Machine Learning Capstone Project: Natural Language Processing and SMS Prediction

Bhavya Garg
April 11, 2017

Definition

Project Overview

This report provides an overview of my work completed for the Machine Learning Nanodegree Capstone project offered by Udacity. The objective of this project is to classify SMS phone messages to Ham/Spam using a collection of more than 5 thousand SMS phone messages labeled ham and spam examples. This has been accomplished by using the Data Science area of Natural Language Processing (NLP) to build a Predictive Model in python.

Natural language processing (NLP) is a component of artificial intelligence (AI) which is used to analyze text, allowing machines to understand how human's speak. This human-computer interaction enables real-world applications like automatic text summarization, sentiment analysis, topic extraction, named entity recognition, parts-of-speech tagging, relationship extraction, stemming, and more. NLP is commonly used for text mining, machine translation, and automated question answering. Natural language processing technology is designed to derive meaningful and actionable data from freely written text. This is particularly useful because it allows to record information in a natural manner and when executed well, it enables a more natural transition between person and database. Here being inspired by spam email classification I've chosen to work on text messaging data to classifying spam/ham messages. Spam is a waste of time and takes unnecessary storage space. There are many other areas where NLP are being used, for example YELP is using NLP to understand people comments and provide ratings based on that.

This report illustrates the SMS classification process using machine learning techniques. It starts by a small introduction about the datasets, followed by an exploratory data analysis that shows some summary statistics about the data sets, some visualization to see the trends and then interesting findings discovered so far and finally the best prediction algorithm.

Problem Statement

Modern NLP is based on machine learning. A Predictive Model is learned via Text Mining analysis of a sufficiently large corpora (set of real-world documents). In my case the Predictive Model application is to classify the given text sms as ham/spam.

I will follow NLP high level common steps which are:
1) Importing data
2) Basic Exploratory Data Analysis
3) Text Pre-Processing
4) Vectorization
5) General preprocessing with a combination of Machine Learning processing (training and testing) to deliver the desired prediction model.

Datasets and Inputs

This project will use SMS Spam Collection(CORPUS)which is a set of SMS tagged messages that have been collected for SMS Spam research. The dataset is available at https://archive.ics.uci.edu/ml/datasets/SMS+Spam+Collection and contains 5574 messages in total out of which 4827 SMS are ham and 747 are spam.

The SMS Spam Collection file contain one message per line. Each line is composed by two columns: one with label (ham or spam) and other with the raw text.

The files contain one message per line. Each line is composed by two columns: one with label (ham or spam) and other with the raw text.

Metrics

Model evaluation is the phase where we want to determine how well our model will do overall on the entire dataset. There are various methods to determine effectiveness; however, precision, recall, and accuracy are most often used. Which one is the most important depends on the task and the business effects of decisions based off the model. I have used SciKit Learn's built-in classification report, which returns Accuracy, precision, recall and f1-score.

1) Accuracy: Accuracy is the ratio of correctly predicted observations out of total predictions made.

2) Precision: Precision the ratio of correct positive predictions made from the total positive predictions made.

$$\text{Precision} = \frac{tp}{tp + fp}$$

3) Recall: Recall is the ratio of correct positive predictions made from the actual total that were positive (correct positive prediction plus incorrect false negative prediction). The formula is

$$\text{Recall} = \frac{tp}{tp + fn}$$

4) F score: The F1 score can be interpreted as a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst at 0:
F1 = 2 * (precision * recall) / (precision + recall)

Since my dataset is imbalanced where one class dominates over the other, Accuracy is not only the metric to consider because that is misleading and in my case, there are 4827 ham SMS and 747 spam meaning the ham messages are dominating the spam and if I calculate the accuracy it will give me 87% accuracy considering all messages as ham, that is the reason I chose precision, recall and f1 score along with the accuracy as an evaluation metrices.

