MACHINE LEARNING ENGINEER NANODEGREE

Capstone Project
Meena Joshi
July 29th 2017

I. Definition

Project Overview

This report provides an overview of my work completed for the Machine Learning Engineer Nanodegree Capstone project offered by Udacity. The objective of this project is to automatically classify the text messages as spam or ham (legitimate), using a dataset of SMS messages in English of more than 5000 messages. I will first train a machine-learning model to learn to discriminate between ham and spam. Then I will use this trained model to classify arbitrary unlabeled messages as ham or spam. I used Data Science part of Natural Language Processing (NLP) to build a predictive model in Python.

NLP (Natural Language Processing) is a field of study that focuses on the interactions between human language and computers. It is a way for computers to analyze, understand and derive meaning from human language in a smart and useful way. By utilizing NLP, we can organize and structure knowledge to perform tasks such as automatic text summarization, translation, named identity recognition, relationship extraction, sentiment analysis, speech recognition, topic segmentation and more. NLP algorithms are based on machine learning algorithms, i.e. it can rely on machine learning to automatically learn rules by analyzing a set of examples, and making the inference. The ability of computers to understand human language has become more relevant with the exponential growth of data, with wide applications in real world. For example, categorizing news stories by topics, organizing emails into various groups (social, promotion, primary), etc. I am interested to built a prediction model that can accurately classify spam and ham texts, to overcome the ever-increasing problem of unwanted texts.

This report explains the SMS classification process using machine-learning techniques. In this project I am using the dataset from UCI datasets of more than 5000 messages and labels. There is some exploratory data analysis that shows some summary statistics about the dataset, some visualization to see the trend. I am implementing various algorithms in Python to detect whether the message is a spam or a ham. The dataset is divided into training and testing set. I am using Naïve Bayes, Random Forest and Support Vector Machines algorithms to implement spam filtering and compared their performance and efficiency.

Problem Statement

The objective is to automatically classify the text messages as spam or ham (legitimate), using a dataset of SMS messages in English of more than 5000 messages; the tasks involved are the following using NLP and machine learning

1. Download 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.

I am using the SMS Spam Collection, which is a public set of SMS labeled messages that have been collected for mobile phone spam research.
https://archive.ics.uci.edu/ml/datasets/SMS+Spam+Collection
The dataset is composed in one text file, where each line has two columns the correct class label (ham/spam) and the raw message.

Metrics

To determine how well our model will perform on the entire dataset, there are some possible metrics for evaluating model's performance. The importance of the evaluation metrics depends on the task and the business effects of decisions based on the model. I have used scikit learn's built-in classification report, which returns precision, recall and f1-score.

- ***True Positive (tp)*** result is one that detects the condition when the condition is present.
- ***True Negative (tn)*** result is one that does not detect the condition when the condition is absent.
- ***False Positive (fp)*** result is one that detects the condition when the condition is absent.
- ***False Negative (fn)*** result is one that does not detect the condition when the condition is present.

Predicted Condition➔ / ⬇Actual Condition	Present	Absent
Present	tp	fn
Absent	fp	tn

- ***Accuracy*** is a common metric for binary classifiers; it takes into account both true positives and true negatives with equal weight.

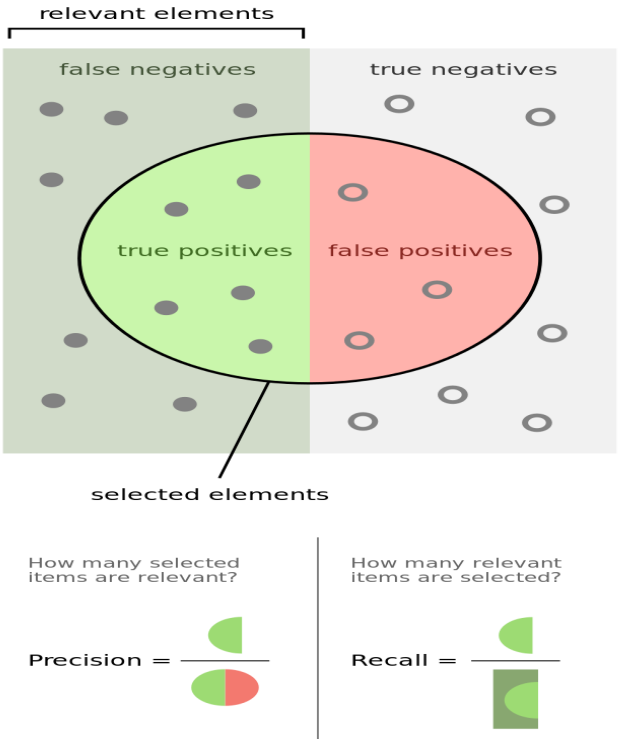
$$\boldsymbol{Accuracy} = \frac{\boldsymbol{tp} + \boldsymbol{tn}}{\boldsymbol{dataset\ size}}$$

- ***Precision*** is the ratio of true positive predictions and total positive predictions made.

