Twitter Bot analysis using Machine Learning

Akshay Hazare
Computer Science
ah4093@nyu.edu

Gaurang Gatlewar
Computer Science
gvg228@nyu.edu

Shyam Nair
Computer Science
smn387@nyu.edu

INTRODUCTION

TWITTER is a popular online social networking and microblogging tool, which was released in 2006. Remarkable simplicity is its distinctive feature. Its community interacts via publishing text-based posts, known as tweets. The tweet size is limited to 140 characters. Hashtag, namely words or phrases prefixed with a # symbol, can group tweets by topic. For example, #Justin Bieber and #Women's World Cup are the two trending hashtags on Twitter in 2015. Symbol @ followed by a username in a tweet enables the direct delivery of the tweet to that user. Unlike most online social networking sites (i.e., Facebook and MySpace), Twitter's user relationship is directed and consists of two ends, friend and follower.

MOTIVATION

As reported in August 2011, Twitter has attracted 200 million users and generated 8.3 million Tweets per hour. It ranks the 10th on the top 500 site list according to Alexa in December 2011. k. On the other hand, malicious bots have been greatly exploited by spammers to spread spam. The definition of spam in this paper is spreading malicious, phishing, or unsolicited commercial content in tweets. These bots randomly add users as their friends, expecting a few users to follow back.1 In this way, spam tweets posted by bots display on users' homepages. Enticed by the appealing text content, some users may click on links and get redirected to spam or malicious sites.2 If human users are surrounded by malicious bots and spam tweets, their twittering experience deteriorates, and eventually the whole Twitter community will be hurt. The objective of this paper is to characterize the automation feature of Twitter accounts, and to classify them into two categories, human and bot.

RELATED WORK

In previous studies on the topic, Twitter REST APIs were used for collecting data such as the friends and followers count. In the study conducted by Erin Shellman, GridSearchCV was used to optimize feature selection, which boosted the accuracy by 2%. In some studies conducted during the DARPA Twitter Bot Challenge, the top 3 ranking teams performed a semi-automated classifying approach including profile feature analysis, sentiment and syntax analysis and behavioural analysis.

DATA

We have analyzed and categorized multiple profiles to create the data set of submission. In our data.csv we have profiles of which about half are bots and others are humans.
We used multiple parameters to classify a profile:

1. Followers Count
2. Friends Count
3. Favourites Count
4. Statuses Count
5. Verified
6. Has extended profile
7. Location
8. If or not the user has 'bot' or random string in their nametag eg. @hellobot or @hxj48jfj9.
9. If or not the user has a profile picture URL and description.
10. If or not the name of the profile and the name

in the URL are different.

ALGORITHMS

The algorithms that were used before the final shortlist included:

1. Bernoulli Naive Bayes
2. Multinomial Naive Bayes
3. Support Vector Machine
4. Decision tree
5. Random Forest
6. Gradient Boost
7. Logistic Regression

These algorithms were used for training and testing on the available data. Comparing the results of the above algorithms, Support Vector Machine, Bernoulli and Multinomial Naive Bayes Classifiers were discarded due to the relatively low classification accuracy on the given data.

A brief introduction on the chosen algorithms is included below:

Gradient Boosting Classifier

Gradient Boosting Classifier produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.

Decision Tree Classifier

Decision tree learning uses a decision tree as a predictive model which maps observations about an item (represented in the branches) to conclusions about the item's target value (represented in the leaves). It is one of the predictive modelling approaches used in statistics, data mining and machine learning. Tree models where the target variable can take a finite set of values are called classification trees; in these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. Decision trees where the target variable can take continuous values (typically real numbers) are called regression trees.

Random Forest Classifier

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of over fitting to their training set.

Logistic Regression

In statistics, logistic regression, or logit regression, or logit model is a regression model where the dependent variable (DV) is categorical. This article covers the case of a binary dependent variable-that is, where it can take only two values, "0" and "1", which represent outcomes such as pass/fail, win/lose, alive/dead or healthy/sick. Cases where the dependent variable has more than two outcome categories may be analysed in multinomial logistic regression, or, if the multiple categories are ordered, in ordinal logistic regression. In the terminology of economics, logistic regression is an example of a qualitative response/discrete choice model.

