NAME OF THE PROJECT

EMAIL SPAM CLASSIFIER PROJECT

Submitted by:

MR. SHIVAM CHAUDHARY

FLIPROBO SME:

MS. GULSHANA CHAUDHARY

ACKNOWLEDGMENT

I would like to express my special gratitude to "Flip Robo" team, who has given me this opportunity to deal with a beautiful dataset and it has helped me to improve my analyzation skills. And I want to express my huge gratitude to Ms. Gulshana Chaudhary (SME Flip Robo), she is the person who has helped me to get out of all the difficulties I faced while doing the project.

A huge thanks to "Data trained" who are the reason behind my Internship at Fliprobo. Last but not least my parents who have been my backbone in every step of my life.

References use in this project:

1. SCIKIT Learn Library Documentation
2. Blogs from towardsdatascience, Analytics Vidya, Medium
3. Andrew Ng Notes on Machine Learning (GitHub)
4. Data Science Projects with Python Second Edition by Packt
5. Hands on Machine learning with scikit learn and tensor flow by Aurelien Geron
6. Pudaruth, S. (2014). Predicting the price of used cars using machine learning techniques. Int. J. Inf. Comput. Technol, 4(7), 753-764 – [1]
7. Agencija za statistiku BiH. (n.d.), retrieved from: http://www.bhas.ba . [accessed July 18, 2018.] – [2]
8. Monburinon, Nitis, Prajak Chertchom, Thongchai Kaewkiriya, Suwat Rungpheung, Sabir Buya, and Pitchayakit Boonpou. "Prediction of prices for used car by using regression models." In 2018 5th

International Conference on Business and Industrial Research (ICBIR), pp. 115-119. IEEE, 2018. – [3]

9. Gegic, Enis, Becir Isakovic, Dino Keco, Zerina Masetic, and Jasmin Kevric. "Car price prediction using machine learning techniques." TEM Journal 8, no. 1 (2019): 113. – [4]
10. Noor, Kanwal, and Sadaqat Jan. "Vehicle price prediction system using machine learning techniques." International Journal of Computer Applications 167, no. 9 (2017): 27-31. – [5]
11. Kuiper, S. 2008. "Introduction to Multiple Regression: How Much Is Your Car Worth?", Journal of Statistics Education, 16(3). – [6]
12. Gongqi, S., Yansong, W., & Qiang, Z. (2011, January). New Model for Residual Value Prediction of the Used Car Based on BP Neural Network and Nonlinear Curve Fit. In Measuring Technology and Mechatronics Automation (ICMTMA), 2011 Third International Conference on (Vol. 2, pp. 682-685). IEEE.

Chap 1. Introduction

1.1 Business Problem Framing

Most of us consider spam emails as one which is annoying and repetitively used for purpose of advertisement and brand promotion. We keep on blocking such email-ids but it is of no use as spam emails are still prevalent. Some major categories of spam emails that are causing great risk to security, such as fraudulent e-mails, identify theft, hacking, viruses, and malware. In order to deal with spam emails, we need to build a robust real-time email spam classifier that can efficiently and correctly flag the incoming mail spam, if it is a spam message or looks like a spam message. The latter will further help to build an Anti-Spam Filter.

Google and other email services are providing utility for flagging email spam but are still in the infancy stage and need regular feedback from the end-user. Also, popular email services such as Gmail, Yandex, yahoo mail, etc provide basic services as free to the end-user and that of course comes with EULA. There is a great scope in building email spam classifiers, as the private companies run their own email servers and want them to be more secure because of the confidential data, in such cases email spam classifier solutions can be provided to such companies.

1.2 Conceptual Background of the Domain Problem

Today, spam has become a big internet issues. Recent 2017, the statistic shown spam accounted for 55% of all e-mail messages, same as during the previous year. Spam which is also known as unsolicited bulk email has

led to the increasing use of email as email provides the perfect ways to send the unwanted advertisement or junk newsgroup posting at no cost for the sender. This chances has been extensively exploited by irresponsible organizations and resulting to clutter the mail boxes of millions of people all around the world.

Evolving from a minor to major concern, given the high offensive content of messages, spam is a waste of time. It also consumed a lot of storage space and communication bandwidth. End user is at risk of deleting legitimate mail by mistake. Moreover, spam also impacted the economical which led some countries to adopt legislation. 2

Text classification is used to determine the path of incoming mail/message either into inbox or straight to spam folder. It is the process of assigning categories to text according to its content. It is used to organized, structures and categorize text. It can be done either manually or automatically. Machine learning automatically classifies the text in a much faster way than manual technique. Machine learning uses pre-labelled text to learn the different associations between pieces of text and it output. It used feature extraction to transform each text to numerical representation in form of vector which represents the frequency of word in predefined dictionary.

