FLIP ROBO

EMAIL SPAM CLASSIFIER PROJECT

Submitted by:

CHETHANA M

ACKNOWLEDGMENT

I express.my sincere gratitude to **Flip Robo Technologies** for giving me this opportunity to carry out the project work.

A special thanks to my mentor **Mohd Kashif** for guiding me in completing this project and being available to resolve my doubts whenever I raise any tickets.

I also would love to take this moment to thank **DataTrained** for giving me all the knowledge about how to build an effective machine learning model and providing this opportunity to work as intern in Flip Robo Technologies.

INTRODUCTION

Spam Detector is used to detect unwanted, malicious and virus infected texts and helps to separate them from the nonspam texts. It uses a binary type of classification containing the labels such as ‘ham’ (nonspam) and spam. Application of this can be seen in Google Mail (GMAIL) where it segregates the spam emails in order to prevent them from getting into the user’s inbox.

Technical goals :

Naive Bayes classifier: It is a supervised machine learning algorithm where words probabilities play the main rule here. If some words occur often in spam but not in ham, then this incoming e-mail is probably spam. Naïve bayes classifier technique has become a very popular method in mail filtering software. Bayesian filter should be trained to work effectively. Every word has certain probability of occurring in spam or ham email in its database. If the total of words probabilities exceeds a certain limit, the filter will mark the e-mail to either category.

Artificial Neural Networks classifier: An artificial neural network (ANN), also called simply a "Neural Network" (NN), is a computational model based on biological neural networks. It consists of an interconnected collection of artificial neurons. An artificial neural network is an adaptive system that changes its structure based on information that flows through the artificial network during a learning phase.

Analytical Problem Framing

The files contain one message per line. Each line is composed by two columns: v1 contains the label (ham or spam) and v2 contains the raw text.

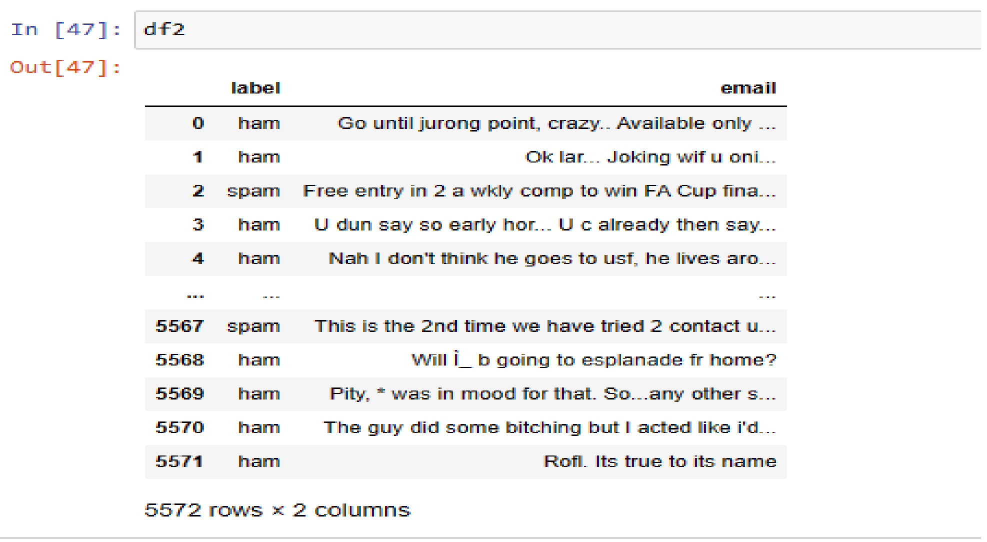
This corpus has been collected from free or free for research sources at the Internet:

A collection of 5573 rows SMS spam messages was manually extracted from the Grumbletext Web site. This is a UK forum in which cell phone users make public claims about SMS spam messages, most of them without reporting the very spam message received.

The identification of the text of spam messages in the claims is a very hard and time-consuming task, and it involved carefully scanning hundreds of web pages.

A subset of 3,375 SMS randomly chosen ham messages of the NUS SMS Corpus (NSC), which is a dataset of about 10,000 legitimate messages collected for research at the Department of Computer Science at the National University of Singapore. The messages largely originate from Singaporeans and mostly from students attending the University. These messages were collected from volunteers who were made aware that their contributions were going to be made publicly available.