Analysis

Data Exploration & Visualization

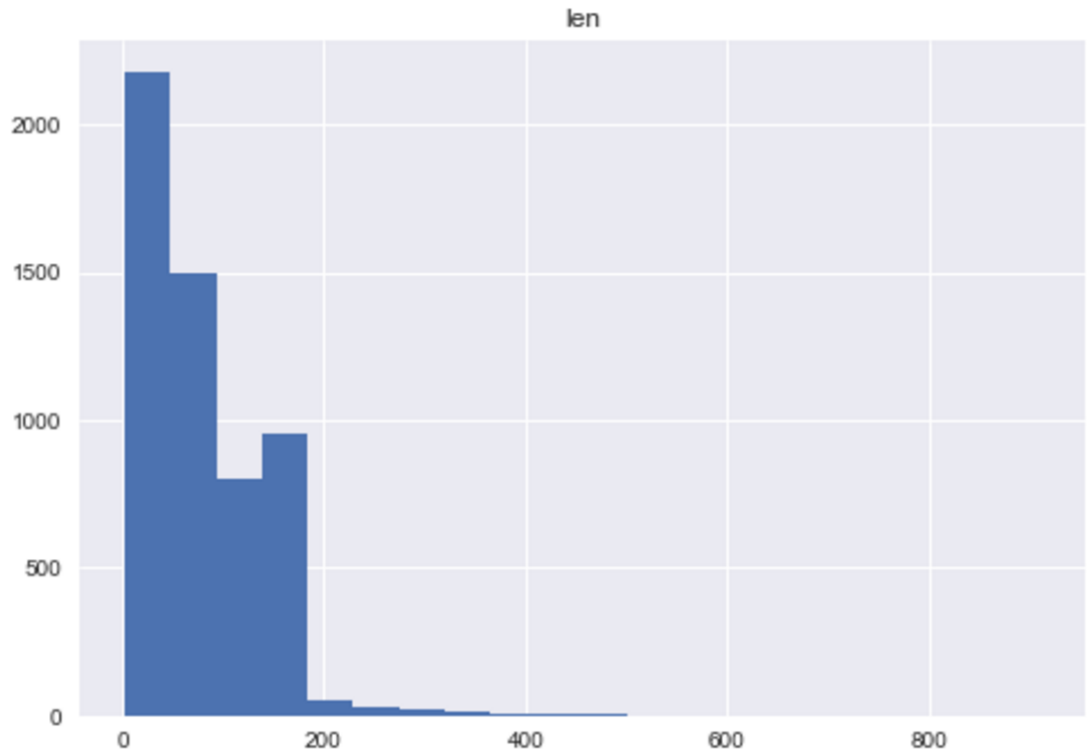
My dataframe has 5572 observations with 2 variables (label, message).

I started my data exploration with checking for the counts and percentage of the ham and spam messages and figured out that the dataset is highly imbalanced with 4825 ham messages and just 747 spam messages meaning 87% of the total messages in the corpus are ham and just 13% belongs to the spam class.

