

3.1 Data Analysis and Feature Engineering

The first step involved performing exploratory data analysis on the dataset with the aim of analysing and understanding the data better. I used python's pandas and matplotlib library to do some basic analysis and data visualization. One of the major findings in this step was that message length was found to be somewhat correlated to the label of the message. The figure below shows the distribution of the message length. As the figure shows, there are two distinct peaks, the first one is for ham

messages that were found to have a shorter length than the spam messages which are represented by the second peak. As a result, an important step in feature engineering was to add message length as a new feature in the dataset. Another important step was organizing the entire message dataset into a pandas dataframe. This made data manipulation a lot easier for the entire course of the project.



3.2 Text Pre-processing and Transformation

The next step was to apply some basic text pre-processing techniques to the data to clean the text messages. There were three major things that were done, punctuation removal, stopwords removal and stemming. The result of this step is a list of clean word tokens that are stored in a new processed_msgs.csv file in the output folder of the project directory. These word tokens are then used to convert each message into a 'bag of words' and then into a TF-IDF vector representation. The message length feature is then appended to this TF-IDF matrix to produce the final feature vector which I later fit into my machine learning classifier.

3.3 Text Classification

Once I have my feature vector, consisting of the TF-IDF scores for each message and the length feature, I split it into a training and test set using a 70:30 split. The training set is used to train various classifiers, namely, SVM, Multinomial Naïve Bayes, Decision Tree, KNN, Random Forest, AdaBoost and Bagging Classifier. The test set is then used to make predictions. Once I have the predictions, I compare the classifiers based on accuracy, f1-measure, precision and recall to get an initial idea of how each classifier performs. In this step I also generate a file called misclassified_msgs.txt in the output folder, that contains the list of all misclassified text messages for each classifier.

3.4 Parameter Tuning

Next, I apply parameter tuning to try and further improve the results of my learned classifiers. I achieve this by using scikit-learn's GridSearchCV to perform an exhaustive grid search. Exhaustive grid search is a way to select the best model out of a family of models by performing K-fold cross-validation and tuning the model over a grid of model parameters.

3.5 K-Fold Cross Validation and Learning Curve

Next, I apply 5-fold cross validation and plot a learning curve to evaluate my learned classifier. The learning curve is generated by plotting the error vs. the training set size (i.e., how better does the model get at predicting the target as you the increase number of instances used to train it). It is important to note that in the learning curve, there are two error scores that are monitored: one for the validation set, and one for the training sets. We plot the evolution of the two error scores as training sets change and end up with two curves. The evolution of the two curves over the training set size gives us a good indication of bias and variance in our model and thereby diagnose whether our classifier shows any underfitting or overfitting symptoms.

3.6 Test learned classifier on 2nd Dataset to check for bias

In the next step, I use the learned classifiers from the previous steps to test how they perform on my second dataset that has been scraped from Dublin Institute of Technology's website. In this step, I re-train my classifiers using previously learned model parameters on the entire UCI message dataset. The Dublin dataset is used simply as the test set and not used for training purpose here. If accuracy of predictions on the Dublin dataset are close to previously achieved accuracies, then we know our model doesn't suffer from bias.

3.7 Addressing Imbalance of UCI Dataset

In the final step of my project, I try to address the imbalance of the UCI message dataset. Originally, the UCI dataset has 4825 (86.6%) ham and 747 (13.4%) spam messages, which seems like a rather big difference. Hence, in this step I cut down the UCI message dataset to 747 spam messages and 1000 ham messages to produce a more even playing field. Once, I have a more balanced dataset, the classifiers are re-trained to check if changing the composition of the dataset has any impact on the final results.

4. Instructions for Compiling

This project has been developed using Python3 and hence you will need Python3 installed on your system to compile and run this project. The project folder contains a total of seven python scripts namely, read_data.py, text_preprocessing.py, text_classification.py, parameter_tuning.py, learning_curve.py, check_bias.py, and address_imbalance.py that must be compiled to produce the final results. The datafiles are present in the folder named 'smsspamcollection'. Here there are two files, SMSSpamCollection that contains the UCI dataset and spam.xml that contain the Dublin dataset. The output folder contains certain output files that are generated by running the programs. The step-wise instructions to compile and run the program are given below.