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

- ***Recall*** is the ratio of true positive predictions and the total actual positive predictions.

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



- ***F1 Score*** is the weighted average of precision and recall, where F1 score reaches its best value at 1 and worst at 0.

$$\boldsymbol{F1 = \frac{2 * precision * recall}{(precision + recall)}}$$

For our dataset, which is highly imbalanced (4825 ham and 747 spam) accuracy should not only be the correct evaluation metric as it will tell us nothing about the true predictive power of the classifier. For instance, even classifying each message as ham can get a minimum accuracy of 87%. So, I will use precision, recall and f1-score along with accuracy as an evaluation metrics.

II. Analysis

Data Exploration

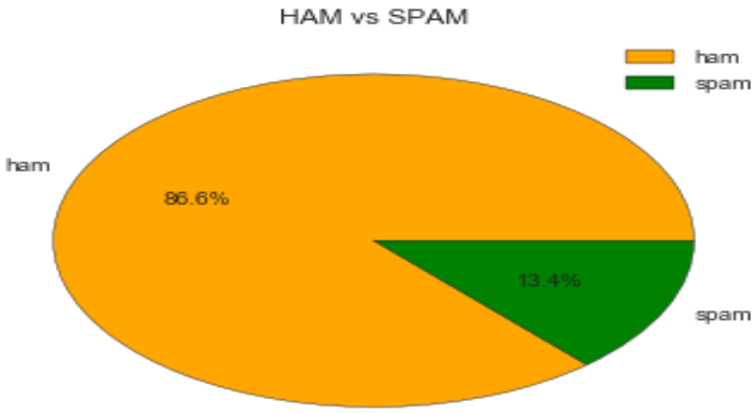
The dataset has 5572 messages with two columns (label, messages). First, I checked the counts and percentage of the two labels (ham and spam) and found that the dataset is highly imbalanced with 747 spams and 4825 hams. i.e 87% of messages are ham and only 13% are spams.

Then I created a new feature 'length', which tells the length of the messages. I applied descriptive statistics on the length variable and got the results for both ham and spam messages. It tells the mean length of ham messages is 71 with minimum 2 letters and maximum 910 letters and for spams the mean length is 138, with minimum 13 letters and maximum 223 letters, which tells that the data is highly skewed. The results I got are:

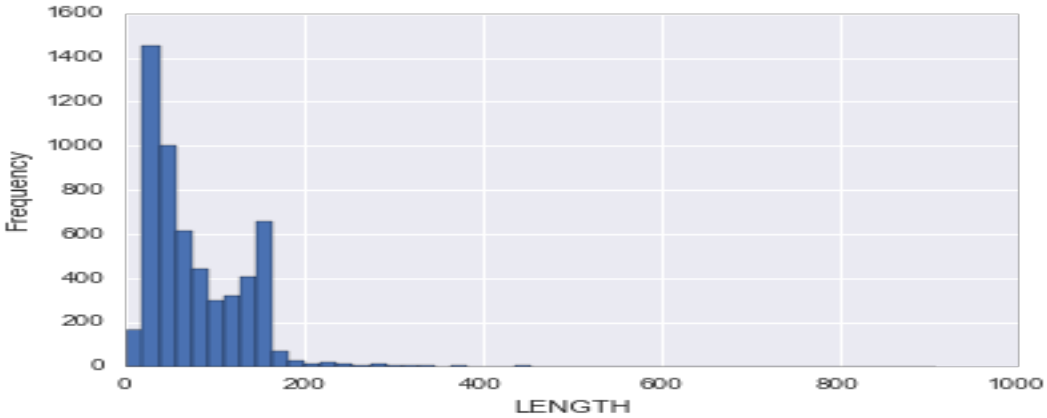
```
ham   count  4825.000000
      mean     71.482487
      std      58.440652
      min       2.000000
      25%      33.000000
      50%      52.000000
      75%      93.000000
      max     910.000000

spam  count   747.000000
      mean    138.670683
      std      28.873603
      min      13.000000
      25%     133.000000
      50%     149.000000
      75%     157.000000
      max     223.000000
```

Exploratory Visualization

I visualized the percentage of spams and hams and found the results below, which tell the dataset is highly imbalanced with 87% of messages as ham and only 13% are spams.