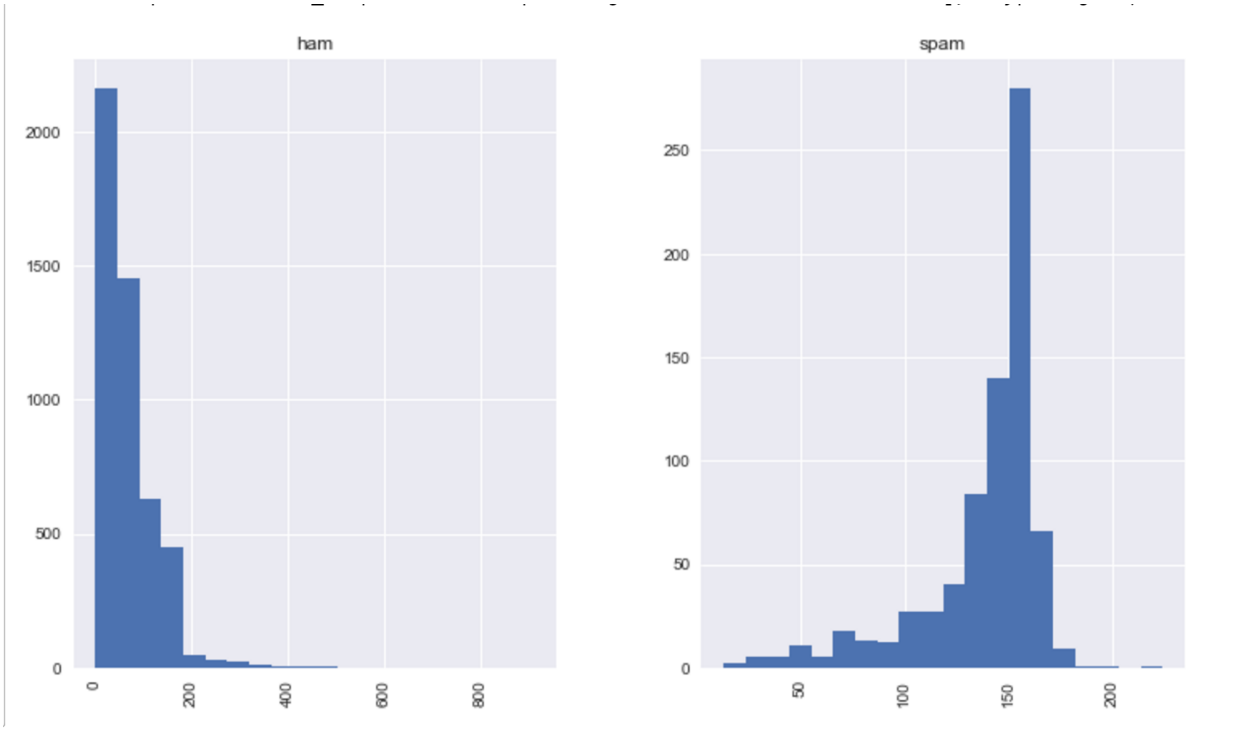
I had also creating a new feature called "len" which tells the length of the text messages. I applied descriptive statistics on the length variable and got the below result which tell that the mean length of the text messages is of length 80 with min of 2 letters and max of 910 letters which tells that the data is highly skewed.

```
count    5572.000000
mean       80.616296
std        60.015593
min         2.000000
25%        36.000000
50%        62.000000
75%       122.000000
max       910.000000
Name: len, dtype: float64
```

Then I created the histogram based on the length variable and observed that the length of the messages is highly skewed towards the left meaning most of the messages length are between 0 to 200 length but there are few messages which are long up to 500 length.



Let’s visualize the length across label



Based on the above visualization ham messages tend to be shorter and spam messages are of longer length.

Algorithms and Techniques

A wide variety of techniques have been designed for text classification. Since text may be modeled as quantitative data with frequencies on the word attributes, it is possible to use most of the methods for quantitative data directly on text. However, text is a particular kind of data in which the word attributes are sparse, and high dimensional, with low frequencies on most of the words. Therefore, it is critical to design classification methods which effectively account for these characteristics of text.

I had used Bag of Words approach to convert the list of tokens into numeric representation for machine learning. The Bag of Words model learns a vocabulary from all the documents, then models each document by counting the number of times each word appears. I have used the below bag of words approach:

- Count how many times does a word occur in each message (Known as term frequency)
- Weigh the counts, so that frequent tokens get lower weight (inverse document frequency)

TF: Term Frequency, which measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. Thus, the term frequency is often divided by the document length (aka. the total number of terms in the document) as a way of normalization:

TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document).

IDF: Inverse Document Frequency, which measures how important a term is. While computing TF, all terms are considered equally important. However, it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus, we need to weigh down the frequent terms while scale up the rare ones, by computing the following:

IDF(t) = log_e(Total number of documents / Number of documents with term t in it).

I've used three classifiers separately to train the model. Those three classifiers are Naïve Bayes, SVM, and Random forest.

