SMS Spam Detection using NLP

Jai Soni
soni.j@husky.neu.edu
CSYE 7245, Spring-18, Northeastern University

Abstract:

Increasing number of mobile phones has also led to increased SMS or Short Message Services relating to market offers, advertisements, job alerts etc. This are called Spam messages as most of them are junk and not very useful to users. Besides creating a mess in the phone, these messages also take unnecessary memory space overtime. Filtering SMS based on message details is similar to Email spam filtering. The purpose of this project is to use machine learning algorithms and NLP techniques to filter spam SMS with realistic accuracy.

Introduction:

Short Message Service (SMS) is a text messaging service component of most telephone, World Wide Web, and mobile device systems. It uses standardized communication protocols to enable mobile devices to exchange short text messages. An intermediary service can facilitate a text-to-voice conversion to be sent to landlines. SMS was the most widely used data application, with an estimated 3.5 billion active users, or about 80% of all mobile subscribers, at the end of 2010.

In 2010, 6.1 trillion (6.1×10^{12}) SMS text messages were sent. This translates into an average of 193,000 SMS per second. SMS has become a huge commercial industry, earning $114.6 billion globally in 2010. The global average price for an SMS message is US$0.11, while mobile networks charge each other interconnect fees of at least US$0.04 when connecting between different phone networks.

In 2015, the actual cost of sending an SMS in Australia was found to be $0.00016 per SMS.

Apparently, SMS spam is not as cost-prohibitive to spammers as it used to be, as the popularity of SMS has led to messaging charges dropping below US$ 0.001 in markets like China, and even free of charge in others. According to Cloudmark stats, the amount of mobile phone spam varies widely from region to region. For instance, in North America, much less than 1% of SMS messages were spam in 2010, while in parts of Asia up to 30% of messages were represented by spam. The main problem with SMS spam is that it is not only annoying, but it can also be expensive since some people pay to receive text messages. Moreover, there is a limited availability of mobile phone spam-filtering software. Other concern is that important legitimate messages as of emergency nature could be blocked. Nonetheless, many providers offer their subscribers means for mitigating unsolicited SMS messages

Dataset:

The Datasets in this project is taken from UC Irvine Machine Learning Repository:
https://archive.ics.uci.edu/ml/datasets/sms+spam+collection

Data Analysis:

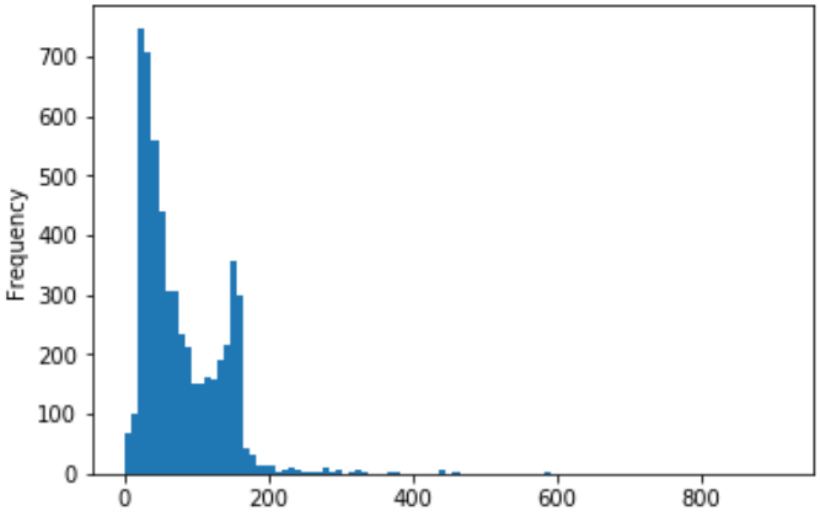
The data consist of Labels and Messages, where non spam messages are referred as ham. The data looks like:

	label	message
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

The data consists of 5572 entries with the following distribution:

