INFO 6105

Data Science Engineering Methods and Tools

Team 1 Monday Batch

Supervised Machine Learning Bot Detection Techniques to Identify Social Twitter Bots

Team Members: -

Atulya Sharma (001306609)

Samruddhi Pasari (001306800)

Vivek Koli (001363809)

Contents

1. Introduction
2. Background of the Project
3. Methodology
4. Dataset
5. Results and Analysis
6. Conclusion
7. Future Scope

Introduction

With the increase in the number of active users of the internet across the globe, the number of bots has also seen an increase in the last few years. According to an article published by thenextweb.com nearly 40% of the internet traffic in 2018 was made up of bots and this figure is rising every year. Even though not all bots are deployed with malefic intentions, it still is a cause of worry.
Twitter just like every other social media platform has also been a victim of these bots and thus the use of data science and machine learning algorithms can be helpful in the identification of these bots.

Background of the Project:

Problem Statement:

To ensure the safety of user privacy and safety, Twitter and other social media have to constantly remove the bot accounts. Identifying these bot accounts is a labour intensive task and isn’t the most efficient thing to do. In this project our biggest problem is to figure out the most effective way to identify these bots accounts using supervised machine learning algorithms.

Objectives:

- Choosing the perfect dataset for this project and cleaning it for implementation of Machine Learning algorithms.
- The primary objective for this project is to implement supervised machine learning algorithms which includes Logistic Regression, Support Vector Machine (SVM) and Random Forest Classifiers on the test dataset to classify the accounts between bots and users.
- Finding the most accurate algorithm for the dataset and creating visualizations

Methodology

We have used the following libraries in our system for the analysis:

● Numpy

● Pandas

● Sklearn

● Matplotlib

a. NumPy

NumPy is the fundamental package for scientific computing with Python. NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases. In our project we have implemented NumPy for computing mean darkness and mean RGB value of each image, data cleaning (if the ratio is infinity we set it to 99999 and if it’s NaN set to 0).

b. Pandas

Pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language. It is built on the NumPy package and its key data structure is called the DataFrame. DataFrames allow you to store and manipulate tabular data in rows of observations and columns of variables. In our project we have implemented pandas to import a csv file as a dataframe, concatenate multiple dataframes into a single dataframe, and for setting image features.

c. SciKit-Learn

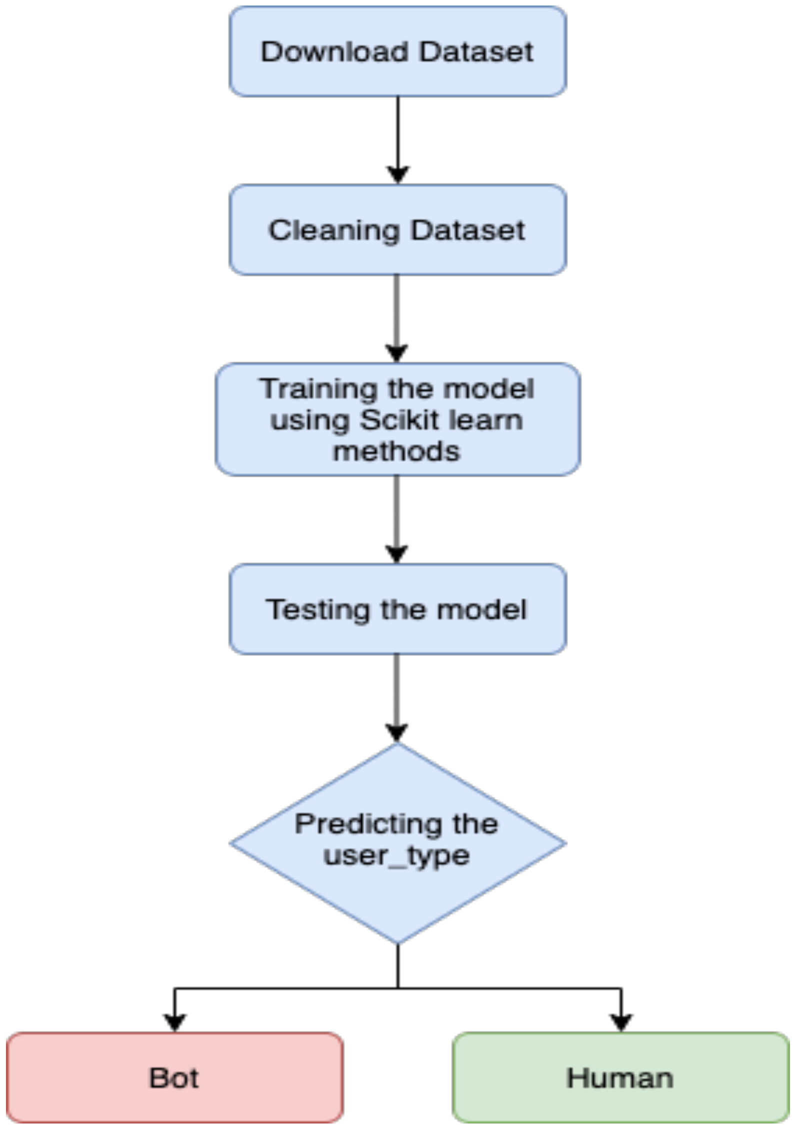
Scikit-learn library is used to implement various algorithms like support vector machines, random forests, Gaussian Naive Bayes, Scalers, and k-neighbours. It also supports

Python numerical and scientific libraries like NumPy and SciPy. In our project, we have used SVC, Scalers and k-neighbors for model training

d. MatPlotLib

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits. In our project, we use MatPlotLib to display a Bar Chart to visualize the Weather details by Month versus Days.

Flow Diagram:



Our Approach

The project can be implemented using three classification models namely:

- Random Forest
- Logistic Regression
- Support Vector Machine (SVM)

The result of these models can be further analysed to determine the model with highest accuracy. However, the dataset being big enough may require high computing power. The problem can be resolved by dimensionality reduction using Principal Component Analysis (PCA). The 25+ columns that are present in the dataset can be further reduced by selecting the uncorrelated variables which are also known as principal components. These components comprise the necessary information required to train the model.The first model, Naive Bayes is a generative model which is very suitable for high-dimensional datasets. This model makes naive assumptions about the generative model for each label to find rough approximation for each class and then uses Bayes classification to define a distribution pattern for each class, here bots and humans. The second model, Logistic Regression, is a discriminative model that logs the probability of a discrete dependent variable in the form of linear combination of independent variables. It also penalizes increasingly large errors at a constant cost. The third model, Support Vector Machine models data as existing in some p-1 dimensional space, where each data point has p features. The objective of the SVM is to learn a separating hyperplane that best divides the classes. In total, six models will be trained: three classification models on full feature space and three classification models using PCA features.

Dataset

Source of Dataset

The dataset used for the project is from 2017 and was created by Cresci. It has multiple datasets on genuine accounts as well as several kinds of bot accounts.

https://botometer.iuni.iu.edu/bot-repository/datasets/cresci-2017/cresci-2017.csv.zip

Specifications of Dataset

The real world Twitter datasets by Cresci et al includes data from genuine accounts and bot accounts. Each of the classification further includes meta data of tweets and users. It has a total of 25+ feature meta-data released in a CSV format. The statistics about the data used for this project can be described using the following table.

Classification Group	Description	No. of Accounts	Tweet count	Year
Humans	Verified accounts by human	3,474	8,337,522	2011
Bots	Account used for spamming	2,611	6,148,293	2013

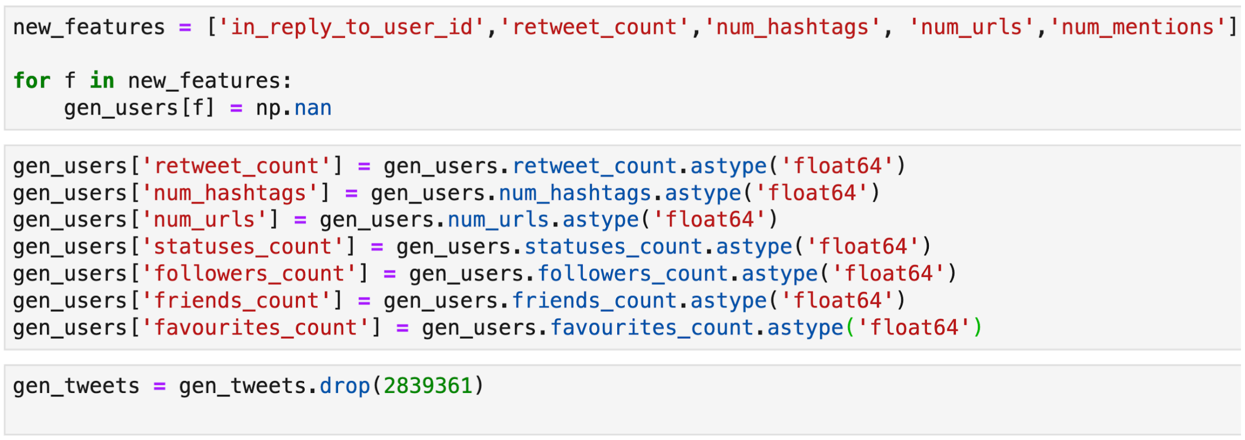
Data Preprocessing

Before the dataset is finalized it has to undergo a lot of preprocessing which involves removal or certain columns, handling missing values by replacing them with ‘0’ and removing rows that are not very useful.

Since we had multiple datasets, we had to merge and concatenate them into one final dataset which is then normalized so that all the rows have values that lie between 0 and 1

In Order to make the dataset compatible with the machine learning algorithms, we had to change certain columns from one data type to another.