Bayesian Classifiers: Naive Bayes is often used in text classification applications and experiments because of its simplicity and effectiveness. However, its performance is often degraded because it does not model text well. In Bayesian classifiers, we attempt to build a probabilistic classifier based on modeling the underlying word features in different classes. The idea is then to classify text based on the posterior probability of the documents belonging to the different classes based on the word presence in the documents

SVM Classifiers: SVM Classifiers attempt to partition the data space with the use of linear or non-linear delineations between the different classes. The key in such classifiers is to determine the optimal boundaries between the different classes and use them for the purposes of classification.

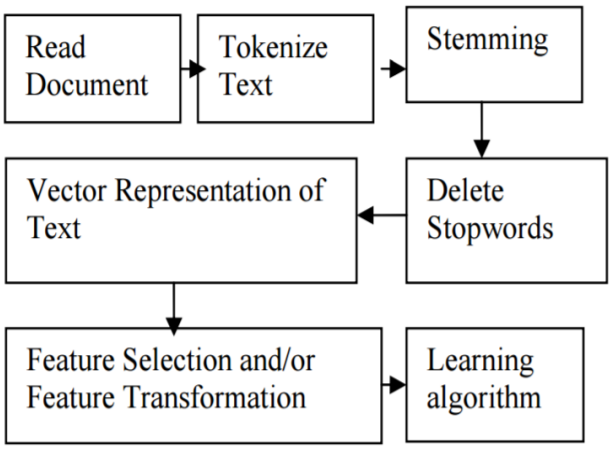
Random Forest: Random forests also known as Ensemble method are a way of averaging multiple deep decision trees, trained on different parts of the same training set, with the goal of overcoming over-

fitting problem of individual decision tree. For instance, a Random Forest might contain 10 Decision Trees. Each Decision Tree receives a different subset of the data from the training dataset, so each will be slightly different and predict slightly differently. To receive the predictions from the whole forest, the predictions of the trees are averaged out in case of regression and voting in case of classification. The main principle behind ensemble methods is that a group of "weak learners" can come together to form a "strong learner".

Benchmark Model

Although the main aim of this project is to classify the SMS messages to ham/spam but the measure of success is how accurately my model is making prediction in classifying ham vs spam messages. I am choosing the benchmark model with the accuracy of 87% considering all messages as ham since my data sets are strongly biased towards ham and I can get a minimum accuracy of 87% by just saying that every single email is ham, which is obviously something that indicates a non-ideal classifier. So, my aim here is to focus on getting over 95% accuracy with over 95% precision and recall value.

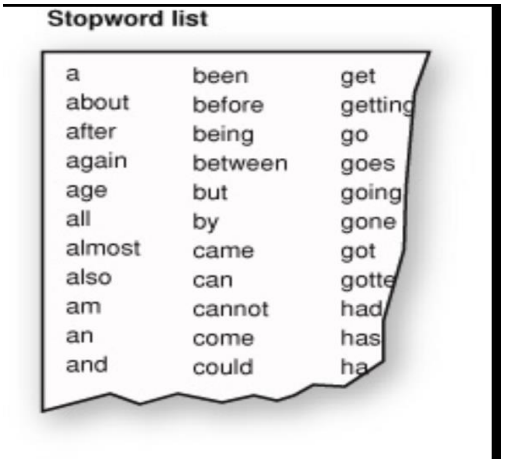
Methodology



Data Preprocessing

The preprocessing done in" NLP for SMS Classification "notebook consists of the following steps:

1) Split a message into its individual words.
2)Imported a list of English stopwords from NLTK library.



3) Created a function called “text process” which will remove the non-letters, punctuation, and stop words.
4) Converted each of the messages into a vector that machine learning models can understand. This was done in three steps using the bag-of-words model:

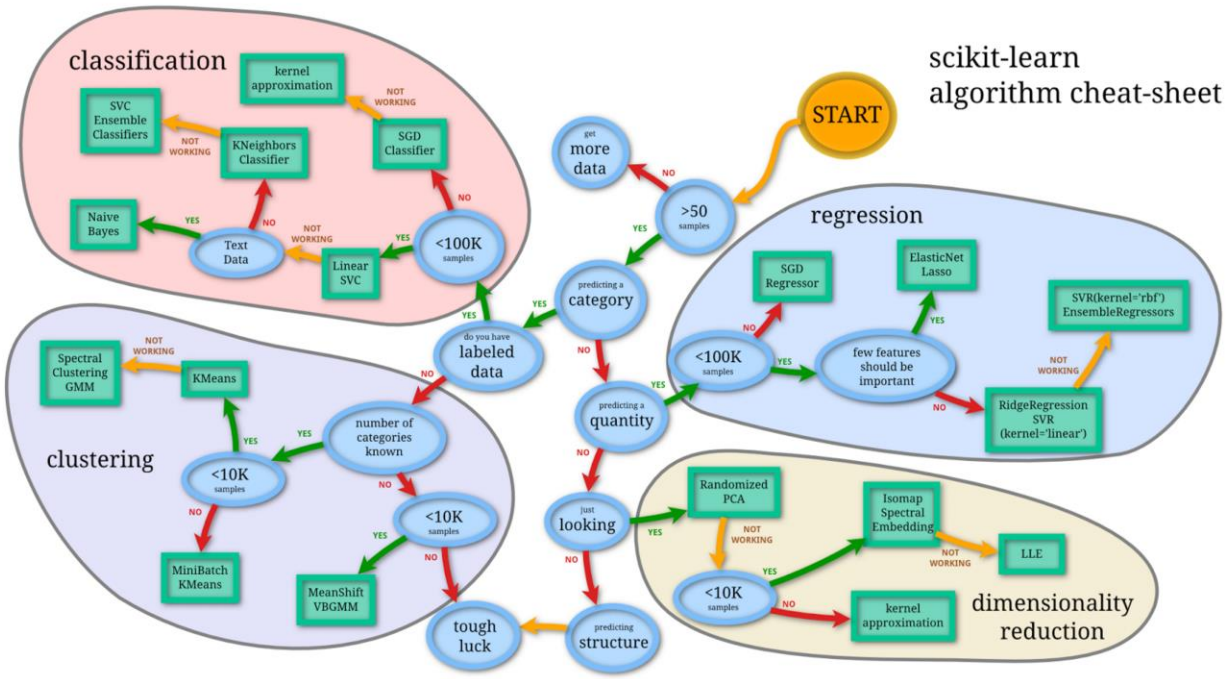
- Count how many times does a word occur in each message (Known as term frequency)
- Weigh the counts, so that frequent tokens get lower weight (inverse document frequency)
- Normalize the vectors to unit length, to abstract from the original text length (L2 norm)

Implementation

The implementation stage can be split into two main stages:

- The training stage: Once the messages are represented as vectors, we can finally train our spam/ham classifier. We can use any kind of classifier to build a model but I chose to start with Naive Bayes classification algorithm since it serves well for classifying texts then I continued to apply SVM and Random forest classifier to train my model.
- Validation stage: Now once we have the model we want to determine how well our model will do overall on the entire dataset. The best way to evaluate our model is to split the data into a training/test set, where the model is built based on the training dataset and we check the performance of the model on the test dataset.

I’ve started with Naive Bayes classification, but scikit-learn supports many classifiers out of the box: Support Vector Machines, Decision Trees, Ensemble methods etc. as shown below:



With **Naïve Bayes** classifier, I got an accuracy, precision and recall value of 98%, 97%, 97% as shown below:

	precision	recall	f1-score	support
ham	1.00	0.97	0.98	1507
spam	0.78	1.00	0.88	165
avg / total	0.98	0.97	0.97	1672

Then, I tried **Random forest** to check if it could further improve my accuracy and I found out that there was 1% increment of my precision and recall after using Random forest classifier and the result were as follows:

	precision	recall	f1-score	support
ham	1.00	0.98	0.99	1494
spam	0.84	1.00	0.91	178
avg / total	0.98	0.98	0.98	1672

Predicted	ham	spam
Actual		
ham	1460	0
spam	34	178

Finally, I gave it a try with SVM. SVMs are great when classifying text data, getting state of the art results very quickly and with pleasantly little tuning and the result was really shocking with 99% accuracy, precision and recall.

```
[[1457    3]
 [  21  191]]
Predicted   ham  spam
Actual
ham        1457     3
spam         21   191
             precision    recall  f1-score   support

        ham       0.99      1.00      0.99      1460
       spam       0.98      0.90      0.94       212

avg / total       0.99      0.99      0.99      1672
```

I also performed some sanity check to see how my different model are performing in classifying ham/spam messages.