a) First install all dependencies.

```
$ pip install numpy

$ pip install pandas

$ pip install nltk

$ pip install scipy

$ pip install matplotlib
```

```
$ pip install scikit-learn
```

b) Once all dependencies are installed you can navigate to the project directory on your terminal. The first script to execute is read_data.py. This script only reads the UCI message dataset file and prints the first 100 messages to give you an idea of what the data looks like.

```
$ python3 read_data.py
```

c) Next you can run the script containing the text pre-processing module.

```
$ python3 text_preprocessing.py
```

d) Next you can run the script containing the initial text classification module.

```
$ python3 text_classification.py
```

e) Next you can run the parameter tuning module.

```
$ python3 parameter_tuning.py
```

NOTE: Parameter tuning on SVM takes a very long time (approx. 1 hour) and hence has been commented out on line 81 of the script. The above script will hence only produce parameter tuning results for Multinomial NB. But if you want to see the results for SVM and time permits please uncomment line 81 and execute.

f) Next you can run the script that contains the k-fold cross validation and learning curve modules.

```
$ python3 learning_curve.py
```

g) Next you can run the script that tests the learned classifier on the Dublin dataset to check for bias.

```
$ python3 check_bias.py
```

h) Finally, you can run the final script that addresses the imbalance of the UCI dataset.

```
$ python3 address_imbalance.py
```

NOTE: The Jupyter Notebook (CSC869-TermProject.ipynb) for this project is also provided in the project folder along with an html version of the notebook. You can also use those to run and view the project respectively.

5. Evaluation and Analysis

The following section covers a detailed discussion of the results produced.

5.1 Text Pre-processing

```
################################################## Processed Messages ##################################################
```

	label	message	length	processed_msg
0	ham	Go until jurong point, crazy.. Available only ...	111	[go, jurong, point, crazi, avail, onli, bugi, ...
1	ham	Ok lar... Joking wif u oni...	29	[ok, lar, joke, wif, u, oni]
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	155	[free, entri, 2, wkli, comp, win, fa, cup, fin...
3	ham	U dun say so early hor... U c already then say...	49	[u, dun, say, earli, hor, u, c, alreadi, say]
4	ham	Nah I don't think he goes to usf, he lives aro...	61	[nah, dont, think, goe, usf, live, around, tho...
5	spam	FreeMsg Hey there darling it's been 3 week's n...	147	[freemsg, hey, darl, 3, week, word, back, id, ...
6	ham	Even my brother is not like to speak with me. ...	77	[even, brother, like, speak, treat, like, aid,...
7	ham	As per your request 'Melle Melle (Oru Minnamin...	160	[per, request, mell, mell, oru, minnaminungint...
8	spam	WINNER!! As a valued network customer you have...	157	[winner, valu, network, custom, select, receiv...
9	spam	Had your mobile 11 months or more? U R entitle...	154	[mobil, 11, month, u, r, entitl, updat, latest...
10	ham	I'm gonna be home soon and i don't want to tal...	109	[im, gonna, home, soon, dont, want, talk, stuf...
11	spam	SIX chances to win CASH! From 100 to 20,000 po...	136	[six, chanc, win, cash, 100, 20000, pound, txt...
12	spam	URGENT! You have won a 1 week FREE membership ...	155	[urgent, 1, week, free, membership, £100000, p...
13	ham	I've been searching for the right words to tha...	196	[ive, search, right, word, thank, breather, pr...
14	ham	I HAVE A DATE ON SUNDAY WITH WILL!!	35	[date, sunday]
15	spam	XXXMobileMovieClub: To use your credit, click ...	149	[xxxmobilemovieclub, use, credit, click, wap, ...
16	ham	Oh k...i'm watching here:)	26	[oh, kim, watch]
17	ham	Eh u remember how 2 spell his name... Yes i di...	81	[eh, u, rememb, 2, spell, name, yes, v, naught...
18	ham	Fine if that□s the way u feel. That□s the way ...	56	[fine, that□, way, u, feel, that□, way, gota, b]
19	spam	England v Macedonia - dont miss the goals/team...	155	[england, v, macedonia, dont, miss, goalsteam,...
20	ham	Is that seriously how you spell his name?	41	[serious, spell, name]
21	ham	I‘m going to try for 2 months ha ha only joking	47	[i'm, go, tri, 2, month, ha, ha, onli, joke]
22	ham	So ü pay first lar... Then when is da stock co...	52	[ü, pay, first, lar, da, stock, comin]
23	ham	Aft i finish my lunch then i go str down lor. ...	88	[aft, finish, lunch, go, str, lor, ard, 3, smt...