Code Snippets

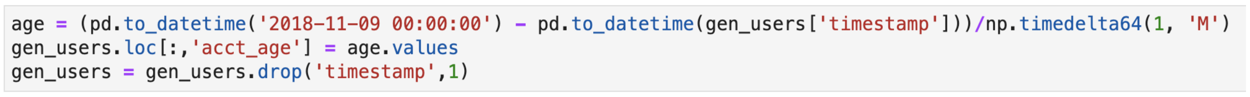


```python
new_features = ['in_reply_to_user_id','retweet_count','num_hashtags', 'num_urls','num_mentions']

for f in new_features:
    gen_users[f] = np.nan
```

```python
gen_users['retweet_count'] = gen_users.retweet_count.astype('float64')
gen_users['num_hashtags'] = gen_users.num_hashtags.astype('float64')
gen_users['num_urls'] = gen_users.num_urls.astype('float64')
gen_users['statuses_count'] = gen_users.statuses_count.astype('float64')
gen_users['followers_count'] = gen_users.followers_count.astype('float64')
gen_users['friends_count'] = gen_users.friends_count.astype('float64')
gen_users['favourites_count'] = gen_users.favourites_count.astype('float64')
```

```python
gen_tweets = gen_tweets.drop(2839361)
```

Adding new features to the dataset, changing the data types and dropping a row based on the index



```python
age = (pd.to_datetime('2018-11-09 00:00:00') - pd.to_datetime(gen_users['timestamp']))/np.timedelta64(1, 'M')
gen_users.loc[:,'acct_age'] = age.values
gen_users = gen_users.drop('timestamp',1)
```

Adding computed column from another column

Column Descriptions

Following are the columns with description after data cleaning:

- acct_age: the age of the twitter account after it was created
- favourites_count: the number of tweets favorited by the account user
- followers_count: the count of the accounts that follow the user
- following: the count of accounts followed by the user
- friends_count: number of friends of this user
- id: unique id of the user
- in_reply_to_user_id: user id whose tweet has been replied to
- lang: language of the tweet

- name: name of the account
- num_hashtags: number of hashtags in the particular tweet
- num_mentions: number of accounts mentioned in the particular tweet
- num_urls: number of urls added in the particular tweet
- retweet_count: number of tweets retweeted by the user
- statuses_count: number of statuses posted by the user
- user_type: Type of user (0 for Bot and 1 for Human)
- verified: Whether the user account has been verified

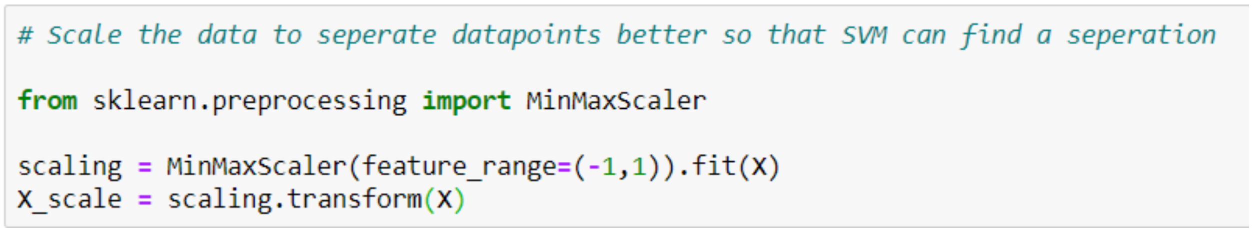
Results and Analysis

Training Model

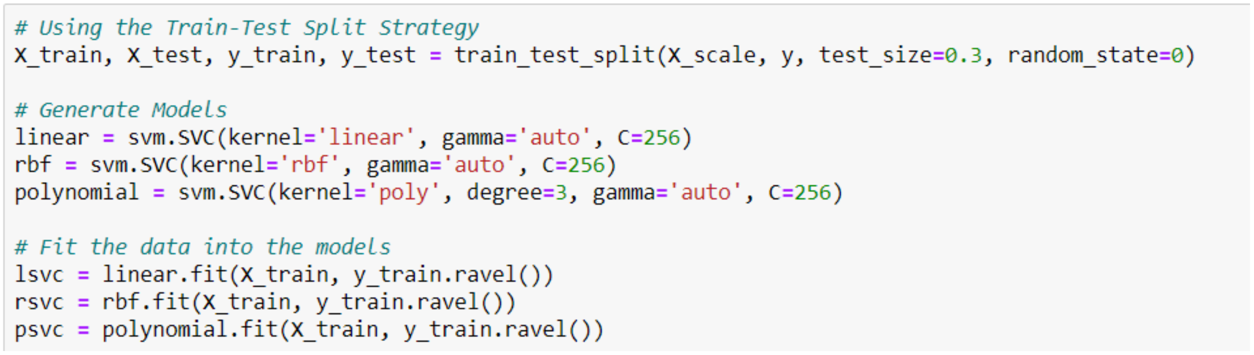
Support Vector Machine

"Support Vector Machine" (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. The problem statement is classification type of bot and human utilizing the Twitter dataset consisting of users and their tweets information. In SVM we have n features used to classify the users into two classes. For this problem statement we have 13 features and two classes (bot, human). SVM performs classification in n-dimensional space and creates a hyperplane that differentiate between the two classes. The SVM has a function named kernel which takes low dimensional input and converts it into high dimensional space for non-linear separation problems. In this project we are using three types of kernel namely: liner, radial basis function and polynomial.

Before training the model we have scaled our data using MinMaxScaler to transform the data.

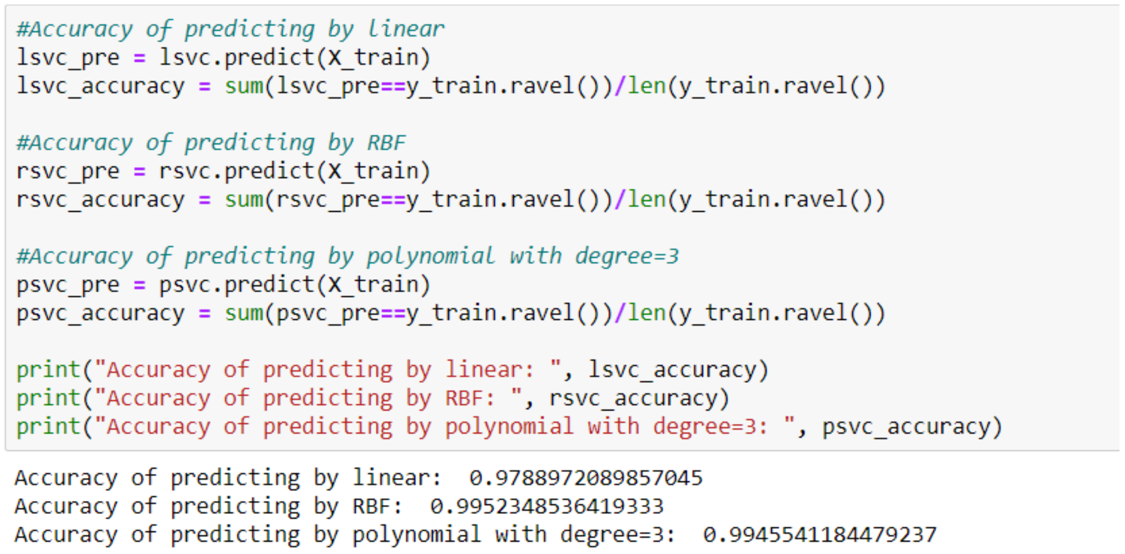


```
# Scale the data to seperate datapoints better so that SVM can find a seperation

from sklearn.preprocessing import MinMaxScaler

scaling = MinMaxScaler(feature_range=(-1,1)).fit(X)
X_scale = scaling.transform(X)
```



```
# Using the Train-Test Split Strategy
X_train, X_test, y_train, y_test = train_test_split(X_scale, y, test_size=0.3, random_state=0)

# Generate Models
linear = svm.SVC(kernel='linear', gamma='auto', C=256)
rbf = svm.SVC(kernel='rbf', gamma='auto', C=256)
polynomial = svm.SVC(kernel='poly', degree=3, gamma='auto', C=256)

# Fit the data into the models
lsvc = linear.fit(X_train, y_train.ravel())
rsvc = rbf.fit(X_train, y_train.ravel())
psvc = polynomial.fit(X_train, y_train.ravel())
```

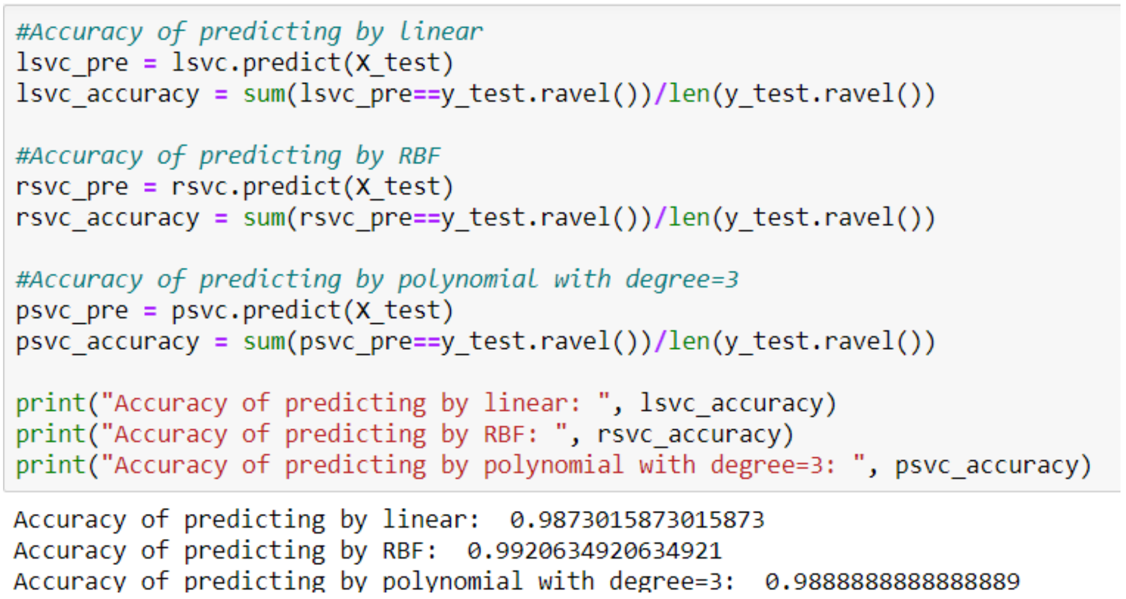
After training the SVM model with all the three kernel we calculated accuracies for train and test data.



```python
#Accuracy of predicting by linear
lsvc_pre = lsvc.predict(X_train)
lsvc_accuracy = sum(lsvc_pre==y_train.ravel())/len(y_train.ravel())

#Accuracy of predicting by RBF
rsvc_pre = rsvc.predict(X_train)
rsvc_accuracy = sum(rsvc_pre==y_train.ravel())/len(y_train.ravel())

#Accuracy of predicting by polynomial with degree=3
psvc_pre = psvc.predict(X_train)
psvc_accuracy = sum(psvc_pre==y_train.ravel())/len(y_train.ravel())

print("Accuracy of predicting by linear: ", lsvc_accuracy)
print("Accuracy of predicting by RBF: ", rsvc_accuracy)
print("Accuracy of predicting by polynomial with degree=3: ", psvc_accuracy)
```

```
Accuracy of predicting by linear:  0.9788972089857045
Accuracy of predicting by RBF:  0.9952348536419333
Accuracy of predicting by polynomial with degree=3:  0.9945541184479237
```

Accuracy calculation of train data for kernel: liner, rbf, polynomial with degree 3