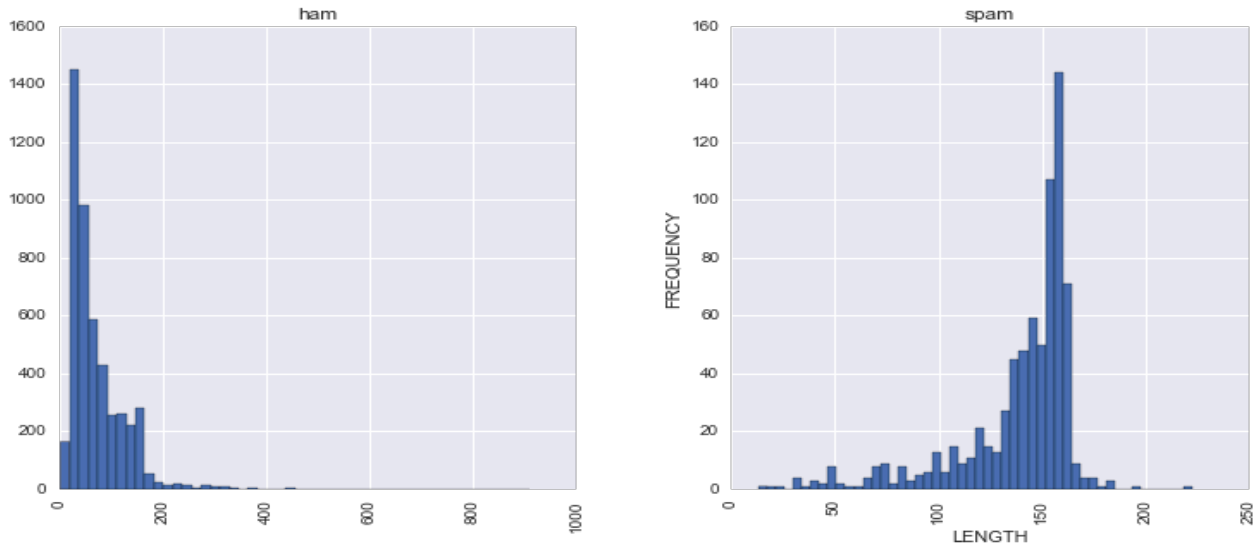


I visualized the distribution of length of text messages by using histogram

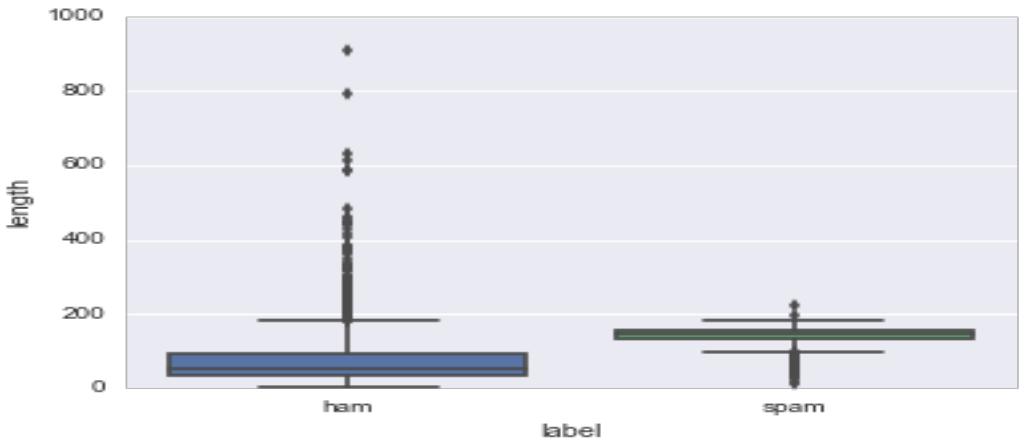


As we can see from the above histogram the length of the messages is highly skewed towards left, most of the messages are of length between 0 and 200, with few of them up to 500 in length.

Lets visualize the distribution of length of text messages across labels. The results are shown below:



Lets look at the boxplot:



From the above visualizations we've been able to discover a trend that spam messages tend to have more characters and ham messages tend to be shorter.

Algorithm and Techniques

As the dataset consist of text file where each line corresponds to a text message, therefore preprocessing of the data, extraction of features and tokenizing each message is required. There are many methods to convert a corpus to vector form. I have used the bag-of-words approach, where one number will represent each unique word in a text. First I wrote a function to remove non-letters, punctuations and stop words from the raw text messages and it will split the messages into individual words. I have used python based Natural Language Processing Toolkit (NLTK) for this.

The bag of words model learns vocabulary from all the documents, and then models each document by counting number of times the word occurs in the document. I have used the bag of words approach in 2 steps:

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

After the counting, the term weighting and normalization is done with TF-IDF, using scikit-learn's TfidfTransformer.