Refinement

The scores I got with Naïve Bayes and Random Forest were really satisfying. But I still wanted to further improve my accuracy so I chose to work on Support Vector classifier with some of the parameters tuned. SVMs are great when classifying text data, getting state of the art results very quickly and with pleasantly little tuning and the result was really shocking with 99% accuracy, precision and recall but because Support vector classifier took comparatively longer time to fit the model so, I focused more on Naïve Bayes and Random Forest. And finally based on some sanity checks on some of the spam text messages from google, I chose Random forest as my final model.

RESULTS

Model Evaluation and Validation

I have chosen Random Forest over the SVM and Naïve Bayes because although Support vector classifier gave me 99% accuracy, the training time was around 27 min which was quite high compared to Random forest and naïve Bayes where it fitted my model in just couple of seconds. Also, there is just a difference of 1% in their accuracy values. Also, I performed few sanity check and the result are as below:

```
print Model_NB.predict(["URGENT! Credit for free!"])[0]
print Model_NB.predict(["Meeting aand Charming Women Long-winded ad for some cheesy-sounding videos"])[0]
print Model_NB.predict_proba(["URGENT! Credit for free!"])[0]
print Model_NB.predict_proba(["Meeting aand Charming Women Long-winded ad for some cheesy-sounding videos"])[0]
```

```
spam
ham
[ 0.43921977  0.56078023]
[ 0.86995923  0.13004077]
```

The predict_proba returns the predicted probability for each class (ham, spam). In the first case, the message is predicted to be spam with arou probability, and ham with 44%. And in the second case the message is predicted to be ham with around 87% probability, and spam with 13%.

```
print SVM_detector.predict(["URGENT! Credit for free!"])[0]
print SVM_detector.predict(["Meeting aand Charming Women Long-winded ad for some cheesy-sounding videos"])[0]
```

```
spam
ham
```

```
print Model_RF.predict(["URGENT! Credit for free!"])[0]
print Model_RF.predict([" Meeting aand Charming Women Long-winded ad for some cheesy-sounding videos"])[0]
```

```
ham
ham
```

Based on the above sanity check, for the message "URGENT! Credit for free!", my Random forest classifier misclassified it as ham whereas Naïve Bayes and SVM correctly identified it as spam. So overall based on training time and the accuracy, precision and recall value, I chose to select Naïve Bayes as my final classifier.