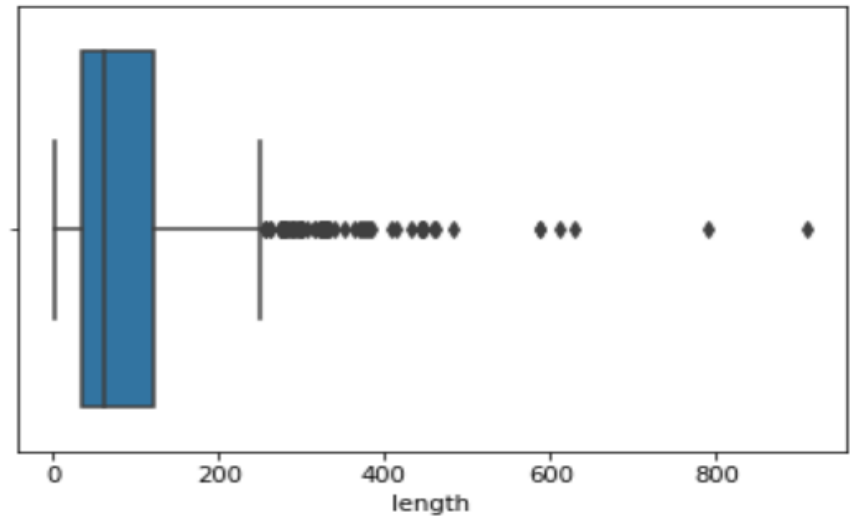
	message			
	count	unique	top	freq
label				
ham	4825	4516	Sorry, I'll call later	30
spam	747	653	Please call our customer service representativ...	4

Distribution of data based on length of messages(Number of characters in a message):



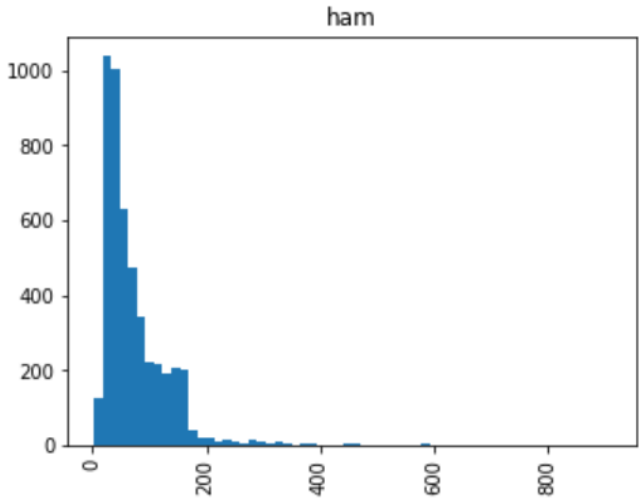
The length of the messages are mostly within sub 200 characters, which seems correct as the max length of a single regular SMS is 160 characters. There are few messages with higher lengths.

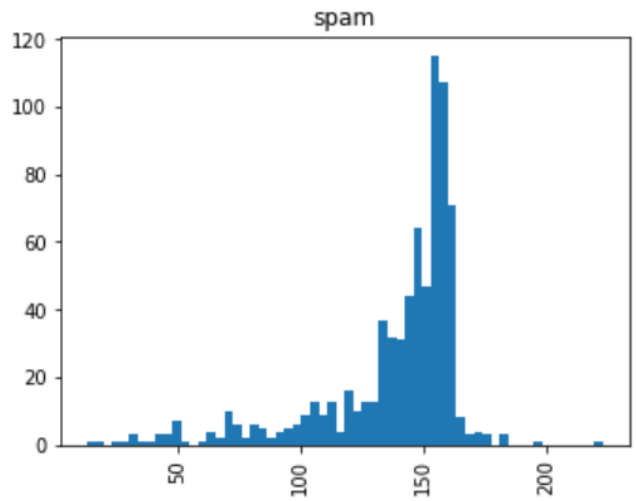
Checking the outliers in the data:



From the above two plots, we can clearly see that most of the length of the messages are around 75-150 characters zone, we also see few outliers around 800 and 900 lengths

Let's look at the distribution of length of messages based on ham and spam:





We can figure out that based on lengths of the messages, the mean length of ham messages is around 80 whereas the mean length is around 150 for spam messages.

After removing the outliers, a logistic regression model is trained based on lengths of ham and spam. The process for predicting values is shown below:

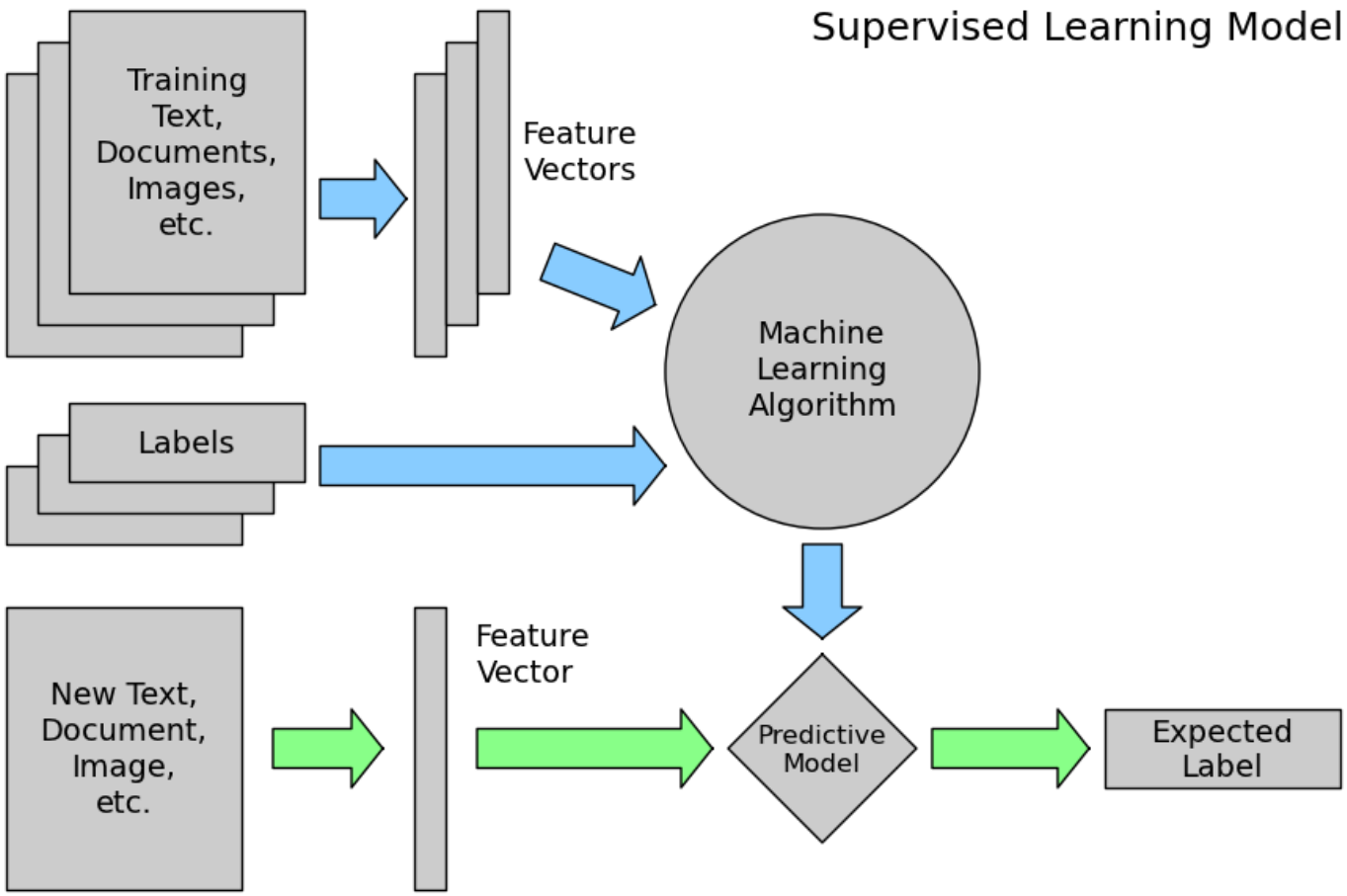
The data id cleaned, outliers are removed, and training and testing sets are created with a training size of 30%. Logistic Regression model is chosen from Sci-kit Learn library. The model is trained using the training data, and predictions are made on the testing data.

Performance Evaluation for a Logistic Regression model:

```
Confusion Matrix :

[[1560   33]
 [ 246    0]]

Classification Report :

             precision    recall  f1-score   support

        ham       0.86      0.98      0.92      1593
       spam       0.00      0.00      0.00       246

avg / total       0.75      0.85      0.80      1839
```

We get a precision of 75% for just training the model with length. To improve the accuracy, we have to take a deeper insight into the data and use Natural Language processing techniques to clean and prepare the data, and use advanced supervised machine learning algorithms like Naïve Bayes, support vector machine etc. We will be using Naïve Bayes to do the same.

The Process:



Data Pre-Processing:

We'll map the raw messages (sequence of characters) into vectors (sequences of numbers).

The mapping is not 1-to-1; we'll use the bag-of-words approach, where each unique word in a text will be represented by one number.