TF-IDF stands for term frequency-inverse document frequency, and the tf-idf weight is a weight often used in information retrieval and text mining. This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus. Variations of the tf-idf weighting scheme are often used by search engines as a central tool in scoring and ranking a document's relevance given a user query.
Typically, the tf-idf weight is composed by two terms:

- ***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:

$$TF(t) = \frac{\text{number of times term t appears in a document}}{\text{total number of terms in the document}}$$

- ***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:

$$IDF(t) = \log\_e \frac{\text{total number of documents}}{\text{number of documents with term t in it}}$$

I have used three different classifier Naïve Bayes, Random Forest and Support Vector machines separately to train the model. The performance of the classifiers is evaluated by splitting the dataset into 70% training set and 30% testing set.

- ***Naïve Bayes***
  It is a simple probabilistic classifier, which is based on Bayes theorem with strong and naïve independence assumptions. Although there are some cases where the naive Bayesian classifier does not produce good results, but is extremely simple, easy to implement, and often works very well, which makes it a desirable classifier for spam detection problems. In Bayesian classifiers, we attempt to build a probabilistic classier 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.

- ***Random Forest***
  It is an averaging ensemble method for classification. The ensemble is a combination of decision trees built from a bootstrap sample from training set. Additionally, in building the decision tree, the split that is chosen when splitting a node, is the best split only among a random set of features. This will increase the bias of a single model, but the averaging reduces the variance and can compensate for increase in bias too. Consequently, a better model is built. The main principle behind ensemble method s is that a group of 'weak learners' can come together to form a 'strong learners'. Comparing to the naive Bayes algorithm, the complexity of the model is increased.
- ***Support Vector Machine Classifiers***
  A support Vector Machine (SVM) is a discriminative classifier, which outputs an optimal hyper plane, which categorizes the two classes by a boundary. The separation boundary is achieved by the hyper plane that has the largest distance to the nearest training data points to any class. Applying SVM with different kernels increases the complexity of the model and thus the running time of training the model on data.

Benchmark

The measure of success of the classifier is to check how accurate the prediction classes are. This dataset is strongly biased towards ham (4825 ham/ 747 spam). Even if I classify every message of the dataset as ham, I can get a minimum accuracy of 87%, which is obviously as indicator of a non-ideal classifier. So, I would choose benchmark model with accuracy 87% and my goal is to get more than 94% accuracy, with more than 94% precision and recall value.

III. Methodology

Data Preprocessing

The preprocessing is done in the following steps:

1. Split the messages into its individual words.
2. Imported a list of English stop words from NLTK library. Stop words are basically a set of commonly used words in any language.
3. Created a function called 'sms_to_words', which removes the non-letters, punctuations and stop words.

   A world cloud is a collage of words and those words that are bigger in size have a high frequency. They are a very information-dense representation of the frequency of all words in a given text. Word clouds are more effective than just using bar charts displaying the counts of words for large amounts of text, as the chart would be difficult to parse if there are too many bars.

   Word clouds for our spam messages:

Word cloud for ham messages

4. Converted each message into a vector that a machine learning models can understand, which was done using the bag-of-words model in steps as:
    - 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 process can be split into two main stages:

1. The classifier's training stage - once the messages are represented as vectors, we finally train our ham/spam classifier. There are many classification algorithms, which can be used here. I started with Naïve Bayes, due to its simplicity and easy implementation. Then I continued to apply Random Forest and SVM classifier.
2. The validation stage – The best way to determine how well our trained model will perform on the entire dataset, is to split the data into training and test set. The model will be trained on the training set and the final evaluation will be done on the test data, which will be true representative of predictive performance.

With Naïve Bayes classifier, the prediction accuracy in Area Under ROC curve was about 98.6%.I got precision, recall and f1 score value of 97%, 96% and 96% respectively as shown below:

	precision	recall	f1-score	support
ham	1.00	0.96	0.98	1507
spam	0.71	1.00	0.83	165
avg/total	0.97	0.96	0.96	1672

Predicted	ham	spam
Actual		
ham	1441	0
spam	66	165

I also tried Random Forest to check if it could improve the performance and found that there was 1% increase in precision, recall and F1 score. The prediction accuracy in Area Under ROC curve of Random Forest model was 99%. The results obtained are shown below:

	precision	recall	f1-score	support
ham	1.00	0.97	0.98	1490
spam	0.79	1.00	0.88	182
avg / total	0.98	0.97	0.97	1672

Predicted	ham	spam
Actual		
ham	1441	0
spam	49	182

Finally, I tried SVM. The performance of SVM was great, with all precision, recall and f1 score as 98% . The results of SVM are shown below:

	precision	recall	f1-score	support
ham	1.00	0.98	0.99	1463
spam	0.88	0.97	0.92	209
avg / total	0.98	0.98	0.98	1672

Predicted	ham	spam
Actual		
ham	1435	6
spam	28	203

Refinement

As seen in the Benchmark section, the benchmark accuracy is 87%. I used SciKit Learn's pipeline capabilities to store a pipeline of workflow. This allows us to set up all the transformations that we will do to the data for future use.