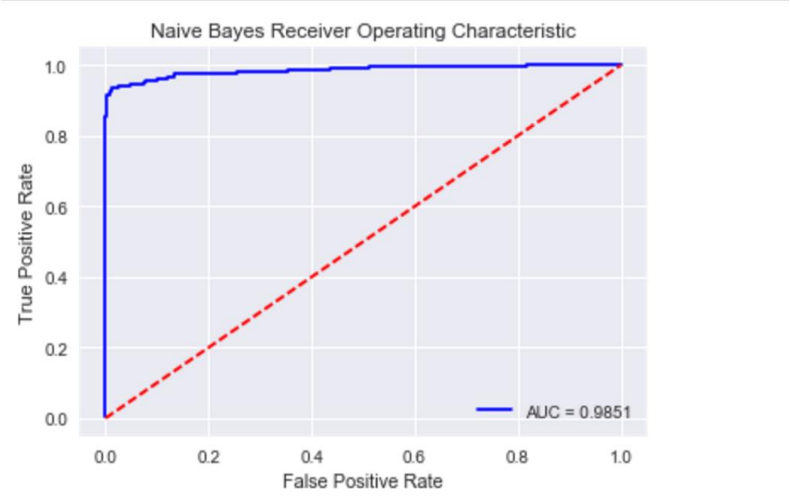
Justification

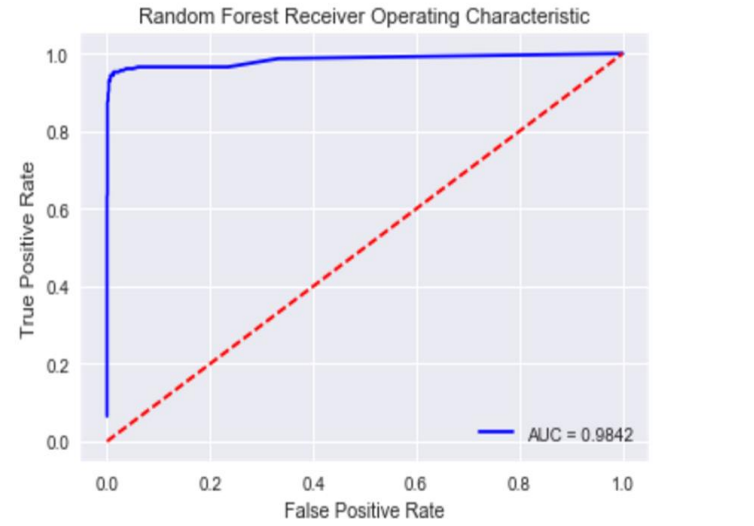
So far, my classification system has performed better than expected, with an accuracy of 98% (with a reasonable classification time) which is pretty high accuracy when compared to the benchmark model.

Conclusion

Free-Form Visualization

I had checked out for ROC AUC score, which be a measure of how efficient the model is to rank the items per probability of belonging to each class. This is often a good indicator when we want to get the predictions per order of probability (*e.g.* if we want to know which are the most likely to be class 1(spam))





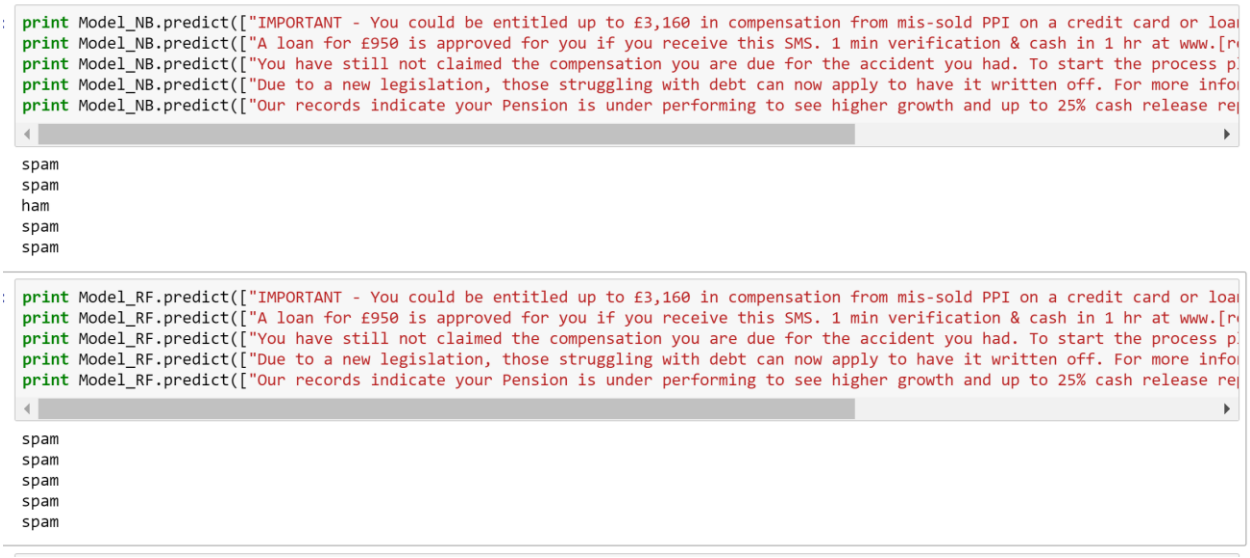
The charts above shows Receiver Operating Characteristic curve which typically feature true positive rate on the Y axis, and false positive rate on the X axis. Accuracy is measured by the area under the ROC curve. An area of 1 represents a perfect test; an area of .5 represents a worthless test

An ROC curve demonstrates several things:

1. It shows the tradeoff between sensitivity and specificity (any increase in sensitivity will be accompanied by a decrease in specificity).
2. The closer the curve follows the left-hand border and then the top border of the ROC space, the more accurate the test.
3. The closer the curve comes to the 45-degree diagonal of the ROC space, the less accurate the test.