The data we have looks like below :



```
In [47]: df2
Out[47]:
```

	label	email
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...
...	...	...
5567	spam	This is the 2nd time we have tried 2 contact u...
5568	ham	Will ì_ b going to esplanade fr home?
5569	ham	Pity, * was in mood for that. So...any other s...
5570	ham	The guy did some bitching but I acted like i'd...
5571	ham	Rofl. Its true to its name

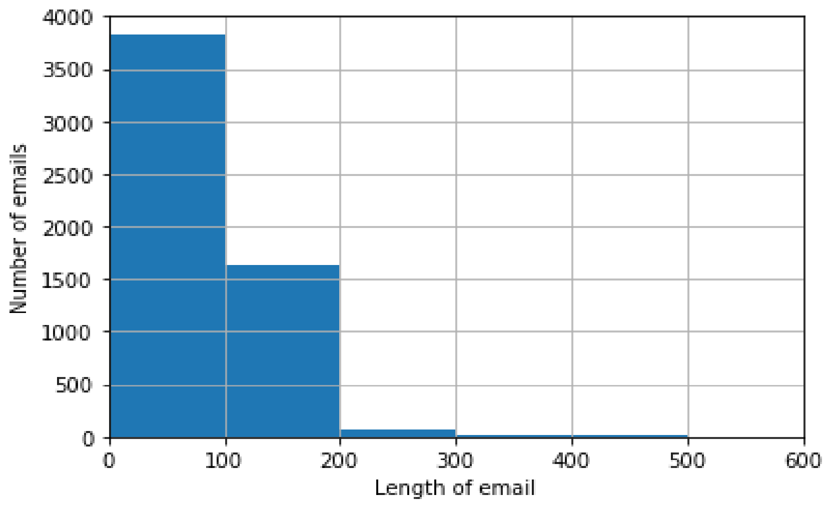
5572 rows × 2 columns

Process included :

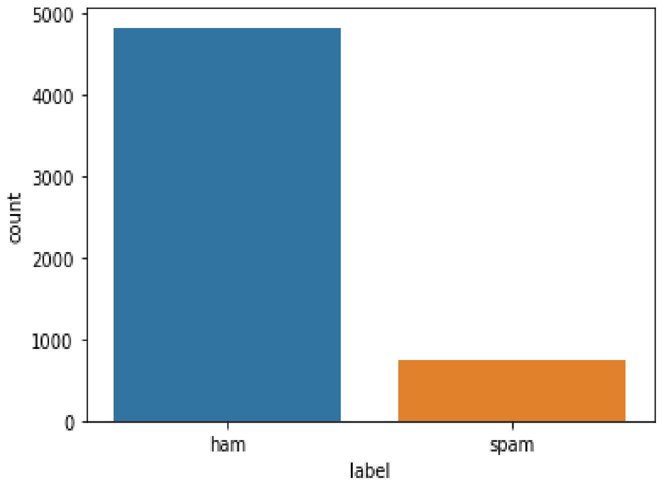
1. Calling the required libraries
2. Import necessary files
3. EDA
4. Separating input and target variables
5. Applying Countvectorizer
6. Splitting data for training and testing
7. Building the model
8. Parameter tuning
9. Saving the final model

EDA and Visualization

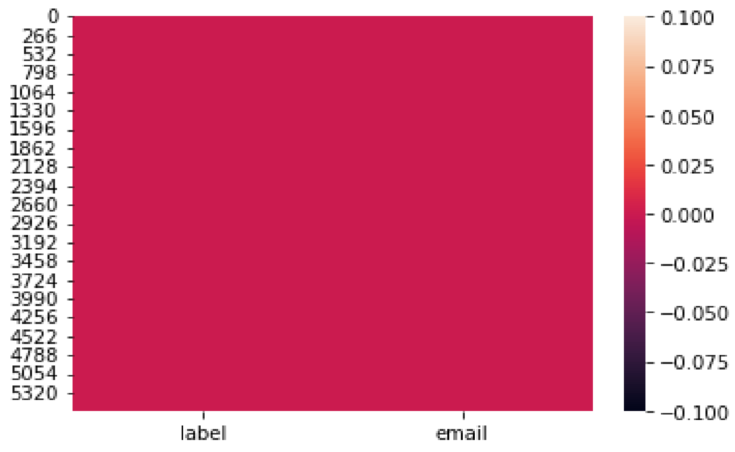
average length of email: 80.119



From the above visualization we can observe that emails have varying lengths from within 0 up to 500. The majority of emails have length up to 200, and as we move towards greater lengths, the number of emails keep on falling.