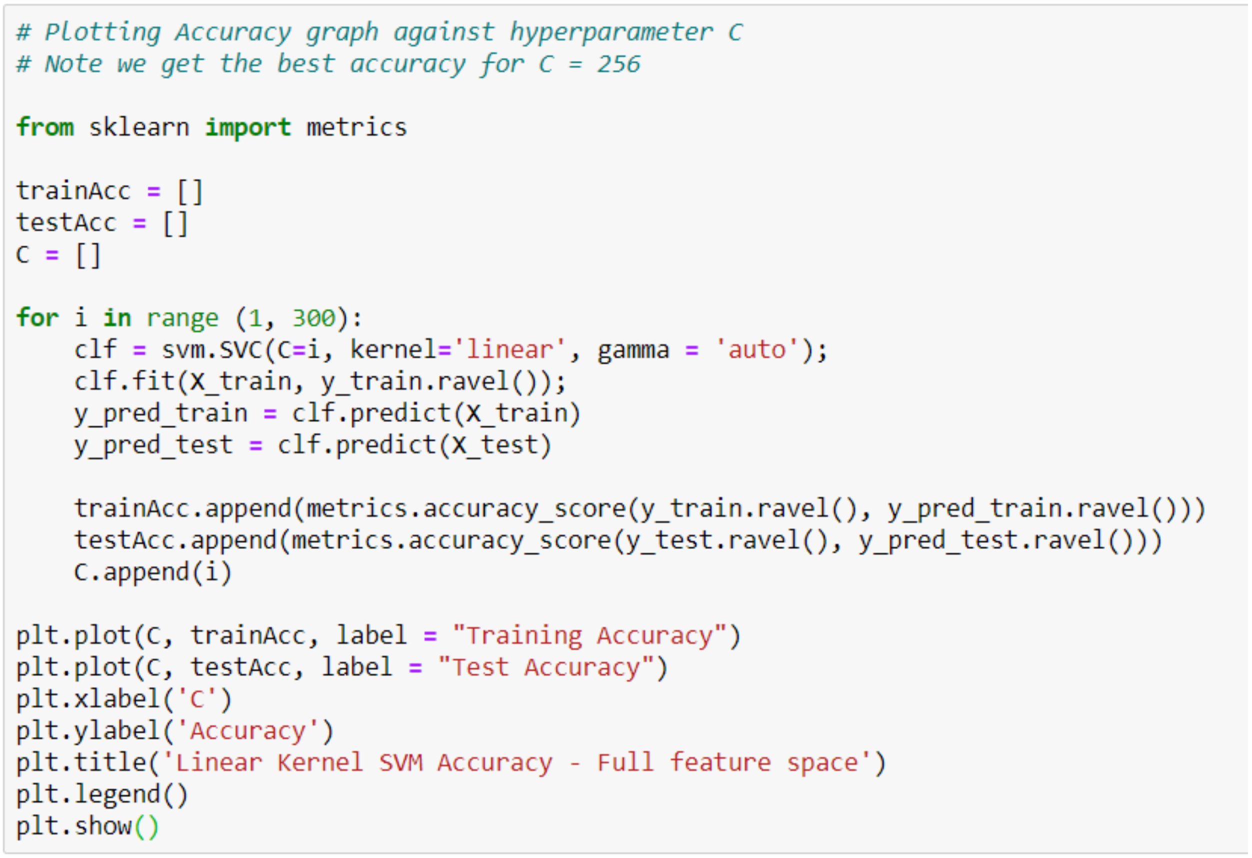


```python
#Accuracy of predicting by linear
lsvc_pre = lsvc.predict(X_test)
lsvc_accuracy = sum(lsvc_pre==y_test.ravel())/len(y_test.ravel())

#Accuracy of predicting by RBF
rsvc_pre = rsvc.predict(X_test)
rsvc_accuracy = sum(rsvc_pre==y_test.ravel())/len(y_test.ravel())

#Accuracy of predicting by polynomial with degree=3
psvc_pre = psvc.predict(X_test)
psvc_accuracy = sum(psvc_pre==y_test.ravel())/len(y_test.ravel())

print("Accuracy of predicting by linear: ", lsvc_accuracy)
print("Accuracy of predicting by RBF: ", rsvc_accuracy)
print("Accuracy of predicting by polynomial with degree=3: ", psvc_accuracy)
```

```
Accuracy of predicting by linear:  0.9873015873015873
Accuracy of predicting by RBF:  0.9920634920634921
Accuracy of predicting by polynomial with degree=3:  0.9888888888888889
```

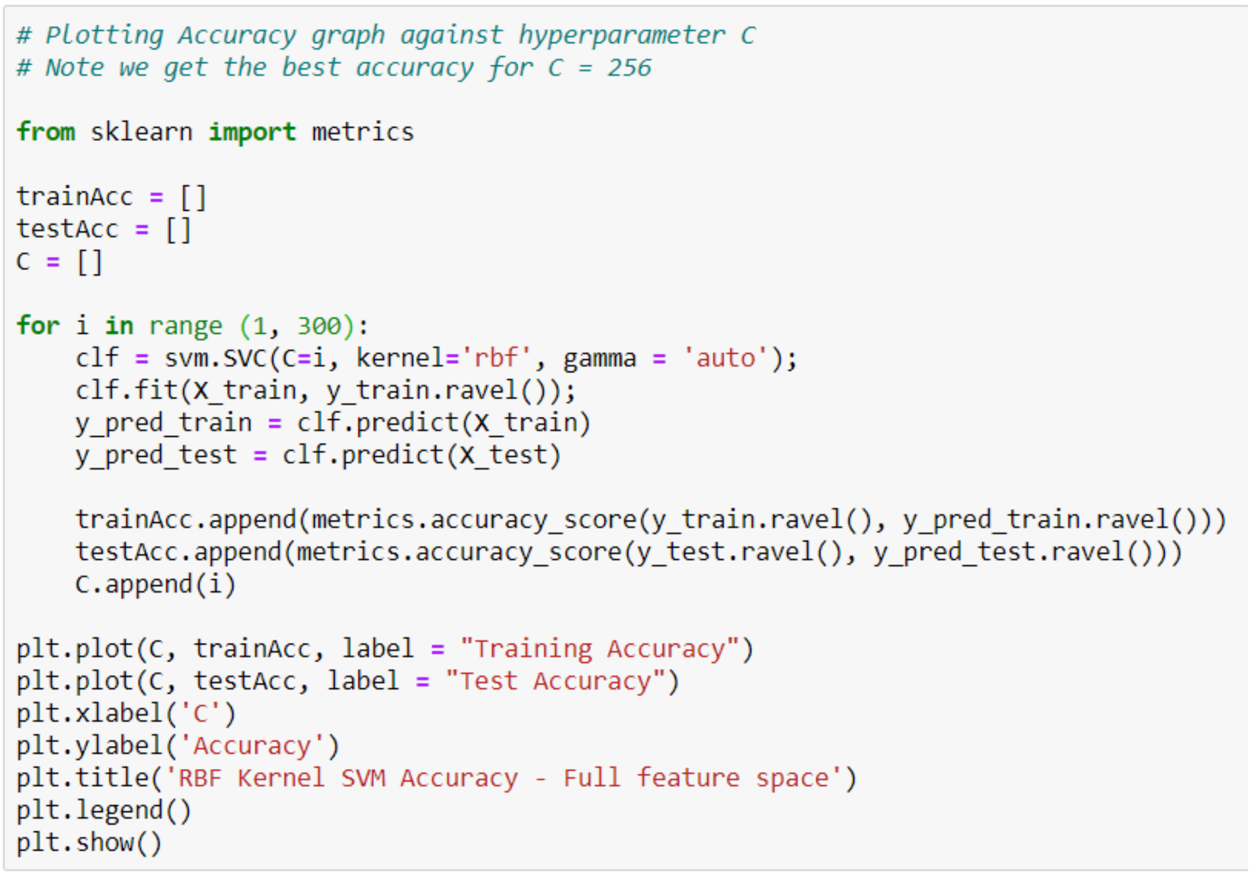
Accuracy calculation of test data for kernel: liner, rbf, polynomial with degree 3

We trained the scaled data with hyperparameter value C. We trained the data for C from 1 to 300 to determine best accuracy for all three models.



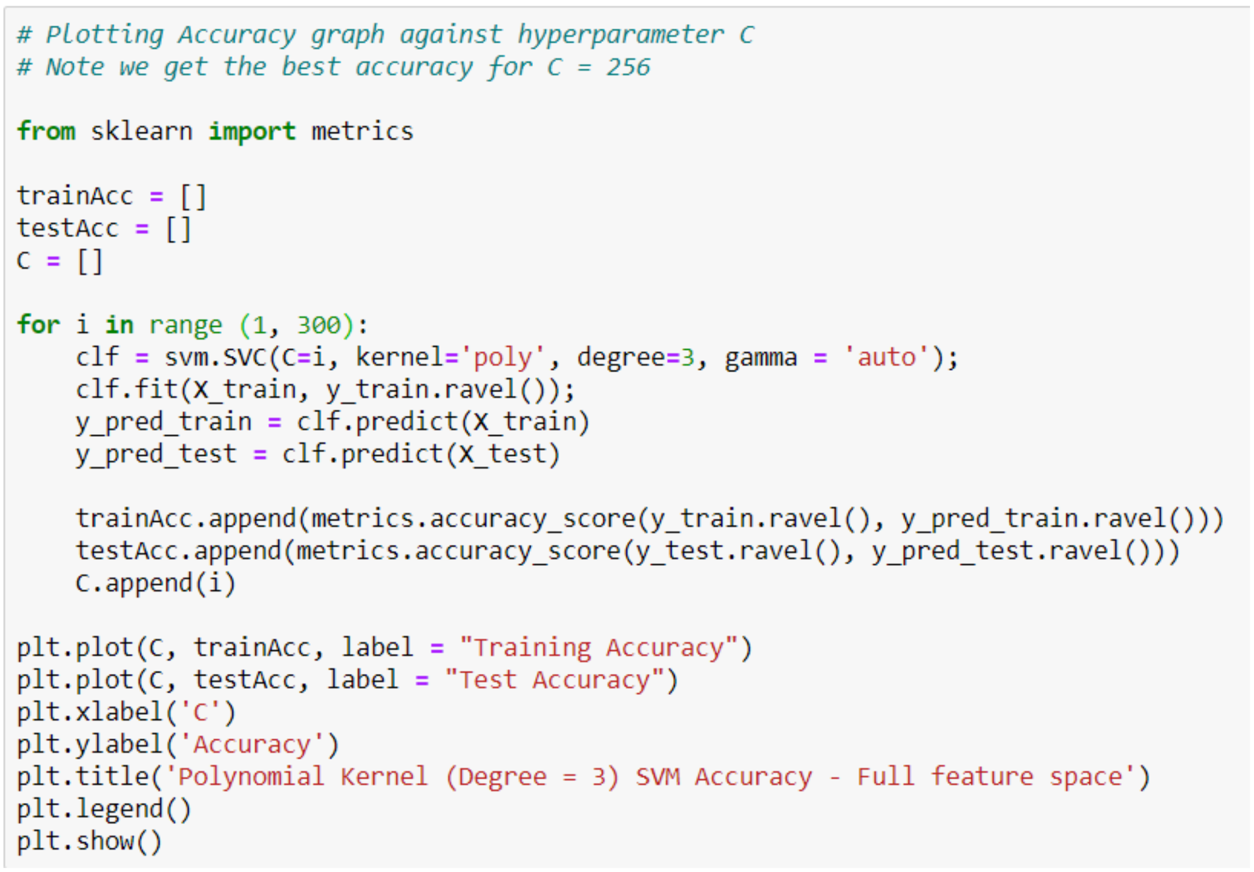
```python
# Plotting Accuracy graph against hyperparameter C
# Note we get the best accuracy for C = 256

from sklearn import metrics

trainAcc = []
testAcc = []
C = []

for i in range (1, 300):
    clf = svm.SVC(C=i, kernel='linear', gamma = 'auto');
    clf.fit(X_train, y_train.ravel());
    y_pred_train = clf.predict(X_train)
    y_pred_test = clf.predict(X_test)

    trainAcc.append(metrics.accuracy_score(y_train.ravel(), y_pred_train.ravel()))
    testAcc.append(metrics.accuracy_score(y_test.ravel(), y_pred_test.ravel()))
    C.append(i)

plt.plot(C, trainAcc, label = "Training Accuracy")
plt.plot(C, testAcc, label = "Test Accuracy")
plt.xlabel('C')
plt.ylabel('Accuracy')
plt.title('Linear Kernel SVM Accuracy - Full feature space')
plt.legend()
plt.show()
```