The figure above shows the result of the pre-processing. The message column shows the original raw text message from the dataset while the processed_msg column shows the list of clean word tokens that are generated that have been stemmed and stripped of punctuations and stopwords.

5.2 Text Classification

Classifier	Precision	Recall	F1-Measure	Support	Accuracy
SVM	0.78	0.88	0.83	1672	0.88
Decision Tree	0.96	0.96	0.96	1672	0.96
Multinomial NB	0.9	0.88	0.83	1672	0.88
KNN	0.93	0.94	0.93	1672	0.93
Random Forest	0.97	0.97	0.97	1672	0.97
AdaBoost	0.95	0.96	0.95	1672	0.95
Bagging Classifier	0.97	0.97	0.97	1672	0.97

The above table shows the results of initial text classification. The ensemble classifiers, Random Forest and Bagging Classifier seem to be giving the best results. What is interesting to note is that a lot of literature states that Naïve Bayes, in particular Multinomial NB works very well with text data. The reason is due to the nature of Naïve Bayes. In a text message the probability of occurrence of one word is independent of the probability of occurrence of another word. And hence past research suggests that Multinomial NB has the best performance amongst all classifiers when it comes to text

classification. And yet in my case it gives the worst performance. My guess is that this happens because I have appended message length as an additional feature to the dataset. Message length might have affected the performance of Naïve Bayes and perhaps it would have performed better if we had simply used text as our features.

Another interesting observation is that SVM which is regarded as one of the most sophisticated classifiers is also giving a less accuracy. By applying parameter tuning, however, we could boost the performance of both these classifiers.

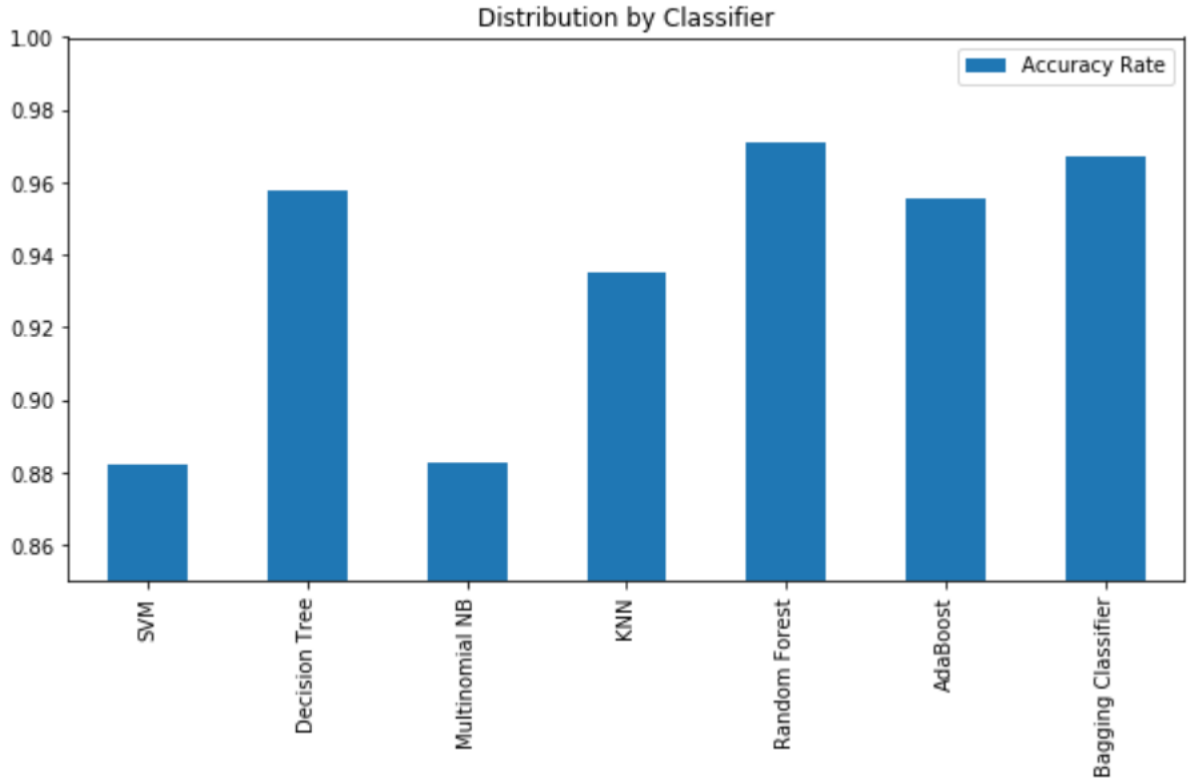


Figure above shows bar graph representation of the accuracy scores from the initial text classification