Text classification is important in the context of structuring the unstructured and messy nature of text such as documents and spam messages in a cost-effective way. A Machine learning platform has capabilities to improve the accuracy of predictions. With regard to Big Data, a Machine Learning platform has abilities to speed up analysing of

gigantic data. It is important especially to a company to analyse text data, help inform business decisions and even automate business processes.

For example, text classification is used in classifying short texts such as tweets or headlines. It can be used in larger documents such as media articles. It also can be applied to social media monitoring, brand monitoring and etc.

In this project, a machine learning technique is used to detect the spam message of a mail. Machine learning is where computers can learn to do something without the need to explicitly program them for the task. It uses data and produce a program to perform a task such as classification. Compared to knowledge engineering, machine learning techniques require messages that have been successfully pre-classified. The pre-classified messages make the training dataset which will be used to fit the learning algorithm to the model in machine learning studio. 3

A specific algorithm is used to learn the classification rules from these messages. Those algorithms are used for classification of objects of different classes. The algorithms are provided with input and output data and have a self-learning program to solve the given task. Searching for the best algorithm and model can be time consuming. The two-class classifier is best used to classify the type of message either spam or ham. This algorithm is used to predict the probability and classification of data outcome.

1.3 Review of Literature

This chapter discusses about the literature review for machine learning classifier that being used in previous researches and projects. It is not about information gathering but it summarizes the prior research that related to this project. It involves the process of searching, reading, analysing, summarising and evaluating the reading materials based on the project.

Literature reviews on machine learning topic have shown that most spam filtering and detection techniques need to be trained and updated from time to time. Rules also need to be set for spam filtering to start working. So eventually it become burdensome to the user.

1.4 Motivation for the Problem Undertaken

The project was the first provided to me by Flip Robo Technologies as a part of the internship programme. The exposure to real world data and the opportunity to deploy my skillset in solving a real time problem has been the primary motivation.

Data needed for this project is provided by Flip Robo Technologies as a part of the internship programme.

Chap 2 Analytical Problem Framing

1. Mathematical / Analytical Modelling of the Problem

Several machine learning algorithms have been used in spam e-mail filtering, but Naıve Bayes algorithm is particularly popular in commercial and open-source spam filters [2]. This is because of its simplicity, which make them easy to implement and just need short training time or fast evaluation to filter email spam 2. Data Sources and their formats

2. Data Sources and their formats

Data is provided by provided by Flip Robo Technologies as a part of the internship programme.

	label	sms_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...

Initially the dataset has 5 columns out of which 3 contains only null values

So, I drop them and the remaining columns are above. Name of these two columns are renamed by me.

3. Data Pre-processing

Now that we have a basic understanding of what our dataset looks like, lets convert our labels to binary variables, 0 to represent 'ham'(i.e. not spam) and 1 to represent 'spam' for ease of computation.

You might be wondering why do we need to do this step? The answer to this lies in how scikit-learn handles inputs. Scikit-learn only deals with numerical values and hence if we were to leave our label values as strings, scikit-learn would do the conversion internally(more specifically, the string labels will be cast to unknown float values).

Our model would still be able to make predictions if we left our labels as strings but we could have issues later when calculating performance metrics, for example when calculating our precision and recall scores. Hence, to avoid unexpected 'gotchas' later, it is good practice to have our categorical values be fed into our model as integers.

**Instructions: *

Convert the values in the 'label' column to numerical values using map method as follows: {'ham':0, 'spam':1} This maps the 'ham' value to 0 and the 'spam' value to 1.

Also, to get an idea of the size of the dataset we are dealing with, print out number of rows and columns using 'shape'.

```
df['label'] = df.label.map({'ham':0, 'spam':1})
print(df.shape)
df.head() # returns (rows, columns)
```

```
(5572, 2)
```

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

Bag of words

What we have here in our data set is a large collection of text data (5,572 rows of data). Most ML algorithms rely on numerical data to be fed into them as input, and email/sms messages are usually text heavy.