Plotting Accuracy against hyperparameter C for linear kernel



```python
# Plotting Accuracy graph against hyperparameter C
# Note we get the best accuracy for C = 256

from sklearn import metrics

trainAcc = []
testAcc = []
C = []

for i in range (1, 300):
    clf = svm.SVC(C=i, kernel='rbf', gamma = 'auto');
    clf.fit(X_train, y_train.ravel());
    y_pred_train = clf.predict(X_train)
    y_pred_test = clf.predict(X_test)

    trainAcc.append(metrics.accuracy_score(y_train.ravel(), y_pred_train.ravel()))
    testAcc.append(metrics.accuracy_score(y_test.ravel(), y_pred_test.ravel()))
    C.append(i)

plt.plot(C, trainAcc, label = "Training Accuracy")
plt.plot(C, testAcc, label = "Test Accuracy")
plt.xlabel('C')
plt.ylabel('Accuracy')
plt.title('RBF Kernel SVM Accuracy - Full feature space')
plt.legend()
plt.show()
```

Plotting Accuracy against hyperparameter C for rbf kernel

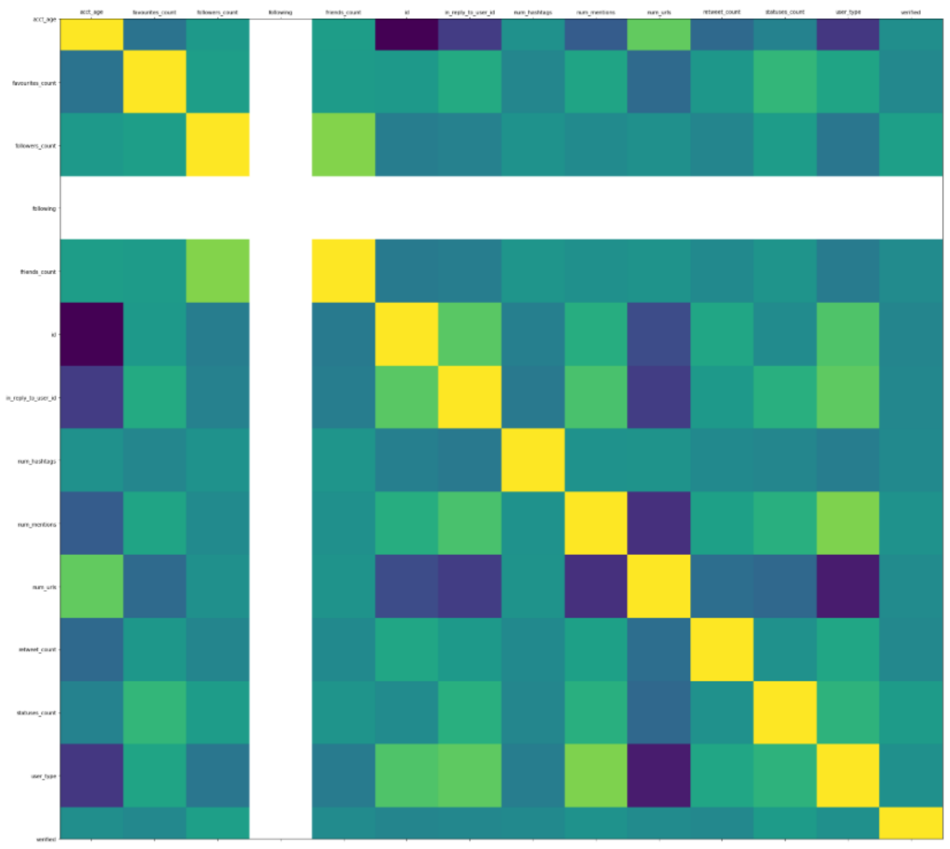


```python
# Plotting Accuracy graph against hyperparameter C
# Note we get the best accuracy for C = 256

from sklearn import metrics

trainAcc = []
testAcc = []
C = []

for i in range (1, 300):
    clf = svm.SVC(C=i, kernel='poly', degree=3, gamma = 'auto');
    clf.fit(X_train, y_train.ravel());
    y_pred_train = clf.predict(X_train)
    y_pred_test = clf.predict(X_test)

    trainAcc.append(metrics.accuracy_score(y_train.ravel(), y_pred_train.ravel()))
    testAcc.append(metrics.accuracy_score(y_test.ravel(), y_pred_test.ravel()))
    C.append(i)

plt.plot(C, trainAcc, label = "Training Accuracy")
plt.plot(C, testAcc, label = "Test Accuracy")
plt.xlabel('C')
plt.ylabel('Accuracy')
plt.title('Polynomial Kernel (Degree = 3) SVM Accuracy - Full feature space')
plt.legend()
plt.show()
```

Plotting Accuracy against hyperparameter C for polynomial kernel

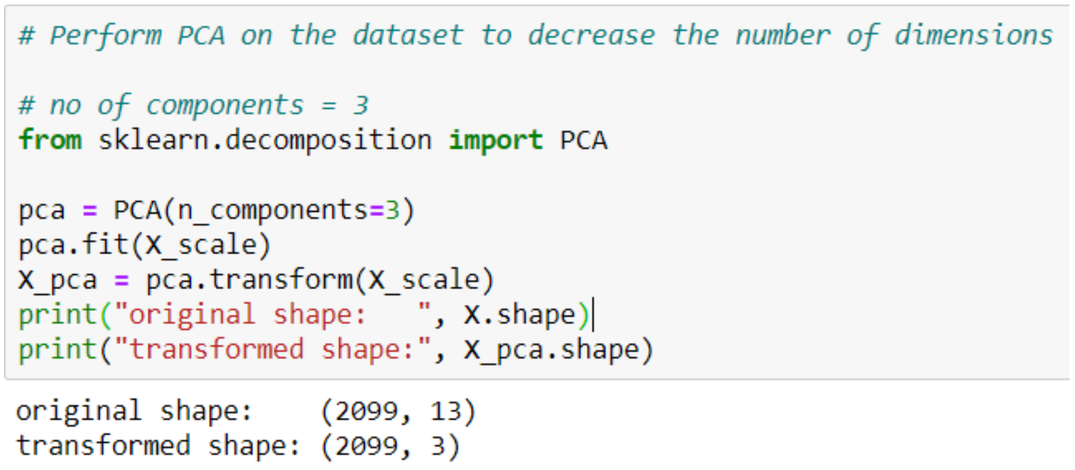
Principal Component Analysis

The dataset we are using for this project has 13 features. Training various models on large datasets with a lot of features may result in poor accuracy. Also it may become difficult for us to determine the important variable. PCA is used to obtain low dimensional important variables from a large set of features.It converts a set of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components.Eliminating components that contain low variation, decreases the dimension of the observation without losing significant amount of information.



Correlation matrix for all features

Using PCA we determined 2 and 3 most important features and visualized the data against it.

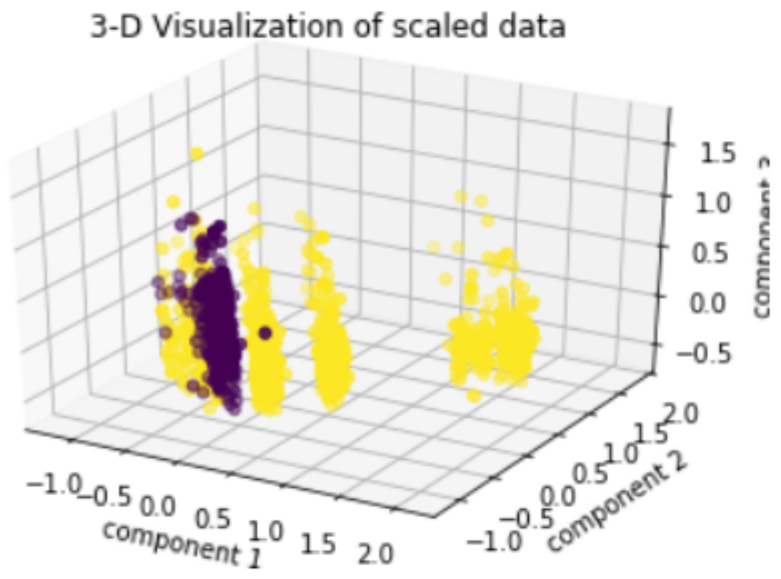


```
# Perform PCA on the dataset to decrease the number of dimensions

# no of components = 3
from sklearn.decomposition import PCA

pca = PCA(n_components=3)
pca.fit(X_scale)
X_pca = pca.transform(X_scale)
print("original shape:   ", X.shape)
print("transformed shape:", X_pca.shape)
```