From the above visualizations, we can observe that the dataset consists of 4825 ‘ham’ emails and 747 spam emails.

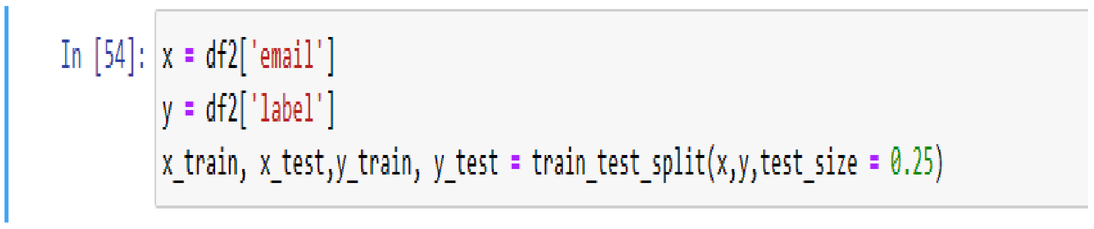


The above plot tells that there is no null data present in the dataset.

Algorithm & Techniques :

The algorithms that are used in the classifier model are Support vector classifier, Decision Tree Classifier and Random forest classifier.

Separating input and target variables, splitting train-test data :



```
In [54]: x = df2['email']
         y = df2['label']
         x_train, x_test,y_train, y_test = train_test_split(x,y,test_size = 0.25)
```

We'll use a train-test split method to train our email spam detector to recognize and categorize spam emails. The train-test split is a technique for evaluating the performance of a machine learning algorithm. We can use it for either classification or regression of any supervised learning algorithm.

The procedure involves taking a dataset and dividing it into two separate datasets. The first dataset is used to fit the model and is referred to as the training dataset. For the second dataset, the test dataset, we provide the input element to the model. Finally, we make predictions, comparing them against the actual output.

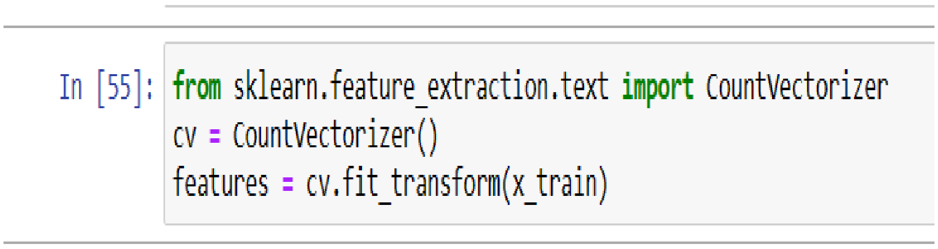
- Train dataset: used to fit the machine learning model
- Test dataset: used to evaluate the fit of the machine learning model

In practice, we'd fit the model on available data with known inputs and outputs. Then, we'd make predictions based on new examples for which we don't have the expected output or target values. We'll take the data from our sample .csv file, which contains examples pre-classified into spam and non-spam, using the labels spam and ham, respectively.

To split the data into our two datasets, we'll use scikit-learn's train_test_split() method.

Let's say we have 100 records in the loaded dataset. If we specify the test dataset is 30 percent, we'll split 70 records for training and use the remaining 30 records for testing.

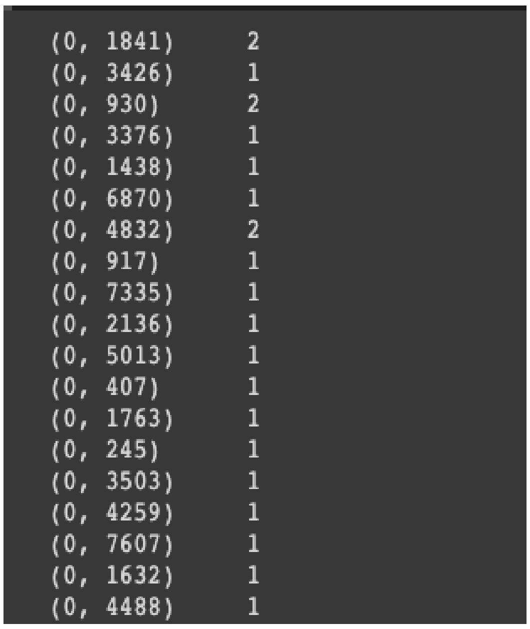
Applying count vectorizer :



```
In [55]: from sklearn.feature_extraction.text import CountVectorizer
         cv = CountVectorizer()
         features = cv.fit_transform(x_train)
```