Here we'd like to introduce the Bag of Words(BoW) concept which is a term used to specify the problems that have a 'bag of words' or a collection of text data that needs to be worked with. The basic idea of BoW is to take a piece of text and count the frequency of the words in that text. It is important to note that the BoW concept treats each word individually and the order in which the words occur does not matter.

Using a process which we will go through now, we can convert a collection of documents to a matrix, with each document being a row and each word(token) being the column, and the corresponding (row,column) values being the frequency of occurrence of each word or token in that document.

- Convert all the strings in the documents set to their lower case. Save them into a list called 'lower_case_documents'. You can convert strings to their lower case in python by using the lower() method.

```
'''
Solution:
'''
documents = ['Hello, how are you!',
             'Win money, win from home.',
             'Call me now.',
             'Hello, Call hello you tomorrow?']

lower_case_documents = []
for i in documents:
    lower_case_documents.append(i.lower())
print(lower_case_documents)
```

```
['hello, how are you!', 'win money, win from home.', 'call me now.', 'hello, call hello you tomorrow?']
```

> *Instructions:* * Remove all punctuation from the strings in the document set. Save them into a list called 'sans_punctuation_documents'.

```
'''
Solution:
'''
sans_punctuation_documents = []
import string

for i in lower_case_documents:
    sans_punctuation_documents.append(i.translate(str.maketrans('','',string.punctuation)))

print(sans_punctuation_documents)
```

```
['hello how are you', 'win money win from home', 'call me now', 'hello call hello you tomorrow']
```

Step 3.3: Implementing Bag of Words in scikit-learn¶

Now that we have implemented the BoW concept from scratch, let's go ahead and use scikit-learn to do this process in a clean and succinct way. We will use the same document set as we used in the previous step.

** Data preprocessing with CountVectorizer() **

In Step 2.2, we implemented a version of the CountVectorizer() method from scratch that entailed cleaning our data first. This cleaning involved converting all of our data to lower case and removing all punctuation marks. CountVectorizer() has certain parameters which take care of these steps for us. They are:

`lowercase = True`

The `lowercase` parameter has a default value of `True` which converts all of our text to its lower case form.

`token_pattern = (?u)\\b\\w\\w+\\b`

The `token_pattern` parameter has a default regular expression value of `(?u)\\b\\w\\w+\\b` which ignores all punctuation marks and treats them as delimiters, while accepting alphanumeric strings of length greater than or equal to 2, as individual tokens or words.

`stop_words`

The `stop_words` parameter, if set to `english` will remove all words from our document set that match a list of English stop words which is defined in scikit-learn. Considering the size of our dataset and the fact that we are dealing with SMS messages and not larger text sources like e-mail, we will not be setting this parameter value.

You can take a look at all the parameter values of your `count_vector` object by simply printing out the object as follows:

```
count_vector.fit(documents)
count_vector.get_feature_names()
```

```
C:\Users\ADMIN\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function get_feature_names is deprec
ated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
```

```
['are',
 'call',
 'from',
 'hello',
 'home',
 'how',
 'me',
 'money',
 'now',
 'tomorrow',
 'win',
 'you']
```

Now we have a clean representation of the documents in terms of the frequency distribution of the words in them. To make it easier to understand our next step is to convert this array into a dataframe and name the columns appropriately.

Instructions: Convert the array we obtained, loaded into 'doc_array', into a dataframe and set the column names to the word names(which you computed earlier using get_feature_names(). Call the dataframe 'frequency_matrix'.

```
frequency_matrix = pd.DataFrame(doc_array, columns=count_vector.get_feature_names())
frequency_matrix
```

	are	call	from	hello	home	how	me	money	now	tomorrow	win	you
0	1	0	0	1	0	1	0	0	0	0	0	1
1	0	0	1	0	1	0	0	1	0	0	2	0
2	0	1	0	0	0	0	1	0	1	0	0	0
3	0	1	0	2	0	0	0	0	0	1	0	1

You have successfully implemented a Bag of Words problem for a document dataset that we created.

One potential issue that can arise from using this method out of the box is the fact that if our dataset of text is extremely large(say if we have a large collection of news articles or email data), there will be certain values that are more common that others simply due to the structure of the language itself. So for example words like 'is', 'the', 'an', pronouns, grammatical constructs etc could skew our matrix and affect our analyis.

There are a couple of ways to mitigate this. One way is to use the `stop_words` parameter and set its value to `english`. This will automatically ignore all words(from our input text) that are found in a built in list of English stop words in scikit-learn.