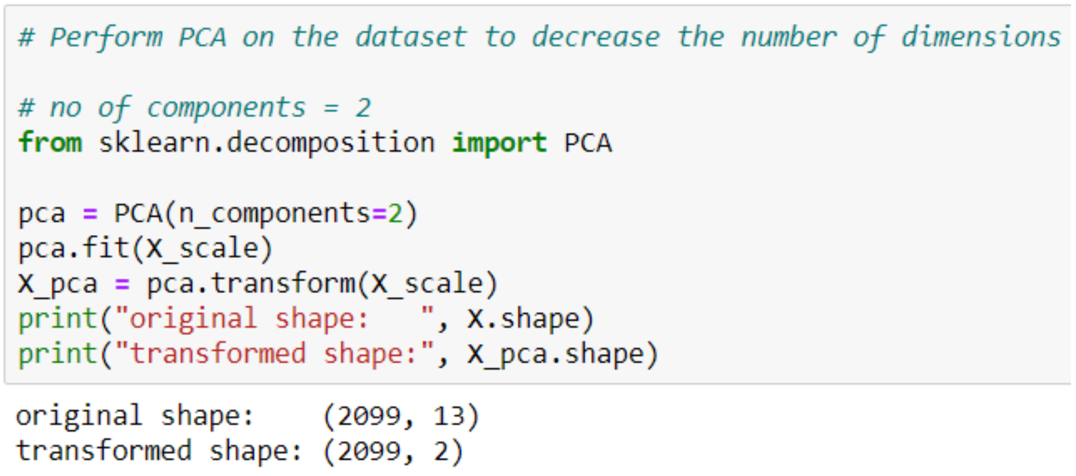
```
original shape:    (2099, 13)
transformed shape: (2099, 3)
```

Principal Component Analysis on scaled data for n_component = 3

With only three features we were able to plot a 3-dimensional plot for our scaled data.



3-D data visualization after PCA

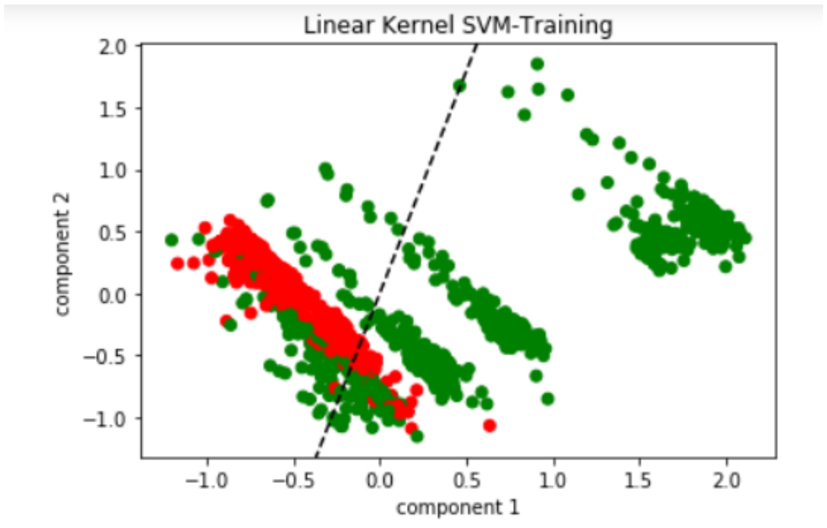


```
# Perform PCA on the dataset to decrease the number of dimensions

# no of components = 2
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
pca.fit(X_scale)
X_pca = pca.transform(X_scale)
print("original shape:   ", X.shape)
print("transformed shape:", X_pca.shape)
```

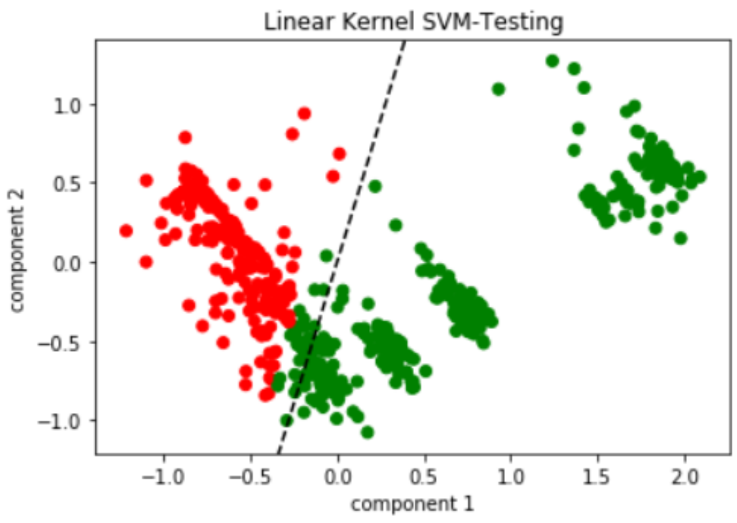
```
original shape:    (2099, 13)
transformed shape: (2099, 2)
```

Principal Component Analysis on scaled data for n_component = 2

We have also plotted 2-D visualization after training and modeling it with linear SVM.



```
Testing Accuracy: 0.8825396825396825
```



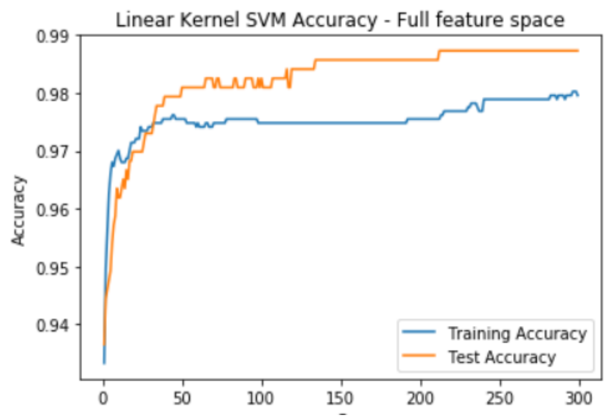
SVM linear classifier with PCA n_component = 2

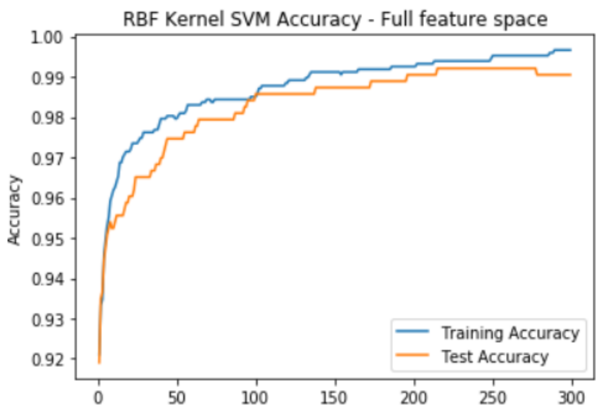
After performing PCA we trained all three models with newly transformed data and determined accuracies.

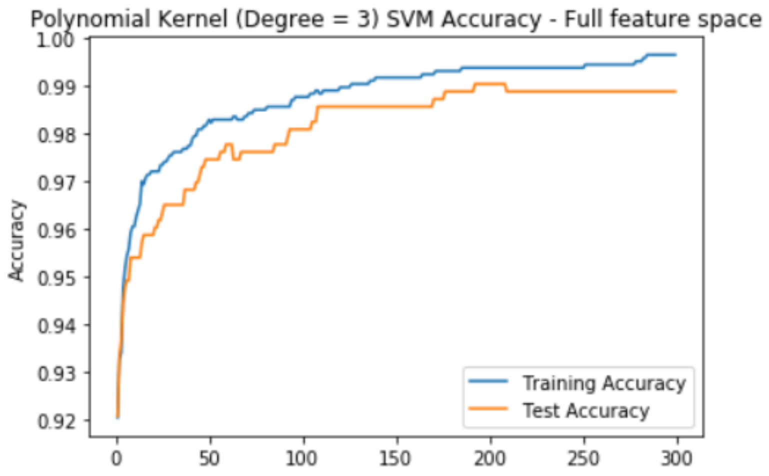
Analysis for SVM model

In this project we have used SVM to train the scaled data. Three kernels: linear, radial basis function and polynomial are used for this purpose. We modeled our data for hyperparameter C from 1 to 300 to determine the best accuray. The C parameter tells the SVM optimization how much you want to avoid misclassifying each training example. For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly. Conversely, a very small value of C will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassified more points.

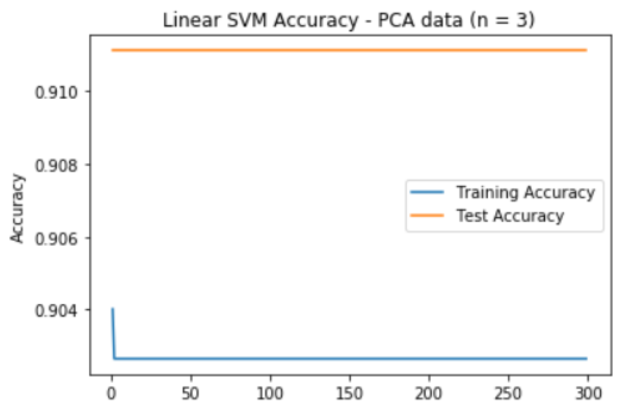
The graphs are plotted for all the 13 features in our dataset.

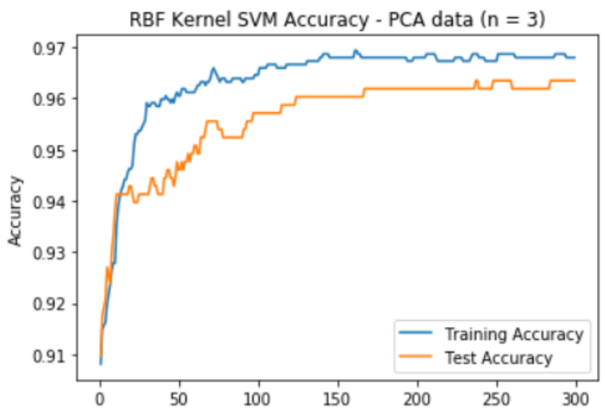


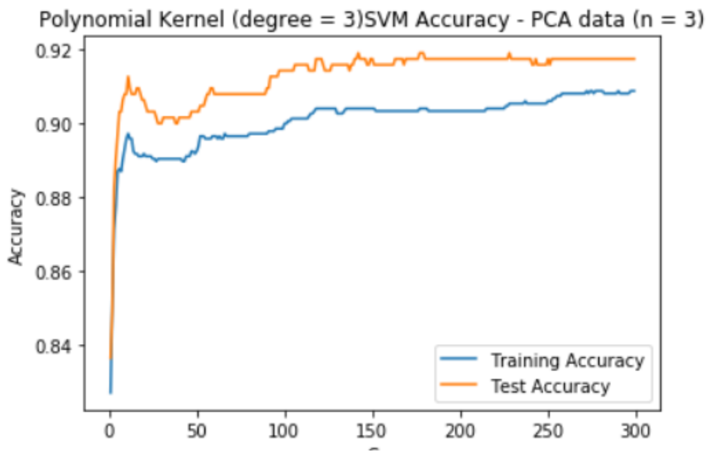




Principal Component Analysis reduces the number of components and affects the accuracy. The following are the graphs plotted for accuracies against hyperparameter C for n_components =2,3