As a first step, we will remove stopwords and punctuations from our messages and return a clean list of individual words. This process is referred to as Tokenization. We will create a function to do the same:

```
#Creating a function for the above shown process so that we can apply it in our messages
def text_process(mess):
    """
    1. remove punc
    2. remove stop words
    3. return list of clean text words
    """
    nopunc = [char for char in mess if char not in string.punctuation]

    nopunc = ''.join(nopunc)

    return [word for word in nopunc.split() if word.lower() not in stopwords.words('english')]
```

```
#Applying Tokenization
cleaned_messages = messages['message'].head(5).apply(text_process)
```

```
cleaned_messages
```

```
0    [Go, jurong, point, crazy, Available, bugis, n...
1                        [Ok, lar, Joking, wif, u, oni]
2    [Free, entry, 2, wkly, comp, win, FA, Cup, fin...
3        [U, dun, say, early, hor, U, c, already, say]
4    [Nah, dont, think, goes, usf, lives, around, t...
Name: message, dtype: object
```

Now we'll convert each message, represented as a list of tokens (lemmas) above, into a vector that machine learning models can understand.

Doing that requires essentially three steps, in the bag-of-words model:

Counting how many times does a word occur in each message (term frequency) weighting the counts, so that frequent tokens get lower weight (inverse document frequency) normalizing the vectors to unit length, to abstract from the original text length (L2 norm) Each vector has as many dimensions as there are unique words in the SMS corpus:

```
# Applying Count Vectorizer

from sklearn.feature_extraction.text import CountVectorizer
```

```
# bow-> Bag of Words
bow_transformer = CountVectorizer(analyzer=text_process).fit(messages['message'])
```

```
messages_bow = bow_transformer.transform(messages['message'])
```

```
messages_bow = bow_transformer.transform(messages['message'])
print('sparse matrix shape:', messages_bow.shape)
print('number of non-zeros:', messages_bow.nnz)
print('sparsity: %.2f%%' % (100.0 * messages_bow.nnz / (messages_bow.shape[0] * messages_bow.shape[1])))
```

```
sparse matrix shape: (5572, 11425)
number of non-zeros: 50548
sparsity: 0.08%
```

And finally, after the counting, the term weighting and normalization can be done with TF-IDF, using Sci-kit learns Tfidf Transformer:

```
from sklearn.feature_extraction.text import TfidfTransformer
```

```
tfidf_transformer = TfidfTransformer().fit(messages_bow)
```

Training a model:

Now we can divide the data again into training and testing set with a 30% training data, and train a Multinomial Naïve Bayes model. We used Multinomial NB as this algorithm is pretty effective in document classification.

We have created a data pipeline to apply the vectorization, Tokenization process re-usable for the training and testing data

```
from sklearn.pipeline import Pipeline
```

```
pipeline = Pipeline([
    ('bow',CountVectorizer(analyzer=text_process)),# 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
])
```

```
pipeline.fit(msg_train,label_train)
```

Prediction and Evaluation:

```
predictions = pipeline.predict(msg_test)
```

```
print('Confusion_Matrix :\n\n',confusion_matrix(label_test,predictions))
print('\n')
print('Classification_report :\n\n',classification_report(label_test,predictions))
```

```
Confusion_Matrix :

 [[1437    0]
 [  69  166]]


Classification_report :

              precision    recall  f1-score   support

         ham       0.95      1.00      0.98      1437
        spam       1.00      0.71      0.83       235

avg / total       0.96      0.96      0.96      1672
```

We get a really high precision of 96% which is amazing and should work well for most scenarios.

Can we use this model for predicting any given random messages? Let's try:

```
print(pipeline.predict(['How are you now?'])[0])
print(pipeline.predict(['URGENT!!! Your Mobile No was awarded a 2,000 $'])[0])
```

```
ham
spam
```

The answer for the above question is answered well by the figure above!

Conclusion:

We get a very high accuracy of prediction when we use classification algorithms like Naïve Bayes. The predictions will work better if trained with more variety of spam and ham messages. There is another problem to look at, that if Spam messages contain abbreviations like ham message such as 'U' for 'you' etc. This may lead to a conflict for classifying messages, and hence further work is required for data pre-processing, to achieve an even better prediction accuracy.

References:

[1] http://www.dt.fee.unicamp.br/~tiago/smsspamcollection/

[2] http://www.dt.fee.unicamp.br/~tiago/smsspamcollection/doceng11.pdf

[3] https://en.wikipedia.org/wiki/SMS

[4] https://en.wikipedia.org/wiki/Sparse_matrix

[5] https://radimrehurek.com/data_science_python/

[6] https://github.com/nikbearbrown/NEU_COE/blob/master/CSYE_7245/Week_7/NBB_NLTK.ipynb

[7] https://sebastianraschka.com/faq/docs/bag-of-words-sparsity.md