Another way of mitigating this is by using the tfidf method. This method is out of scope for the context of this lesson.

Step 4.1: Training and testing sets

Now that we have understood how to deal with the Bag of Words problem we can get back to our dataset and proceed with our analysis. Our first step in this regard would be to split our dataset into a training and testing set so we can test our model later.

Instructions: Split the dataset into a training and testing set by using the train_test_split method in sklearn. Split the data using the following variables:

`X_train` is our training data for the 'sms_message' column.

y_train is our training data for the 'label' column

X_test is our testing data for the 'sms_message' column.

y_test is our testing data for the 'label' column Print out the number of rows we have in each our training and testing data.

```
# split into training and testing sets
# USE from sklearn.model_selection import train_test_split to avoid seeing deprecation warning.
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(df['sms_message'],
                                                    df['label'],
                                                    random_state=1)

print('Number of rows in the total set: {}'.format(df.shape[0]))
print('Number of rows in the training set: {}'.format(X_train.shape[0]))
print('Number of rows in the test set: {}'.format(X_test.shape[0]))
```

```
Number of rows in the total set: 5572
Number of rows in the training set: 4179
Number of rows in the test set: 1393
```

Step 4.2: Applying Bag of Words processing to our dataset.

Now that we have split the data, our next objective is to follow the steps from Step 2: Bag of words and convert our data into the desired matrix format. To do this we will be using CountVectorizer() as we did before. There are two steps to consider here:

Firstly, we have to fit our training data (X_train) into CountVectorizer() and return the matrix.

Secondly, we have to transform our testing data (X_test) to return the matrix.

Note that X_train is our training data for the 'sms_message' column in our dataset and we will be using this to train our model.

X_test is our testing data for the 'sms_message' column and this is the data we will be using(after transformation to a matrix) to make predictions on. We will then compare those predictions with y_test in a later step.

For now, we have provided the code that does the matrix transformations for you!

Step 5.1: Bayes Theorem implementation from scratch¶

Now that we have our dataset in the format that we need, we can move onto the next portion of our mission which is the algorithm we will use to make our predictions to classify a message as spam or not spam. Remember that at the start of the mission we briefly discussed the Bayes theorem but now we shall go into a little more detail. In layman's terms, the Bayes theorem calculates the probability of an event occurring, based on certain other probabilities that are related to the event in question. It is composed of a prior(the probabilities that we are aware of or that is given to us) and the posterior(the probabilities we are looking to compute using the priors).

Let us implement the Bayes Theorem from scratch using a simple example. Let's say we are trying to find the odds of an individual having diabetes, given that he or she was tested for it and got a positive result. In the medical field, such probabilies play a very important role as it usually deals with life and death situations.

We assume the following:

P(D) is the probability of a person having Diabetes. It's value is 0.01 or in other words, 1% of the general population has diabetes(Disclaimer: these values are assumptions and are not reflective of any medical study).

P(Pos) is the probability of getting a positive test result.

P(Neg) is the probability of getting a negative test result.

P(Pos|D) is the probability of getting a positive result on a test done for detecting diabetes, given that you have diabetes. This has a value 0.9. In other words the test is correct 90% of the time. This is also called the Sensitivity or True Positive Rate.

P(Neg|~D) is the probability of getting a negative result on a test done for detecting diabetes, given that you do not have diabetes. This also has a value of 0.9 and is therefore correct, 90% of the time. This is also called the Specificity or True Negative Rate.

Step 5.2: Naive Bayes implementation from scratch

Now that you have understood the ins and outs of Bayes Theorem, we will extend it to consider cases where we have more than feature.

Let's say that we have two political parties' candidates, 'Jill Stein' of the Green Party and 'Gary Johnson' of the Libertarian Party and we have the probabilities of each of these candidates saying the words 'freedom', 'immigration' and 'environment' when they give a speech:

Probability that Jill Stein says 'freedom': 0.1 ---------> P(F|J)

Probability that Jill Stein says 'immigration': 0.1 -----> P(I|J)

Probability that Jill Stein says 'environment': 0.8 -----> P(E|J)

Probability that Gary Johnson says 'freedom': 0.7 -------> P(F|G)

Probability that Gary Johnson says 'immigration': 0.2 ---> P(I|G)

Probability that Gary Johnson says 'environment': 0.1 ---> P(E|G)

And let us also assume that the probability of Jill Stein giving a speech, P(J) is 0.5 and the same for Gary Johnson, P(G) = 0.5.