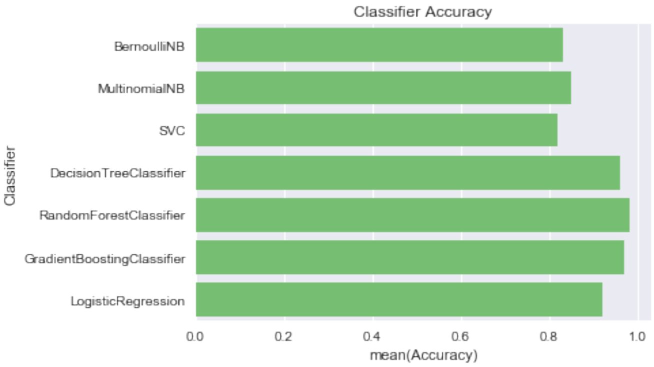
After shortlisting the algorithms, the GridSearchCV algorithm was used to improve the fitting accuracy.

GridSearchCV

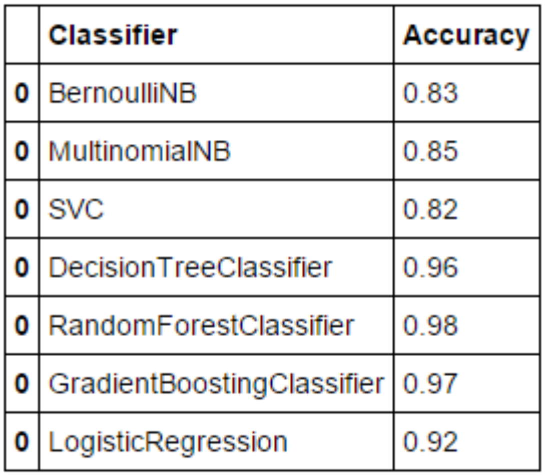
GridSearchCV implements a “fit” and a “score” method. It also implements “predict”, “predict_proba”, “decision_function”, “transform” and “inverse_transform” if they are implemented in the estimator used. The parameters of the estimator used to apply these methods are optimized by cross-validated grid-search over a parameter grid.

RESULT

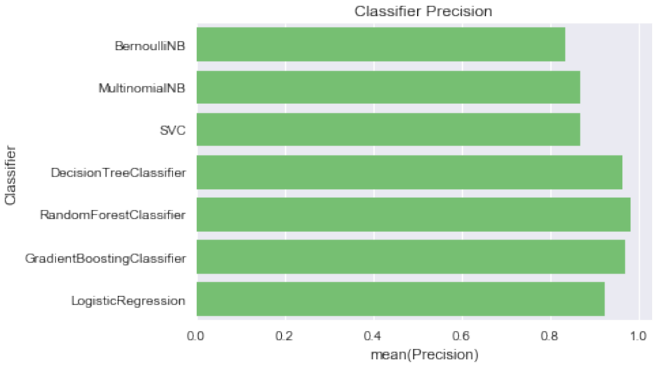
The 7 algorithms were used to train and test data and were further tested on to check the performance of each. The parameters that were used to compare the performance of the algorithms included accuracy, precision, recall, F1 and AUC scores. The accuracy score plot has been shown below:

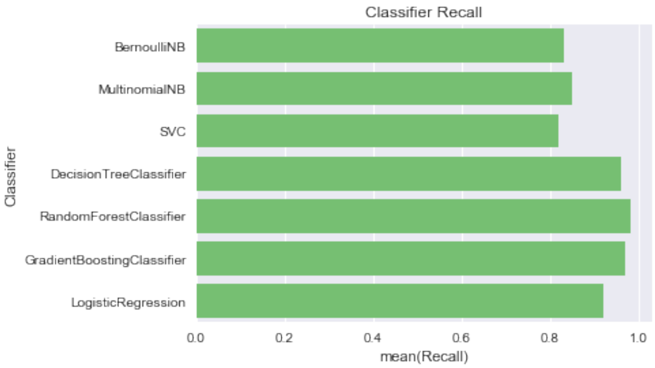


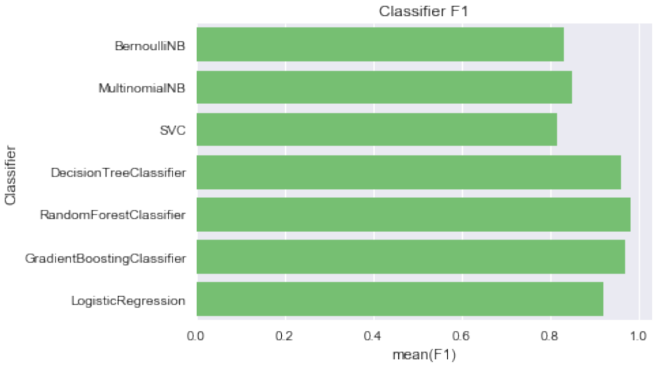
As it can be seen clearly from the plot as well as the table, the other 4 algorithms have superior classification performance compared to SVM and Bernoulli NB and Multinomial NB. Thus, the other 4 algorithms were chosen for further training. The results of all of these were very similar on accuracy and other parameters (precision, recall, F1 and AUC) as can be seen from the following plots:

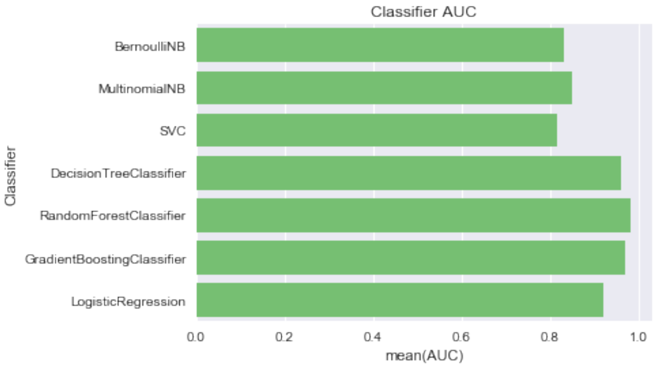


	Classifier	Accuracy
0	BernoulliNB	0.83
0	MultinomialNB	0.85
0	SVC	0.82
0	DecisionTreeClassifier	0.96
0	RandomForestClassifier	0.98
0	GradientBoostingClassifier	0.97
0	LogisticRegression	0.92





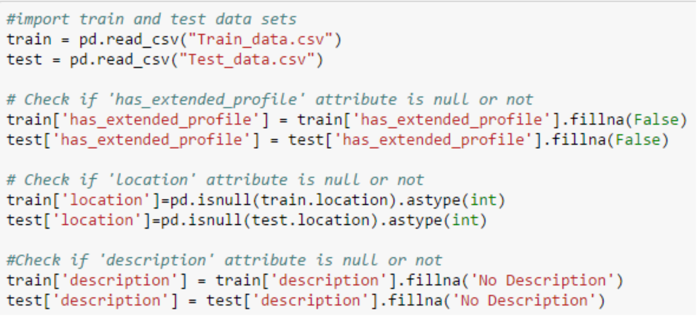




CODE

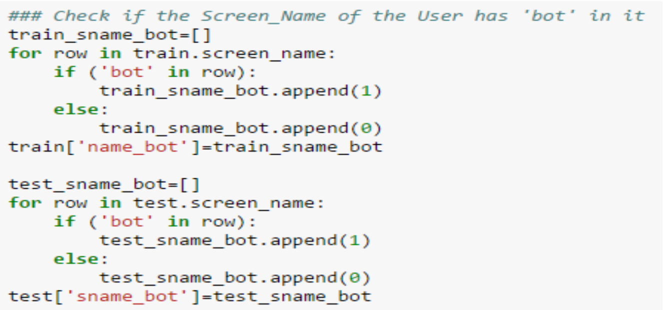
The entire code and other files can be accessed on GitHub : https://github.com/AkshayHazare/Twitter_Bot_Detection

The data was imported and cleaned before training. Many of the fields such as location and description contained multiple empty fields and were replaced by appropriate values as required.