In **cv= CountVectorizer(), CountVectorizer()** randomly assigns a number to each word in a process called tokenizing. Then, it counts the number of occurrences of words and saves it to cv. At this point, we've only assigned a method to cv.

features = cv.fit_transform(x_train) randomly assigns a number to each word. It counts the number of occurrences of each word, then saves it to cv. In the image below, 0 represents the index of the email. The number sequences in the middle column represent a word recognized by our function, and the numbers on the right indicate the number of times that word was counted:



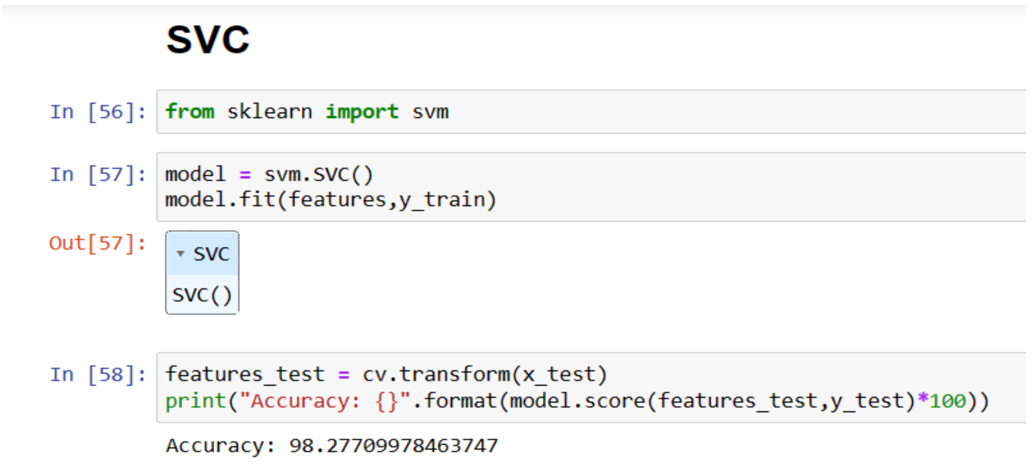
```
  (0, 1841)	2
  (0, 3426)	1
  (0, 930)	2
  (0, 3376)	1
  (0, 1438)	1
  (0, 6870)	1
  (0, 4832)	2
  (0, 917)	1
  (0, 7335)	1
  (0, 2136)	1
  (0, 5013)	1
  (0, 407)	1
  (0, 1763)	1
  (0, 245)	1
  (0, 3503)	1
  (0, 4259)	1
  (0, 7607)	1
  (0, 1632)	1
  (0, 4488)	1
```

For example, in the image above, the word corresponding to 1841 is used twice in email number 0.

Now, our machine learning model will be able to predict spam emails based on the number of occurrences of certain words that are common in spam emails.

Building the models :

SVM, the support vector machine algorithm, is a linear model for classification and regression. The idea of SVM is simple, the algorithm creates a line, or a hyperplane, which separates the data into classes. SVM can solve both linear and non-linear problems



SVC

```
In [56]: from sklearn import svm

In [57]: model = svm.SVC()
         model.fit(features,y_train)

Out[57]: ▾ SVC
         SVC()

In [58]: features_test = cv.transform(x_test)
         print("Accuracy: {}".format(model.score(features_test,y_test)*100))

         Accuracy: 98.27709978463747
```