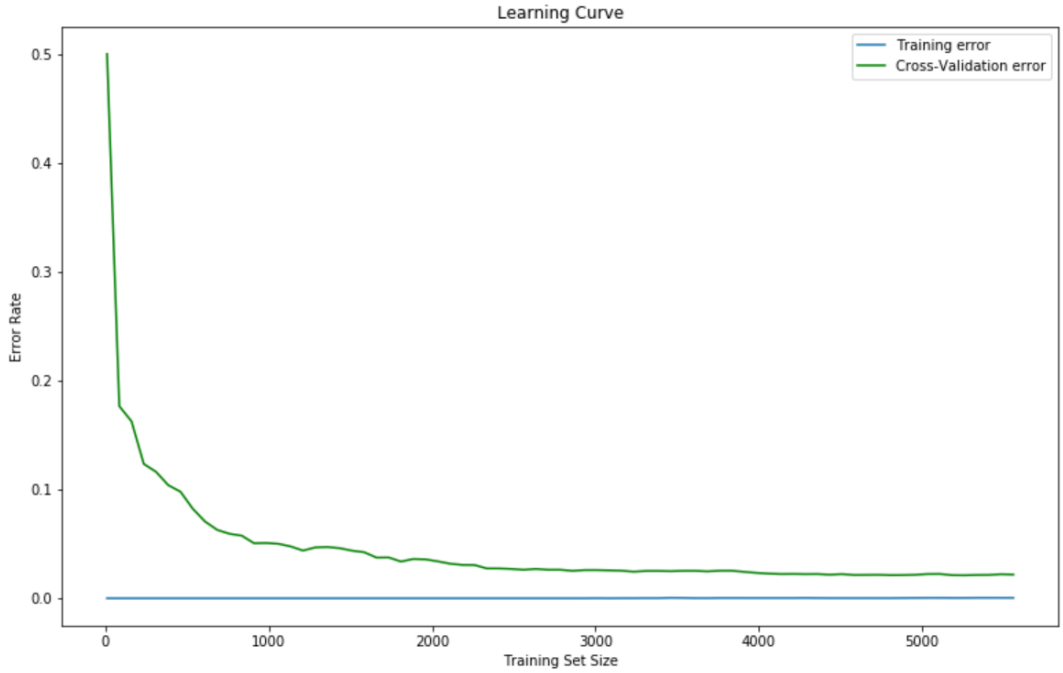
5.3 Parameter Tuning

Classifier	Precision	Recall	F1-Measure	Support	Accuracy
SVM	0.98	0.98	0.98	1672	0.978
Multinomial NB	0.98	0.98	0.98	1672	0.98

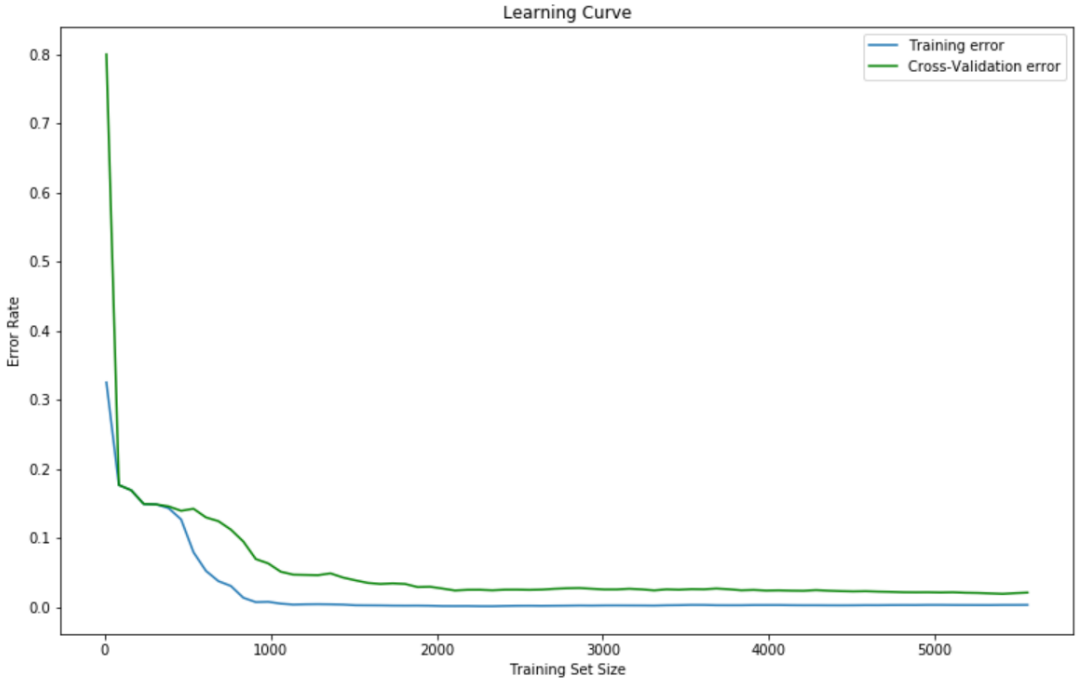
The above table shows the results after applying parameter tuning. Clearly there is a significant boost in the accuracy scores. The parameters that works best for SVM is a linear kernel with gamma = 1.0