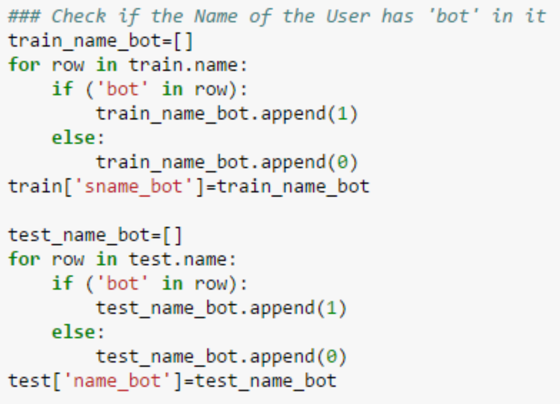


```
#import train and test data sets
train = pd.read_csv("Train_data.csv")
test = pd.read_csv("Test_data.csv")

# Check if 'has_extended_profile' attribute is null or not
train['has_extended_profile'] = train['has_extended_profile'].fillna(False)
test['has_extended_profile'] = test['has_extended_profile'].fillna(False)

# Check if 'location' attribute is null or not
train['location']=pd.isnull(train.location).astype(int)
test['location']=pd.isnull(test.location).astype(int)

#Check if 'description' attribute is null or not
train['description'] = train['description'].fillna('No Description')
test['description'] = test['description'].fillna('No Description')
```

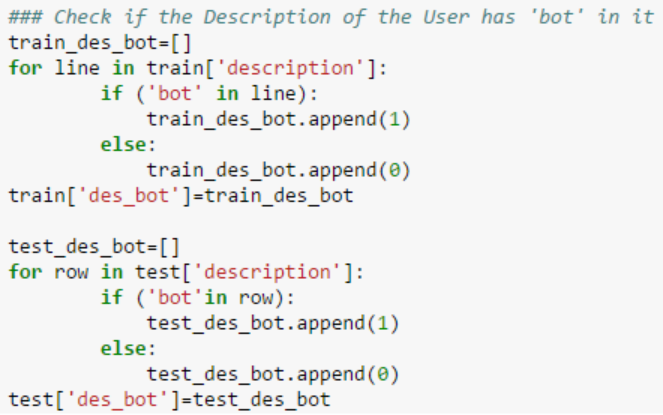
A column was appended to the DataFrame which was used to classify on the basis of the name, screen name or description attribute values containing the phrase “bot”. This was one of the attributes used for bot classification later.



```
### Check if the Screen_Name of the User has 'bot' in it
train_sname_bot=[]
for row in train.screen_name:
    if ('bot' in row):
        train_sname_bot.append(1)
    else:
        train_sname_bot.append(0)
train['name_bot']=train_sname_bot

test_sname_bot=[]
for row in test.screen_name:
    if ('bot' in row):
        test_sname_bot.append(1)
    else:
        test_sname_bot.append(0)
test['sname_bot']=test_sname_bot
```