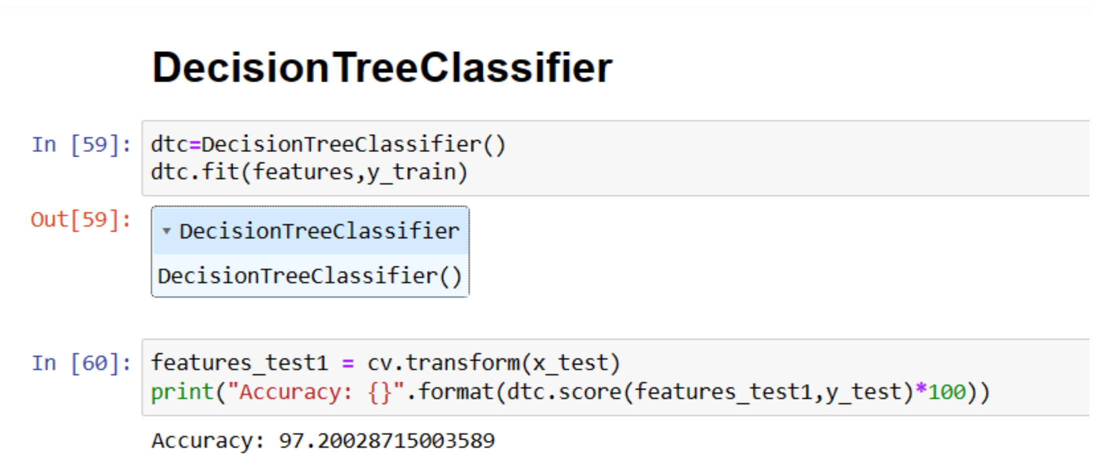
model = svm.SVC() assigns **svm.SVC()** to the **model**. In the **model.fit(features,y_train)** function, **model.fit** trains the model with features and **y_train**. Then, it checks the prediction against the **y_train** label and adjusts its parameters until it reaches the highest possible accuracy.

The **features_test = cv.transform(x_test)** function makes predictions from **x_test** that will go through count vectorization. It saves the results to the **features_test** file.

In the **print(model.score(features_test,y_test))** function, **mode.score()** scores the prediction of **features_test** against the actual labels in **y_test**. In the similar way we can build the other 2 models as well.

DecisionTreeClassifier, is a Supervised Machine Learning Algorithm that uses a set of rules to make decisions, similarly to how humans make decisions.

One way to think of a Machine Learning classification algorithm is that it is built to make decisions.

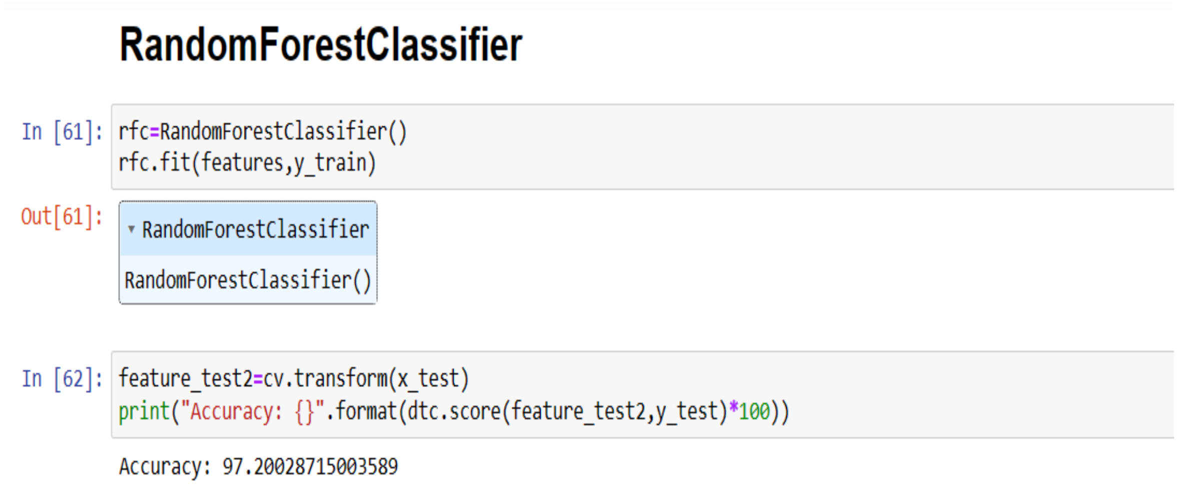


DecisionTreeClassifier

```
In [59]: dtc=DecisionTreeClassifier()
         dtc.fit(features,y_train)

Out[59]: ▾ DecisionTreeClassifier
         DecisionTreeClassifier()

In [60]: features_test1 = cv.transform(x_test)
         print("Accuracy: {}".format(dtc.score(features_test1,y_test)*100))

         Accuracy: 97.20028715003589
```

RandomForestClassifier, is a Supervised Machine Learning Algorithm that is used widely in Classification and Regression problems. It builds decision trees on different samples and takes their majority vote for classification and average in case of regression.