In my case I got AUC of around 99% with both my models which is pretty great.

To find the robustness of the final model, I also did some sanity check. For that I took couple of spam messages from the internet wanted to cross check how my model is performing in classifying ham vs spam. Based on the chart above it shows that my Random Forest classifier could correctly classify all the spam messages whereas Naïve Bayes classifier misclassified one spam message as ham message.



```
print Model_NB.predict(["IMPORTANT - You could be entitled up to £3,160 in compensation from mis-sold PPI on a credit card or loa
print Model_NB.predict(["A loan for £950 is approved for you if you receive this SMS. 1 min verification & cash in 1 hr at www.[r
print Model_NB.predict(["You have still not claimed the compensation you are due for the accident you had. To start the process p
print Model_NB.predict(["Due to a new legislation, those struggling with debt can now apply to have it written off. For more info
print Model_NB.predict(["Our records indicate your Pension is under performing to see higher growth and up to 25% cash release re
```

```
spam
spam
ham
spam
spam
```

```
print Model_RF.predict(["IMPORTANT - You could be entitled up to £3,160 in compensation from mis-sold PPI on a credit card or loa
print Model_RF.predict(["A loan for £950 is approved for you if you receive this SMS. 1 min verification & cash in 1 hr at www.[r
print Model_RF.predict(["You have still not claimed the compensation you are due for the accident you had. To start the process p
print Model_RF.predict(["Due to a new legislation, those struggling with debt can now apply to have it written off. For more info
print Model_RF.predict(["Our records indicate your Pension is under performing to see higher growth and up to 25% cash release re
```

```
spam
spam
spam
spam
spam
```

Reflection

Below are the steps I followed for this project:

- Downloaded the Dataset from the UCI datasets.
- Exploratory Data Analysis: Once I had the dataset ready, I did some basic Exploratory data analysis on the dataset to get familiar with it.
- Pre-Processing: Once done with EDA, I did some text preprocessing steps like split a message into its individual words, remove common words, ('the', 'a', ‘can’, ‘they’ etc.), remove punctuation, converted the words into the lowercase etc. using NLTK library.
- Vectorization: Converted each of the messages into a vector that machine learning models could understand.
- Splitted the dataset into training and test dataset
- Trained a model: Once the messages were represented as vectors, I finally trained my spam/ham classifier using the training dataset. I used Naïve Bayes, SVM and Random forest classifier to train my model.
- Model evaluation: Once I had the model ready I wanted to evaluate the model on the test dataset to check the performance of each classifier.

Since I am very new to Natural Language Processing and this is my first project on NLP,I really enjoyed working on step 3 and 4 although it was bit challenging because I had to familiarize myself with using NLTK library and how to make use of all the modules present inside that.

Improvement

There might be few things which could have been improved:

- Fine tuning on Naïve Bayes and Random forest classifier
- Could have applied more sophisticated deep neural networks to train the model
- More real-time text sms could be collected to check the model performance.

References

https://www.kaggle.com/c/word2vec-nlp-tutorial/details/part-1-for-beginners-bag-of-words
http://scikit-learn.org/stable/modules/pipeline.html
http://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html
https://github.com/udacity/machine-learning/blob/master/projects/capstone/report-example-1.pdf
https://github.com/udacity/machine-learning/blob/master/projects/capstone/capstone_report_template.md
https://nbviewer.jupyter.org/github/jmportilla/Udemy---Machine-Learning/blob/master/NLP%20%28Natural%20Language%20Processing%29.ipynb
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.95.9153&rep=rep1&type=pdf