Similarly, for Multinomial Naïve Bayes the parameter that gives the best model is alpha ≈ 0.1

5.4 Learning Curve



The above figure shows the learning curve for SVM. Looking at evolution of the learning curve it seems our classifier is suffering from slight overfitting. The model is almost fitting all data points and hence the training error always stays low. On the other hand, the cross-validation error continues to decrease as training set size increases. This is classic behaviour in an overfitting classifier.



The above figure shows the learning curve for Multinomial Naïve Bayes. It seems as though this is also overfitting.

5.5 Test learned classifier on Dublin dataset to check for bias

Classifier	Precision	Recall	F1-Measure	Support	Accuracy
SVM	1	0.84	0.91	1353	0.835
Multinomial NB	1	0.89	0.94	1353	0.888

The above table shows the results of testing our learned classifiers on the Dublin spam dataset. As the table shows, the support is 1353 which is the entirety of the Dublin dataset. The entire dataset is used for testing and only the UCI dataset is used for the training phase. Keeping that in mind, we are getting pretty good results. Although the accuracy scores aren't as high as 0.98, they are pretty close which suggests that the classifiers and the methodology followed in this project is not biased. This is a big positive as past research on SMS Spam detection has only focussed on working with the UCI dataset and the proposed classifiers and techniques are clearly suffering from bias.

One thing that needs to be noted is that the Dublin dataset only contains spam messages and hence the scores shown in the table above are only for spam messages.

5.6 Addressing Imbalance of UCI Dataset

Classifier	Precision	Recall	F1-Measure	Support	Accuracy
SVM	0.95	0.95	0.95	525	0.94
Multinomial NB	0.95	0.95	0.95	525	0.95

The above table shows the results produced after addressing the balance of the UCI dataset. The support is only 525 because the number of ham messages in the dataset has been significantly cut down (from 4825 to only 1000 ham msgs). There is a small drop in the accuracy compared to the one's produced after parameter tuning on the entire dataset. This suggests that imbalance of the dataset probably doesn't have any impact the final results in this case.

6. Pros & Cons

Pros:

- Overall the precision, recall, f1-measure and accuracy scores are pretty high across the board.
- Proposed strategy addresses the balance of the UCI SMS Spam dataset.
- Perhaps the biggest pro of this project is that it is able to prove that the followed methodology and strategies are not biased. A lot of past research on SMS Spam detection using the UCI dataset solely focuses on using one dataset to show and evaluate the produced results. Subsequently the strategies turn out to be clearly biased. In my opinion, having an additional dataset in the Dublin dataset, to test the learned classifier gives this work a lot of merit.

Cons:

- The method does suffer from the overfitting problem.
- Overall, inspite of using two different datasets, we only have a sample of approximately 7500 SMS messages. Having more data could definitely help.

7. Post Analysis and Future Work

The primary goal of this project right from the beginning was to follow every step of the KDD cycle and come up with a sound, novel approach to solve the problem of SMS Spam detection. My goal was always to keep things simple but do full justice to the KDD process and I believe I succeeded in doing so.

Moving forward I think this project could be taken into interesting directions. Deep Learning is an area that has always fascinated me, and while there were times during the course of this project when I thought of incorporating neural networks, eventually I decided to keep things simple and use classic machine learning techniques taught in class, instead of going in a direction that I did not have a strong knowledge base in. However, in the future, I think it would be rather interesting to see what results could be produced if neural networks and deep learning was introduced into this method.