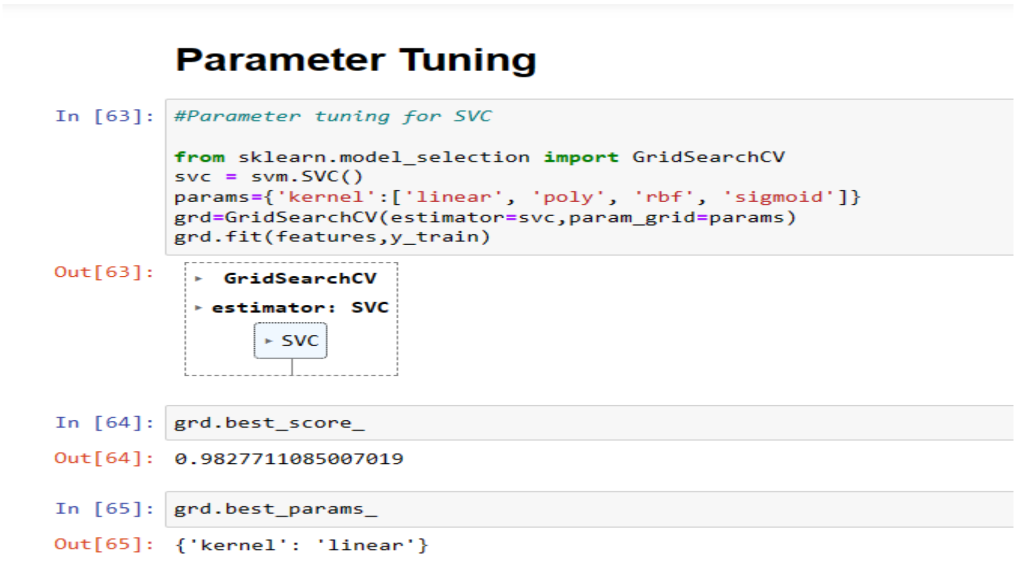
One of the most important features of the Random Forest Algorithm is that it can handle the data set containing continuous variables as in the case of regression and categorical variables as in the case of classification. It performs better results for classification problems.



RandomForestClassifier

```
In [61]: rfc=RandomForestClassifier()
         rfc.fit(features,y_train)
Out[61]: ▾ RandomForestClassifier
         RandomForestClassifier()

In [62]: feature_test2=cv.transform(x_test)
         print("Accuracy: {}".format(dtc.score(feature_test2,y_test)*100))

         Accuracy: 97.20028715003589
```

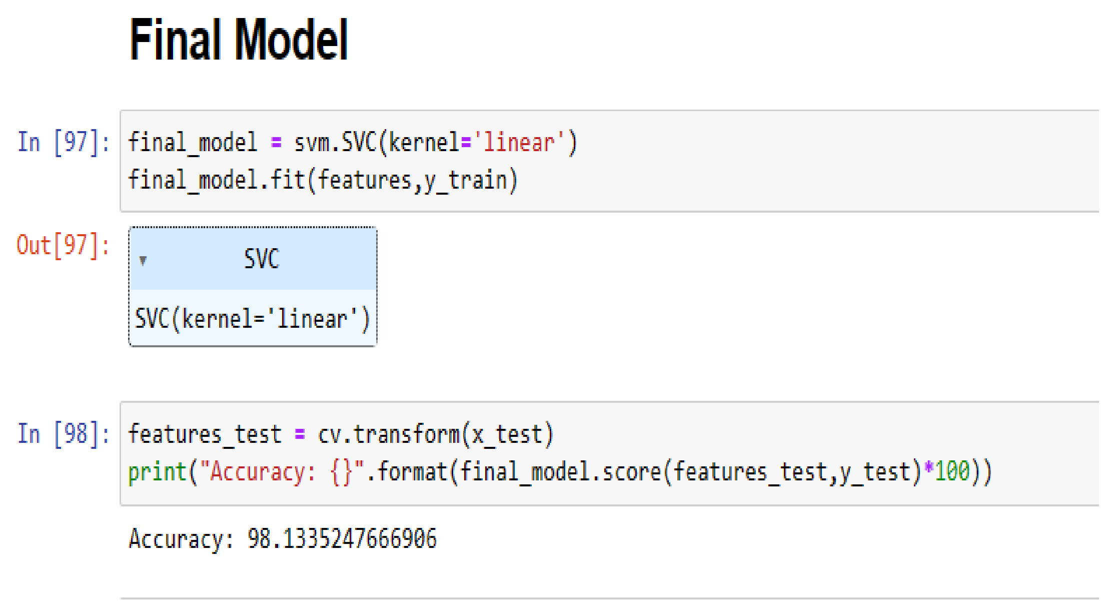
From the above 3 models, we can observe that SVC is giving the best score of 98.2%, hence we can do the parameter tuning on the same model and finalize the model with the best parameters. Parameter tuning on SVC can be done using the code as shown below.