Given this, what if we had to find the probabilities of Jill Stein saying the words 'freedom' and 'immigration'? This is where the Naive Bayes'theorem comes into play as we are considering two features, 'freedom' and 'immigration'.

$$P(y \mid x_1, \ldots, x_n) = \frac{P(y)P(x_1, \ldots x_n \mid y)}{P(x_1, \ldots, x_n)}$$

Step 6: Naive Bayes implementation using scikit-learn¶

Thankfully, sklearn has several Naive Bayes implementations that we can use and so we do not have to do the math from scratch. We will be using sklearns sklearn.naive_bayes method to make predictions on our dataset.

Specifically, we will be using the multinomial Naive Bayes implementation. This particular classifier is suitable for classification with discrete features

(such as in our case, word counts for text classification). It takes in integer word counts as its input. On the other hand Gaussian Naive Bayes is better suited for continuous data as it assumes that the input data has a Gaussian(normal) distribution.

```
from sklearn.naive_bayes import MultinomialNB
naive_bayes = MultinomialNB()
naive_bayes.fit(training_data, y_train)
```

```
MultinomialNB()
```

Step 7: Evaluating our model¶

Now that we have made predictions on our test set, our next goal is to evaluate how well our model is doing. There are various mechanisms for doing so, but first let's do quick recap of them.

** Accuracy ** measures how often the classifier makes the correct prediction. It’s the ratio of the number of correct predictions to the total number of predictions (the number of test data points).

** Precision ** tells us what proportion of messages we classified as spam, actually were spam. It is a ratio of true positives(words classified as spam, and which are actually spam) to all positives(all words classified as spam, irrespective of whether that was the correct classification), in other words it is the ratio of

[True Positives/(True Positives + False Positives)]

** Recall(sensitivity)** tells us what proportion of messages that actually were spam were classified by us as spam. It is a ratio of true positives(words classified as spam, and which are actually spam) to all the words that were actually spam, in other words it is the ratio of

[True Positives/(True Positives + False Negatives)]

For classification problems that are skewed in their classification distributions like in our case, for example if we had a 100 text messages and only 2 were spam and the rest 98 weren't, accuracy by itself is not a very good metric. We could classify 90 messages as not spam(including the 2 that were spam but we classify them as not spam, hence they would be false negatives) and 10 as spam(all 10 false positives) and still get a reasonably good accuracy score. For such cases, precision and recall come in very handy. These two metrics can be combined to get the F1 score, which is weighted average of the precision and recall scores. This score can range from 0 to 1, with 1 being the best possible F1 score.

We will be using all 4 metrics to make sure our model does well. For all 4 metrics whose values can range from 0 to 1, having a score as close to 1 as possible is a good indicator of how well our model is doing.

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
print('Accuracy score: ', format(accuracy_score(y_test, predictions)))
print('Precision score: ', format(precision_score(y_test, predictions)))
print('Recall score: ', format(recall_score(y_test, predictions)))
print('F1 score: ', format(f1_score(y_test, predictions)))
```

```
Accuracy score:  0.9856424982053122
Precision score:  0.9545454545454546
Recall score:  0.9333333333333333
F1 score:  0.9438202247191012
```

Step 7: Conclusion¶

One of the major advantages that Naive Bayes has over other classification algorithms is its ability to handle an extremely large number of features. In our case, each word is treated as a feature and there are thousands of different words. Also, it performs well even with the presence of irrelevant features and is relatively unaffected by them. The other major advantage it has is its relative simplicity. Naive Bayes' works well right out of the box and tuning it's parameters is rarely ever necessary, except usually in cases where the distribution of the data is known. It rarely ever overfits the data. Another important advantage is that its model training and prediction times are very fast for the amount of data it can handle. All in all, Naive Bayes' really is a gem of an algorithm!

Congratulations! You have successfully designed a model that can efficiently predict if an SMS message is spam or not!

Hardware & Software Requirements with Tool Used

Hardware Used -

1. Processor — Intel i3 processor with 2.4GHZ

2. RAM — 4 GB

3. GPU — 2GB AMD Radeon Graphics card

Software utilised -

1.Anaconda – Jupyter Notebook

2. Selenium - Webscraping

3. Google Colab – for Hyper parameter tuning

Limitations of this work and Scope for Future Work

Based on the result of this project, only text (messages) can be classified and score instead of domain name and email address. This project only focus on filtering, analysing and classifying message and do not blocking them.