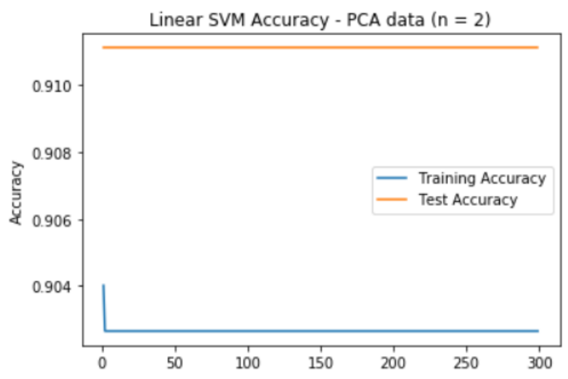


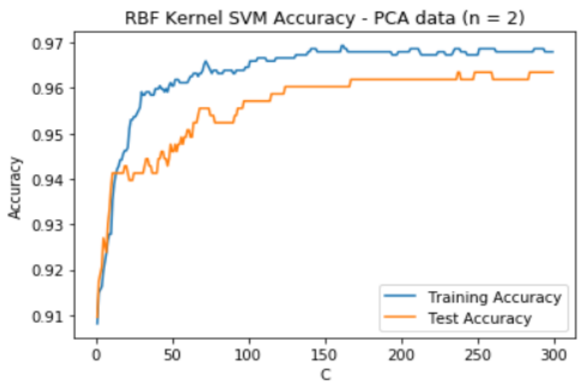


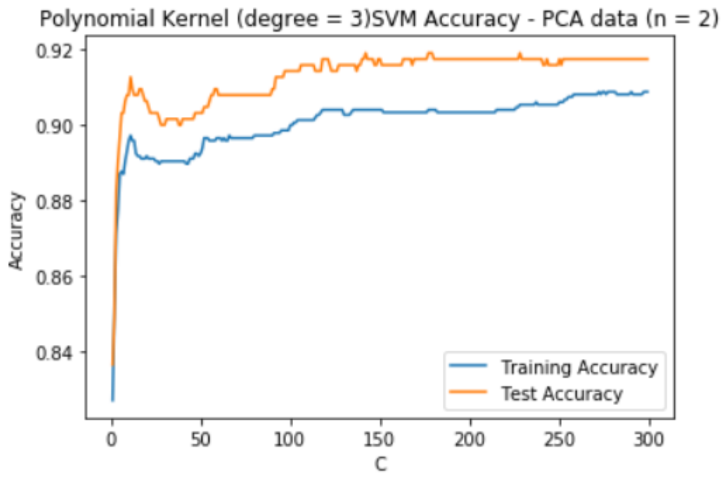


Accuracy vs C graph for PCA data n_components=3

The Accuracy has reduced for PCA data with n_components =2 as it will lose a significant amount of information while reducing the feature columns.







Accuracy vs C graph for PCA data n_components=2

By analysing the SVM classifier for accuracy vs C for scaled data and PCA data it can be determined that accuracy is reduced for PCA data specifically for n_components = 2. The hyperparameter value C = 256 gives the best accuracy in both the cases.

Logistic Regression

Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist. In regression analysis, logistic regression (or logit regression) is estimating the parameters of a logistic model (a form of binary regression). Mathematically, a binary logistic model has a dependent variable with two possible values, such as pass/fail which is represented by an indicator variable, where the two values are labeled "0" and "1".

Before training the data using Logistic Regression, we scale the data using MinMaxScaler similar to what we did before implementing SVM.

```
# Splitting test and train data before implementing Logistic Regresssion

X_train, X_test, y_train, y_test = train_test_split(X_scale, y, test_size=0.3, random_state=0)

# Implementing Logistic Regression

log_reg = LogisticRegression(solver='lbfgs')

# Fit the data into the model

log_reg.fit(X_train, y_train.ravel())
```

After scaling the data, we split the train and the test data followed by predicting the accuracies for train as well as test data

```
# Accuracy of predicting the train data

logreg_train = log_reg.predict(X_train)
logreg_train_accuracy = sum(logreg_train==y_train.ravel())/len(y_train.ravel())
print('Accuracy of Training Data',logreg_train_accuracy)

# Accuracy of predicting the test data

print('Accuracy of Testing Data',log_reg.score(X_test, y_test))
```

```
Accuracy of Training Data 0.9189925119128659
Accuracy of Testing Data 0.9126984126984127
```

Cross Validation is done to verify the accuracies of the model, after which the confusion matrix for both the train and test data is generated

```
# Cross Validation

from sklearn.model_selection import cross_val_score

X = df.loc[:, df.columns != 'user_type']
y = np.array(df['user_type'])
cross_val_score(log_reg, X_scale, y, cv = 5)
```

```
array([0.93586698, 0.92857143, 0.9452381 , 0.83054893, 0.92362768])
```

```
# Predict using the model

predict_train = log_reg.predict(X_train)
predict_test = log_reg.predict(X_test)
```

```
# Confusion Matrix to identify True Positives, True Negatives, False Positives and False Negatives

cm_train = confusion_matrix(y_train.ravel(), predict_train)
cm_test = confusion_matrix(y_test.ravel(), predict_test)
print('Training Accuracy: \n', cm_train)
print('')
print('Testing Accuracy: \n', cm_test)
```

```
Training Accuracy: 
 [[659  57]
 [ 62 691]]

Testing Accuracy: 
 [[281  19]
 [ 36 294]]
```

Finally, we plot the confusion matrices along with the accuracies for the full feature matrix

Principal Component Analysis (PCA)

Just like SVM, PCA is also implemented in logistic regression before retraining the data to get the new accuracy results followed by the confusion matrices.

```
logreg_train = log_reg.predict(X_train)
logreg_train_accuracy = sum(logreg_train==y_train.ravel())/len(y_train.ravel())
print('Accuracy of Training Data after PCA', logreg_train_accuracy)

print('Accuracy of Testing Data after PCA', log_reg.score(X_test, y_test))
```

```
Accuracy of Training Data after PCA 0.8917631041524847
Accuracy of Testing Data after PCA 0.8857142857142857
```

```
# Cross Validation

from sklearn.model_selection import cross_val_score

X = df.loc[:, df.columns != 'user_type']
y = np.array(df['user_type'])
cross_val_score(log_reg, X_pca, y, cv = 5)
```

```
array([0.91686461, 0.91666667, 0.92380952, 0.81861575, 0.90214797])
```

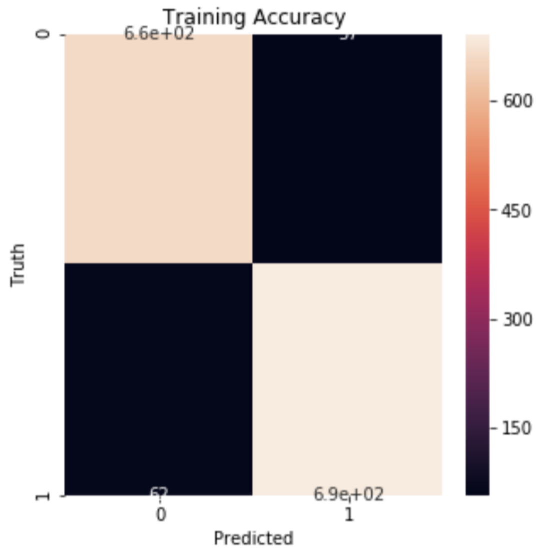
```
# Confusion Matrix to identify True Positives, True Negatives, False Positives and False Negatives

cm_train_pca = confusion_matrix(y_train.ravel(), predict_train)
cm_test_pca = confusion_matrix(y_test.ravel(), predict_test)
print('Training Accuracy: \n', cm_train_pca)
print('')
print('Testing Accuracy: \n', cm_test_pca)
```

```
Training Accuracy: 
 [[634  82]
 [ 77 676]]

Testing Accuracy: 
 [[273  27]
 [ 45 285]]
```

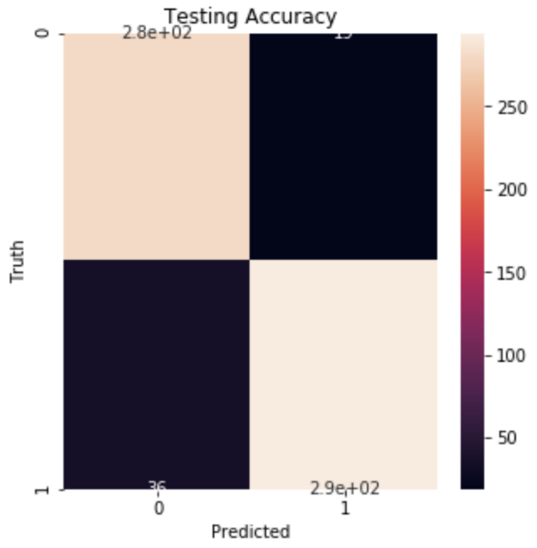
Analysis of Logistic Regression:

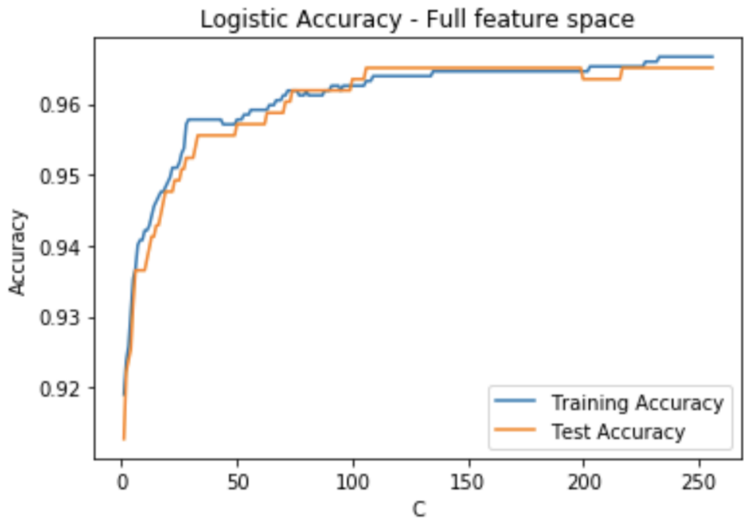
Visualization of confusion matrix along with the plot of accuracies for the feature matrix is shown below:

```
Text(0.5, 1, 'Training Accuracy')
```