To get the initial results, I used Naïve Bayes Classifier for multinomial models with default parameters.

```
nb_pipeline = Pipeline([
    ('CV', CountVectorizer(analyzer = sms_to_words)),  # strings to token integer counts
    ('tfidf', TfidfTransformer()),  # integer counts to weighted TF-IDF scores
    ('classifier', MultinomialNB()),  # train on TF-IDF vectors w/ Naive Bayes classifier
])
```

```
nb_pipeline.fit(sms_train,label_train)
nb_pred = nb_pipeline.predict(sms_test)
print(classification_report(nb_pred,label_test))
print(pd.crosstab(label_test,nb_pred, rownames= ['Actual'], colnames= ['Predicted']))
```

```
             precision    recall  f1-score   support

        ham       1.00      0.96      0.98      1502
       spam       0.74      1.00      0.85       170

avg / total       0.97      0.96      0.97      1672

Predicted   ham  spam
Actual
ham        1443     0
spam         59   170
```

The results were improved by using following techniques:

- Random Forest (n_estimator): n_estimators was increased to compare the performance of the classifier. I selected my final Random Forest Classifier with n_estimators =100.

RandomForest (n_estimator = 10):

```
rf_pipeline = Pipeline([
    ('CV', CountVectorizer(analyzer = sms_to_words)),  # strings to token integer counts
    ('tfidf', TfidfTransformer()),  # integer counts to weighted TF-IDF scores
    ('classifier', RandomForestClassifier(n_estimators= 10)),  # train on TF-IDF vectors w/ Random Forest classifier
])
```

```
rf_pipeline.fit(sms_train,label_train)
rf_pred = rf_pipeline.predict(sms_test)
print(classification_report(rf_pred,label_test))
print(pd.crosstab(label_test,rf_pred, rownames= ['Actual'], colnames= ['Predicted']))
```

```
             precision    recall  f1-score   support

        ham       1.00      0.97      0.99      1478
       spam       0.83      0.98      0.90       194

avg / total       0.98      0.98      0.98      1672

Predicted   ham  spam
Actual
ham        1440     3
spam         38   191
```

RandomForest (n_estimator = 60):

```
rf_pipeline = Pipeline([
    ('CV', CountVectorizer(analyzer = sms_to_words)),  # strings to token integer counts
    ('tfidf', TfidfTransformer()),  # integer counts to weighted TF-IDF scores
    ('classifier', RandomForestClassifier(n_estimators= 60)),  # train on TF-IDF vectors w/ Random Forest classifier
])
```

```
rf_pipeline.fit(sms_train,label_train)
rf_pred = rf_pipeline.predict(sms_test)
print(classification_report(rf_pred,label_test))
print(pd.crosstab(label_test,rf_pred, rownames= ['Actual'], colnames= ['Predicted']))
```

```
             precision    recall  f1-score   support

        ham       1.00      0.98      0.99      1478
       spam       0.85      1.00      0.92       194

avg / total       0.98      0.98      0.98      1672

Predicted   ham  spam
Actual
ham        1443     0
spam         35   194
```

RandomForest (n_estimator= 100):

```
rf_pipeline = Pipeline([
    ('CV', CountVectorizer(analyzer = sms_to_words)),  # strings to token integer counts
    ('tfidf', TfidfTransformer()),  # integer counts to weighted TF-IDF scores
    ('classifier', RandomForestClassifier(n_estimators= 100)),  # train on TF-IDF vectors w/ Random Forest classifier
])
```

```
rf_pipeline.fit(sms_train,label_train)
rf_pred = rf_pipeline.predict(sms_test)
print(classification_report(rf_pred,label_test))
print(pd.crosstab(label_test,rf_pred, rownames= ['Actual'], colnames= ['Predicted']))
```

```
              precision    recall  f1-score   support

         ham       1.00      0.98      0.99      1475
        spam       0.86      1.00      0.92       197

avg / total       0.98      0.98      0.98      1672

Predicted   ham  spam
Actual
ham        1443     0
spam         32   197
```

The results obtained from Random Forest and Naïve Bayes were satisfying.
But I still chose to try SVM. I tried SVM with default as well as with some parameter tuning. I tuned some of the parameters for SVC (C, gamma and kernel) and used grid search to choose the best options. The performance of SVM with parameter tuning was great but the training time was quite high compared to training time for Random Forest and Naïve Bayes, they fitted the model within seconds with performance very close to SVM's. The difference in performance of SVM with default parameters and tuned parameters could be seen below:

SVC (default)

```
from sklearn.grid_search import GridSearchCV
from sklearn.cross_validation import StratifiedKFold

svc_pipeline = Pipeline([
    ('CV', CountVectorizer(analyzer = sms_to_words)),  # strings to token integer counts
    ('tfidf', TfidfTransformer()),  # integer counts to weighted TF-IDF scores
    ('classifier', SVC()),  # train on TF-IDF vectors w/ Support Vector classifier
])
```

```
svc_pipeline.fit(sms_train,label_train)
svc_pred = svc_pipeline.predict(sms_test)
print(classification_report(svc_pred,label_test))
```

```
             precision    recall  f1-score   support

        ham       1.00      0.86      0.93      1672
       spam       0.00      0.00      0.00         0

avg / total       1.00      0.86      0.93      1672
```

SVC (Tuned parameters)

```
from sklearn.grid_search import GridSearchCV
from sklearn.cross_validation import StratifiedKFold

svc_pipeline = Pipeline([
    ('CV', CountVectorizer(analyzer = sms_to_words)),  # strings to token integer counts
    ('tfidf', TfidfTransformer()),  # integer counts to weighted TF-IDF scores
    ('classifier', SVC()),  # train on TF-IDF vectors w/ Support Vector classifier
])

# Classifier parameters
params = [
  {'classifier__C': [1, 10, 100, 1000], 'classifier__gamma': [0.001, 0.0001], 'classifier__kernel':
['rbf', 'linear']},
]

svc_grid = GridSearchCV(
    svc_pipeline,# Our pipeline from above
    param_grid= params, # The parameters we set above
    scoring='accuracy', # score we are optimizing
    cv= StratifiedKFold(label_train, n_folds=5)#type of cross fold validation to use
)
```

```
svc_pred = svc_spamdetector.predict(sms_test)
print(classification_report(svc_pred,label_test))
print(pd.crosstab(label_test,svc_pred, rownames= ['Actual'], colnames= ['Predicted']))
```

```
             precision    recall  f1-score   support

        ham       1.00      0.98      0.99      1463
       spam       0.90      0.98      0.94       209

avg / total       0.98      0.98      0.98      1672
```

Based on the performance ad efficiency of classifiers, the final model I selected is RandomForest (n_estimator = 100)

IV. Results

Model Evaluation and Validation

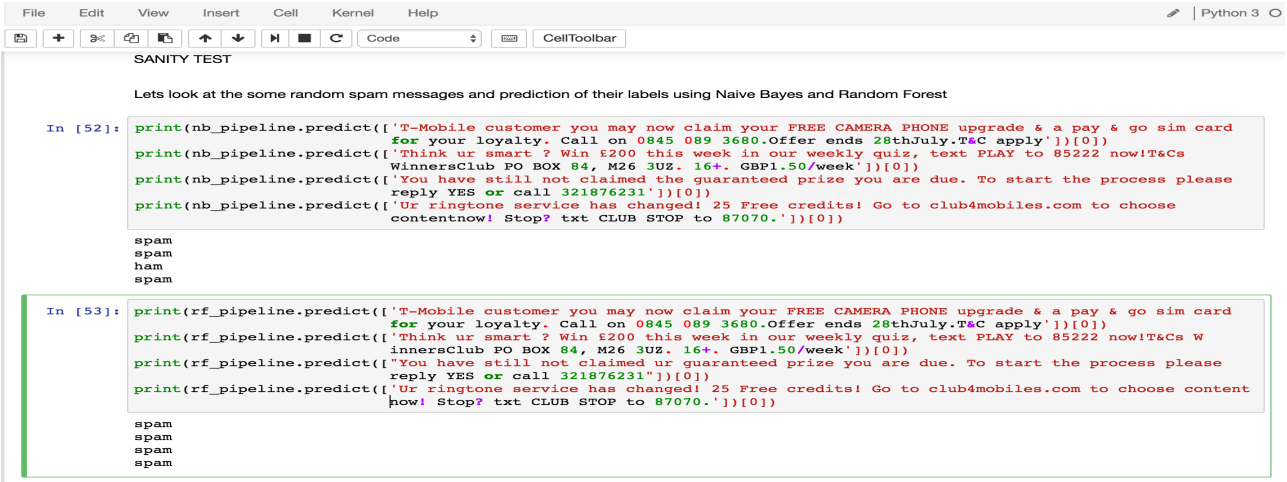
During development, a validation set was used to evaluate the model. The final algorithm and parameters were chosen because they performed the best among the tried combinations.