Parameter Tuning

```
In [63]: #Parameter tuning for SVC

         from sklearn.model_selection import GridSearchCV
         svc = svm.SVC()
         params={'kernel':['linear', 'poly', 'rbf', 'sigmoid']}
         grd=GridSearchCV(estimator=svc,param_grid=params)
         grd.fit(features,y_train)
Out[63]: ▸ GridSearchCV
         ▸ estimator: SVC
             ▸ SVC

In [64]: grd.best_score_
Out[64]: 0.9827711085007019

In [65]: grd.best_params_
Out[65]: {'kernel': 'linear'}
```

From the above results we can see that SVC is giving the best result of **98.2%** where the best parameter for kernel is chosen to be **‘linear’.** Now we can finalize the model for the same parameters as shown below.

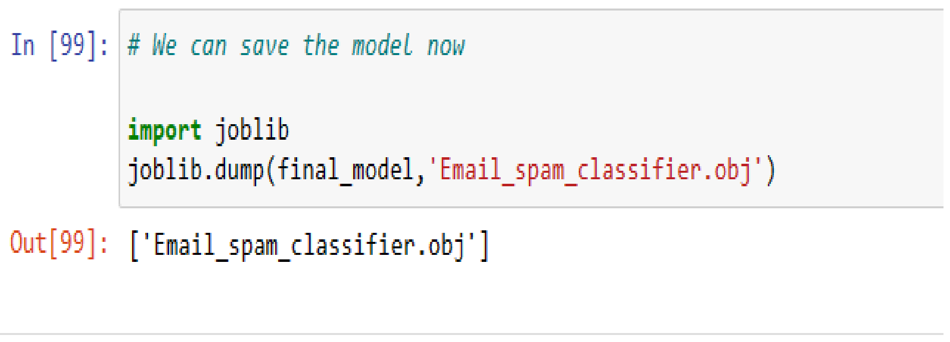


Final Model

```
In [97]: final_model = svm.SVC(kernel='linear')
         final_model.fit(features,y_train)

Out[97]: ▾        SVC
         SVC(kernel='linear')
```

```
In [98]: features_test = cv.transform(x_test)
         print("Accuracy: {}".format(final_model.score(features_test,y_test)*100))

         Accuracy: 98.1335247666906
```

Saving the model :



```
In [99]: # We can save the model now

         import joblib
         joblib.dump(final_model,'Email_spam_classifier.obj')

Out[99]: ['Email_spam_classifier.obj']
```

CONCLUSION

To summarize this project, the first step involved studying the dataset.

The second major step was to separate the input and target variables, followed by that splitting was done for training and testing the model. Applied Countvectorizer where it randomly assigns a number to each word in a process called tokenizing. Then, it counts the number of occurrences of words.

The third step was to build models SVC, DecisionTreeClassifier and RandomForestClassifier and check for the best performing model comparing their accuracy scores.

Finally parameter tuning is done on SVC as it was performing better compared to other 2 models and chose the best model with best parameter where **kernel = linear,** which was giving the best score of **98.2%.**

Although we have built a model which is giving us accuracy of 98.2%, it is always good to build 4 to 5 models for the same test and train data to check if other model perform better than the current one.