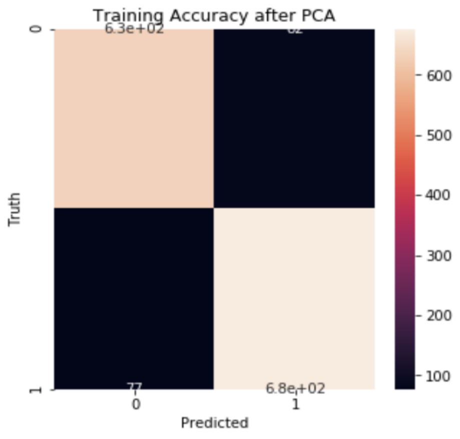
```
Text(0.5, 1, 'Testing Accuracy')
```

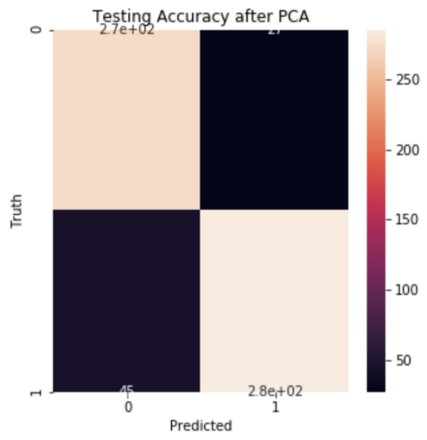


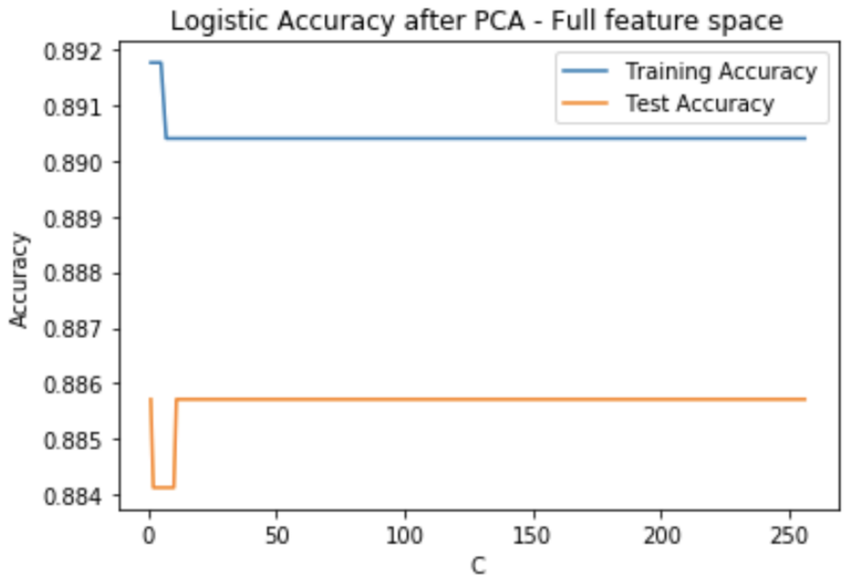


After performing the PCA and again retraining the model, the visualization of the confusion matrix and accuracy plot for the feature matrix is shown below:

```
Text(0.5, 1, 'Training Accuracy after PCA') Text(0.5, 1, 'Testing Accuracy after PCA')
```







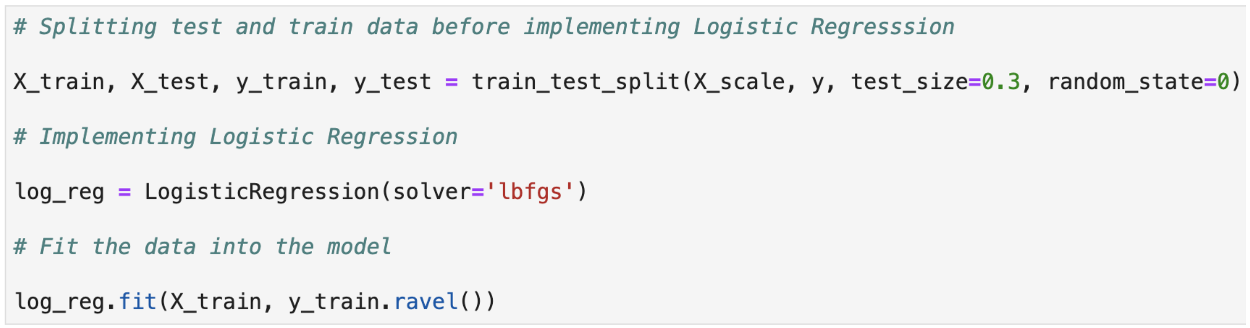
Random Forest

Decision trees are the most powerful and popular tool for classification and prediction. A Decision tree is a flowchart like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label. Decision trees classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance. An instance is classified by starting at the root node of the tree,testing the attribute specified by this node,then moving down the tree branch corresponding to the value of the attribute as shown in the above figure.This process is then repeated for the subtree rooted at the new node.

Random forest, like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction. The low correlation between models is the key. Just like how investments with low correlations (like stocks and bonds) come together to form a portfolio that is greater than the sum of its parts, uncorrelated models can produce ensemble predictions that are more accurate than any of the individual predictions. The reason for this wonderful effect is that the trees protect each other from their individual errors (as long as they don't constantly all err in the same direction). While some trees may be wrong, many other trees will be right, so as a group the trees are able to move in the correct direction. So the prerequisites for random forest to perform well are:

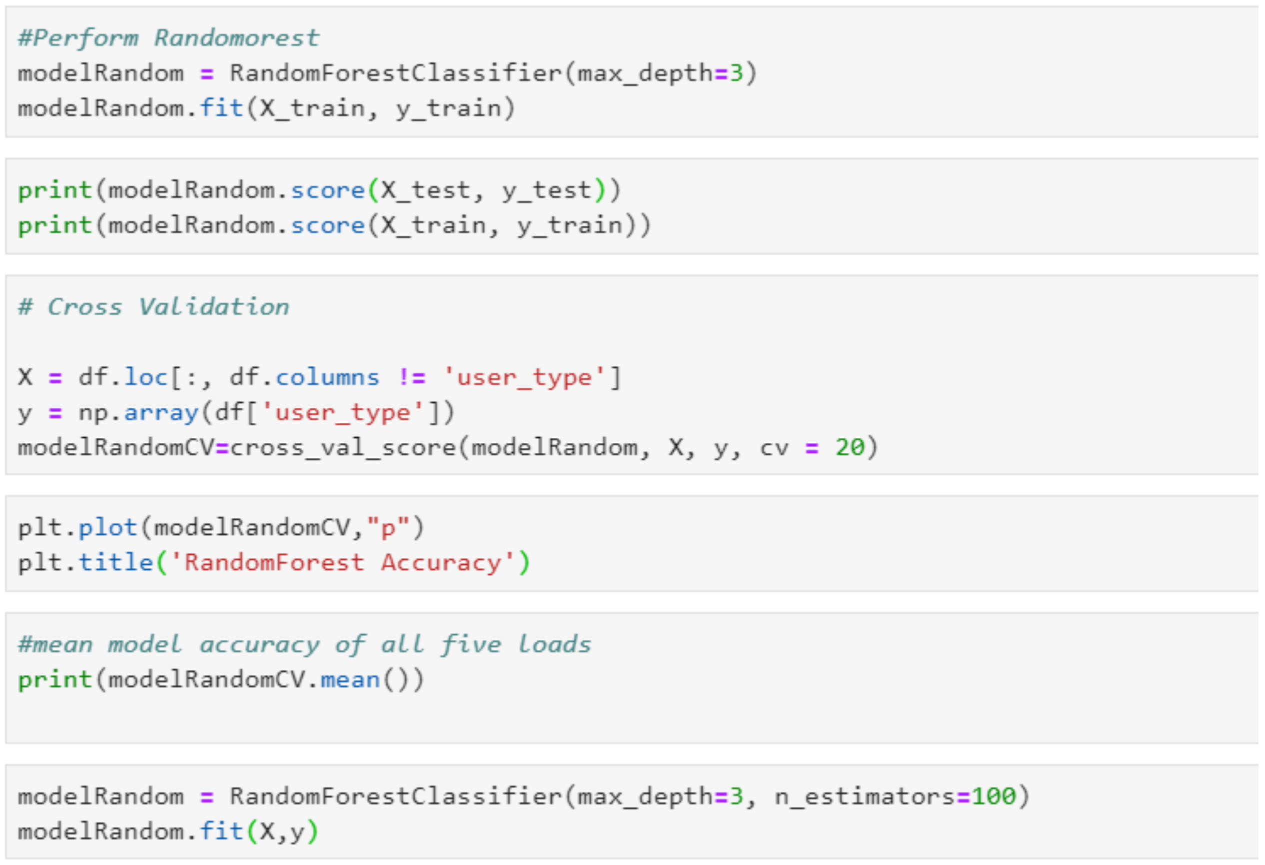
1. There needs to be some actual signal in our features so that models built using those features do better than random guessing.
2. The predictions (and therefore the errors) made by the individual trees need to have low correlations with each other.

Before training the data using Random Forest, we scale the data using MinMaxScaler similar to what we did before implementing SVM.



```
# Splitting test and train data before implementing Logistic Regresssion

X_train, X_test, y_train, y_test = train_test_split(X_scale, y, test_size=0.3, random_state=0)

# Implementing Logistic Regression

log_reg = LogisticRegression(solver='lbfgs')

# Fit the data into the model

log_reg.fit(X_train, y_train.ravel())
```

Cross Validation is done to verify the accuracies of the model, after which the confusion matrix for both the train and test data is generated



```
#Perform Randomorest
modelRandom = RandomForestClassifier(max_depth=3)
modelRandom.fit(X_train, y_train)
```

```
print(modelRandom.score(X_test, y_test))
print(modelRandom.score(X_train, y_train))
```

```
# Cross Validation

X = df.loc[:, df.columns != 'user_type']
y = np.array(df['user_type'])
modelRandomCV=cross_val_score(modelRandom, X, y, cv = 20)
```

```
plt.plot(modelRandomCV,"p")
plt.title('RandomForest Accuracy')
```