I have chosen Random Forest over SVM and Naïve Bayes. The performance of Random Forest is better than the Naïve Bayes. Although SVM's performance is better than both the models but the training time was more than 10 minutes, which is quite high compared to training time for Random Forest and Naïve Bayes. They fitted the model within seconds with performance very close to SVM's.

Final model selected:
Random Forest with n_estimators = 100

To verify robustness of the final model, I did sanity test using random messages from my inbox. The following observations are based on the results of the test:

Naive Bayes classifier misclassified one spam message as ham. Random Forest on the other hand, classified all the messages correctly. So, based on performance and robustness I chose Random Forest as my final classifier.



```
print(rf_pipeline.predict(['The item you placed on hold is available for pickup from the Library
nd will be kept for three library business days, not including the day of your notice'])[0])
print(rf_pipeline.predict(['Invite your friends and family. Share the Uber love and give friends
ree rides to try Uber, worth up to $20 each!'])[0])
print(rf_pipeline.predict(['Thanks for updating your DSW Rewards profile!'])[0])
print(rf_pipeline.predict(['Great news we hav already created your online My Place Rewards* accou
t for you. Click here to confirm the below email address and reset your password.'])[0])
```

```
ham
ham
ham
ham
```

The classifiers have performed better than expected, with accuracy in Area Under ROC curve as 99% with a reasonable classification time, which is much better than the benchmark model. As seen from the above examples of hams and spams, the classifier can reliably classify spam and hams. So, based on performance and robustness I chose Random Forest as my final classifier.