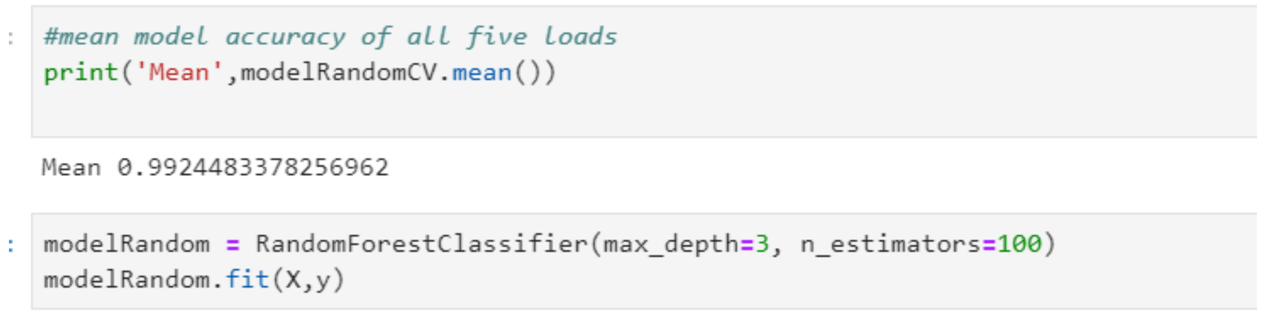
```
#mean model accuracy of all five loads
print(modelRandomCV.mean())
```

```
modelRandom = RandomForestClassifier(max_depth=3, n_estimators=100)
modelRandom.fit(X,y)
```

Testing Accuracy 0.9888888888888889

Train Accuracy 0.9965963240299524

Mean model accuracy of all five loads.



```
#mean model accuracy of all five loads
print('Mean',modelRandomCV.mean())
```

```
Mean 0.9924483378256962
```

```
modelRandom = RandomForestClassifier(max_depth=3, n_estimators=100)
modelRandom.fit(X,y)
```

```python
predict_train = modelRandom.predict(X_train)
predict_test = modelRandom.predict(X_test)
```

```python
from sklearn.metrics import confusion_matrix
cm_train = confusion_matrix(y_train.ravel(), predict_train)
cm_test = confusion_matrix(y_test.ravel(), predict_test)
print('Training Accuracy: \n', cm_train)
print('')
print('Testing Accuracy: \n', cm_test)
```

```
Training Accuracy: 
 [[711   5]
 [729  24]]

Testing Accuracy: 
 [[299   1]
 [325   5]]
```

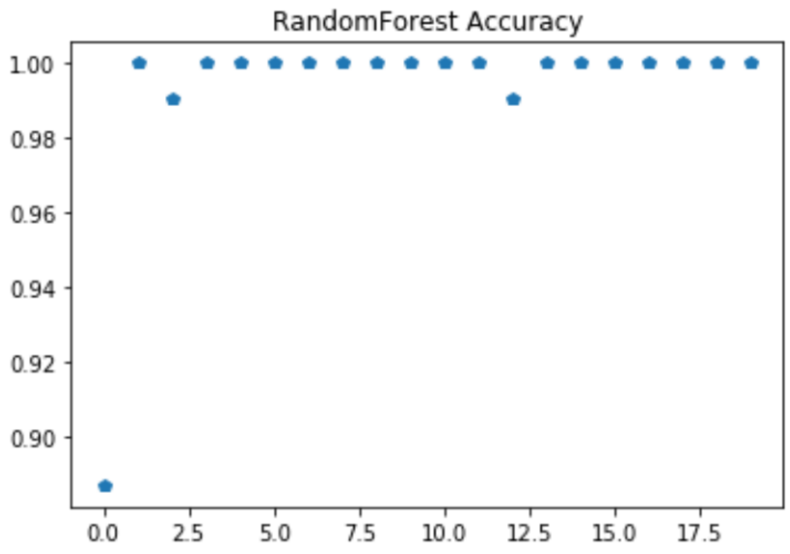
```python


import seaborn as sns
plt.figure(figsize = (5,5))
sns.heatmap(cm_train, annot = True)
plt.xlabel('Predicted')
plt.ylabel('Truth')
plt.title('Training Accuracy')
```

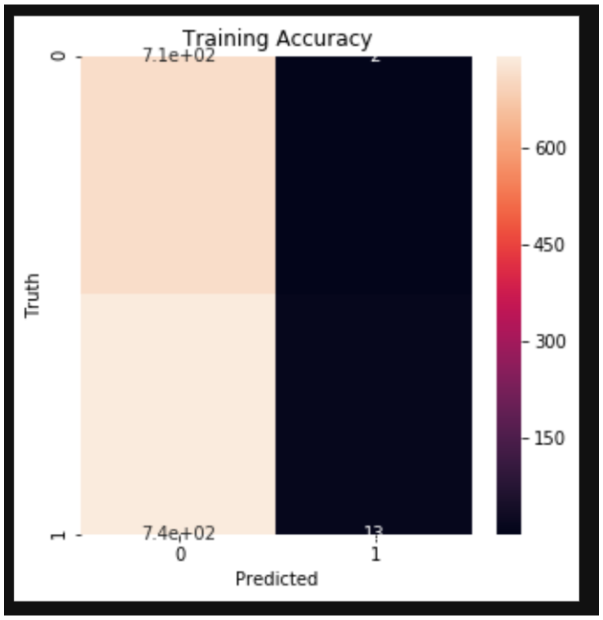
```
from sklearn.model_selection import validation_curve

param_range = np.arange(1, 250, 2)

train_scores, test_scores = validation_curve(RandomForestClassifier(),
                                             X,
                                             y,
                                             param_name="n_estimators",
                                             param_range=param_range,
                                             cv=3,
                                             scoring="accuracy",
                                             n_jobs=-1)
```

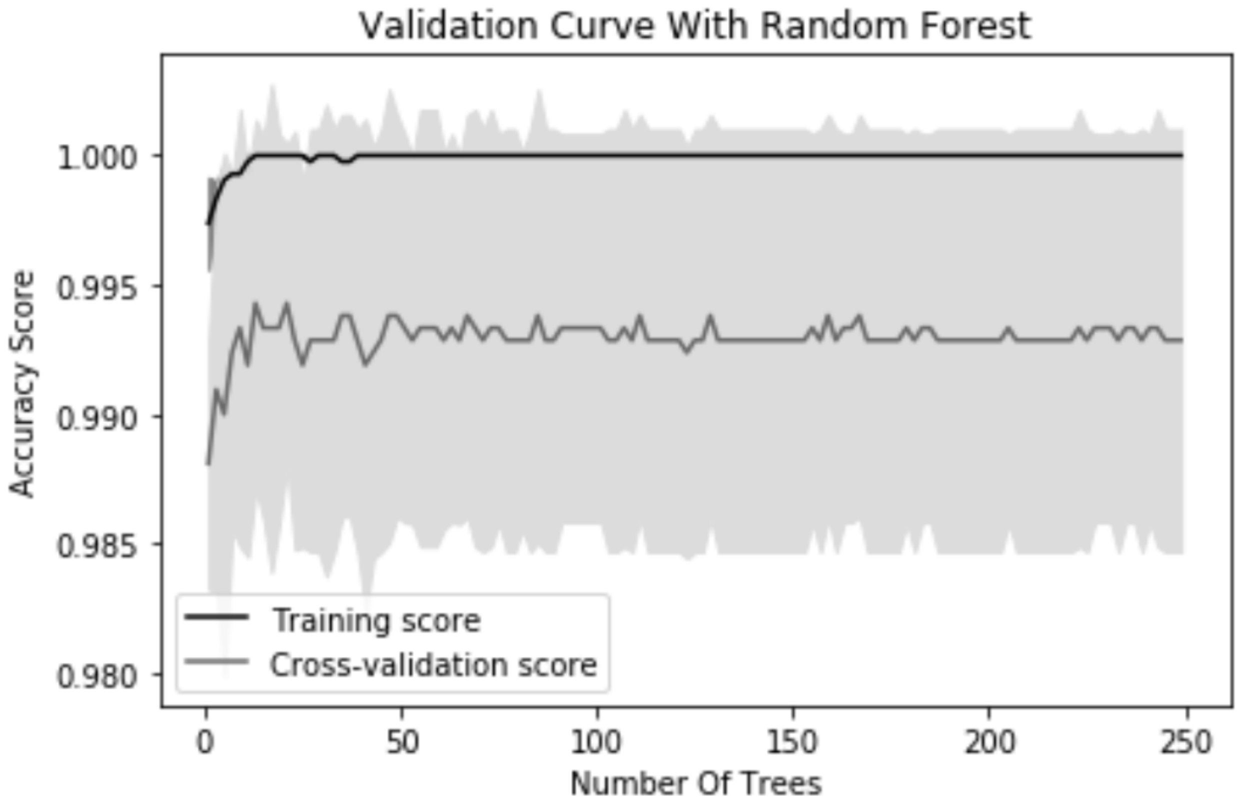
```
# Calculate mean and standard deviation for training set scores
train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)

# Calculate mean and standard deviation for test set scores
test_mean = np.mean(test_scores, axis=1)
test_std = np.std(test_scores, axis=1)

# Plot mean accuracy scores for training and test sets
plt.plot(param_range, train_mean, label="Training score", color="black")
plt.plot(param_range, test_mean, label="Cross-validation score", color="dimgrey")

# Plot accurancy bands for training and test sets
plt.fill_between(param_range, train_mean - train_std, train_mean + train_std, color="gray")
plt.fill_between(param_range, test_mean - test_std, test_mean + test_std, color="gainsboro")

# Create plot
plt.title("Validation Curve With Random Forest")
plt.xlabel("Number Of Trees")
plt.ylabel("Accuracy Score")
plt.tight_layout()
plt.legend(loc="best")
plt.show()
```

Analysis of Random Forest:

Visualization of the confusion matrix followed by the validation curve is presented in the below graphs:






Validation Curve With Random Forest
Accuracy Score
Number Of Trees
Training score
Cross-validation score
1.000
0.995
0.990
0.985
0.980
0
50
100
150
200
250

Result

Among our three selected models, **SVM with RBF kernel** resulted in the highest classification accuracy. Overall, using a lower-dimension feature space had the expected effect of reducing computation time, but with one exception resulted in poorer classification accuracy. Results are summarized in following table:

Classifier Model	Full feature space(n=13)		PCA(n=3)	
	Train accuracy%	Test accuracy%	Train accuracy%	Test accuracy%
Support Vector Machine				
Linear	97.88	98.73	90.26	91.11
RBF	99.52	99.20	96.86	96.34
Polynomial	99.45	98.88	90.74	91.74
Logistic Regression	91.89	91.26	89.17	88.57
Random Forest	98.65	99.88	96.75	91.43

Conclusion:

Based on the above analysis and results following are the conclusions:

- The most accurate machine learning model is **SVM with RBF kernel**, with the training accuracy of 96.34**%** after **PCA**.
- The general observation after implementing PCA is that the accuracy of the prediction reduces with just a few exceptions.

Future Scope

Following are the future scope of this project:

- This approach can be used for other social media platforms to identify bots websites and ensure the safety and privacy of the users.
- This can be used to automate things and reduce load on these social media sites.