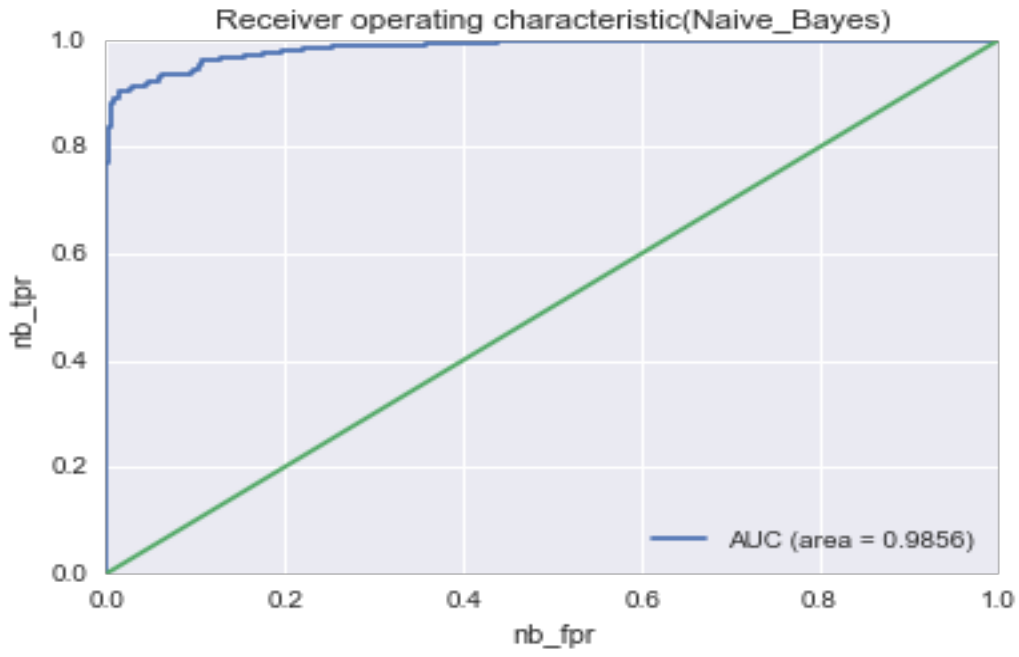
Justification

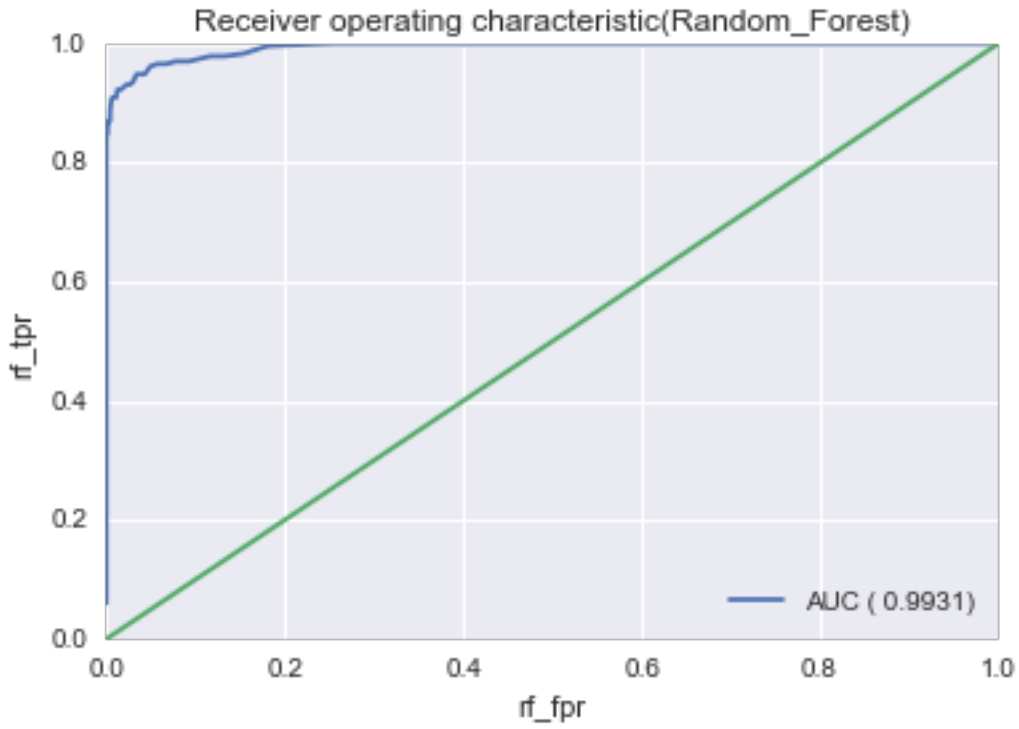
The classifiers have performed better than expected, with accuracy in Area Under ROC curve as 99% with a reasonable classification time, which is much better than the benchmark model.

IV. Conclusion

Free-Form Visualization

I had used ROC AUC score, which is a measure of how efficient the model is to rank the probability to belong to each class. The area under the curve is a measure of text accuracy. It is a plot of the true positive rate against the false positive rate for the different possible thresholds of a test. The ROC curves for Naïve Bayes and Random Forest model are shown below:





Receiver operating characteristic (ROC) curves demonstrates several things:

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

The AUC (Area under Curve) for Naïve Bayes I got is 0.9856

The AUC (Area under Curve) for Random Forest I got is 0.9931

To check robustness of the final model, I did sanity test using random spam and ham messages. On checking the performance of models in classifying spams vs. hams Random Forest is able to correctly classify all the spam and ham random messages while Naïve Bayes misclassified one spam as ham.

Reflection

The process for this project can be summarized in following steps:

1. Download the dataset from UCI datasets.
2. Exploratory Data Analysis- Once the dataset is ready; some basic exploratory data analysis is done to get better understanding of the data.

3. Text Preprocessing- is done to convert the raw text messages into tokens, which includes splitting sentence to words, removing stop words, normalization and more. For this step I used NLTK library, which is a standard library in Python to process text and has many useful features.
4. Vectorization- the tokens were then converted to vectors that a machine- learning algorithm can understand. I used bag of words model, which learns vocabulary from all the documents, then models each document by counting the number of times each word appears.
5. Splitting the dataset into training and test sets
6. Training a model- once messages are represented as vectors, I trained the classifier. There are many classification algorithms, which can be used here. I used Naïve Bayes, Random Forest and SVM.
7. Evaluation of model-. The model was trained on the training set and the final evaluation is done on the test data, which will be true representative of predictive performance.

This is my first project on NLP, I enjoyed working on it especially steps 3 and 4 above were of more interest. It was bit challenging for me to familiarize myself with various NLTK tools and library.

Improvement

There might be few things that could have been improved:

- Fine tuning of Naïve Bayes and Random Forest.
- More real time text messages could be collected to check model's performance.
- More sophisticated deep learning neural networks could be applied to train the model

References

https://github.com/udacity/machine-learning/blob/master/projects/capstone/capstone_report_template.md
https://github.com/udacity/machine-learning/blob/master/projects/capstone/report-example-1.pdf
http://gim.unmc.edu/dxtests/ROC2.htm
http://scikit-learn.org/stable/modules/pipeline.html
https://en.wikipedia.org/wiki/Natural_language_processing
https://www.kaggle.com/c/word2vec-nlp-tutorial/details/part-1-for-beginners-bag-of-words
http://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html
https://github.com/udacity/machine-learning/blob/master/projects/capstone/capstone_proposal_template.md
http://www.nltk.org/book/
https://archive.ics.uci.edu/ml/datasets/SMS+Spam+Collection
http://cs229.stanford.edu/proj2013/ShiraniMehr-SMSSpamDetectionUsingMachineLearningApproach.pd