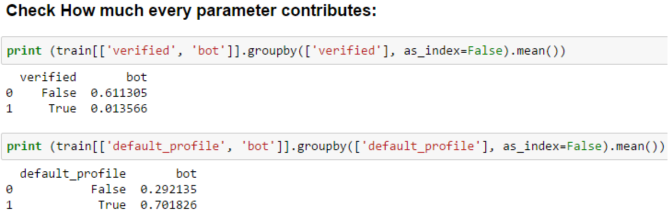


```
### Check if the Name of the User has 'bot' in it
train_name_bot=[]
for row in train.name:
    if ('bot' in row):
        train_name_bot.append(1)
    else:
        train_name_bot.append(0)
train['sname_bot']=train_name_bot

test_name_bot=[]
for row in test.name:
    if ('bot' in row):
        test_name_bot.append(1)
    else:
        test_name_bot.append(0)
test['name_bot']=test_name_bot
```



```
### Check if the Description of the User has 'bot' in it
train_des_bot=[]
for line in train['description']:
        if ('bot' in line):
            train_des_bot.append(1)
        else:
            train_des_bot.append(0)
train['des_bot']=train_des_bot

test_des_bot=[]
for row in test['description']:
        if ('bot'in row):
            test_des_bot.append(1)
        else:
            test_des_bot.append(0)
test['des_bot']=test_des_bot
```

Each attribute was individually tested to determine its impact on the final bot classification and was used as a measure to decide which parameters to include in the classification process. The attributes used were 'followers_count', 'friends_count', 'favourites_count', 'statuses_count', 'verified', 'default_profile', 'default_profile_image', 'has_extended_profile', 'location', 'des_bot' and 'name_bot'.



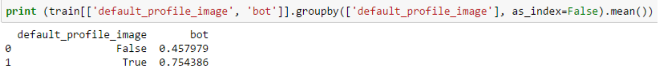
Check How much every parameter contributes:

```
print (train[['verified', 'bot']].groupby(['verified'], as_index=False).mean())
```

```
  verified       bot
0    False  0.611305
1     True  0.013566
```

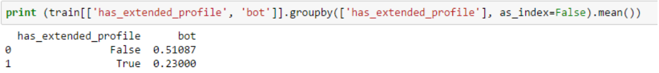
```
print (train[['default_profile', 'bot']].groupby(['default_profile'], as_index=False).mean())
```

```
  default_profile       bot
0           False  0.292135
1            True  0.701826
```



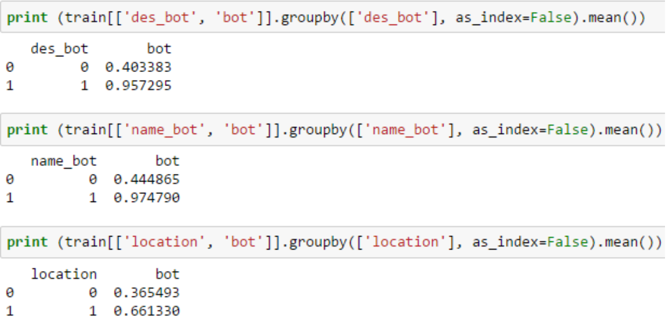
```
print (train[['default_profile_image', 'bot']].groupby(['default_profile_image'], as_index=False).mean())
```

```
  default_profile_image       bot
0                 False  0.457979
1                  True  0.754386
```



```
print (train[['has_extended_profile', 'bot']].groupby(['has_extended_profile'], as_index=False).mean())
```

```
  has_extended_profile      bot
0                False  0.51087
1                 True  0.23000
```



```
print (train[['des_bot', 'bot']].groupby(['des_bot'], as_index=False).mean())
```

```
   des_bot       bot
0        0  0.403383
1        1  0.957295
```

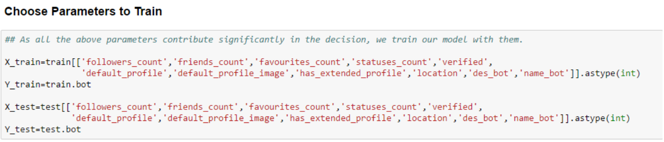
```
print (train[['name_bot', 'bot']].groupby(['name_bot'], as_index=False).mean())
```

```
   name_bot       bot
0         0  0.444865
1         1  0.974790
```

```
print (train[['location', 'bot']].groupby(['location'], as_index=False).mean())
```

```
   location       bot
0         0  0.365493
1         1  0.661330
```

The training and testing data was created by using the mentioned attributes from the data



Choose Parameters to Train

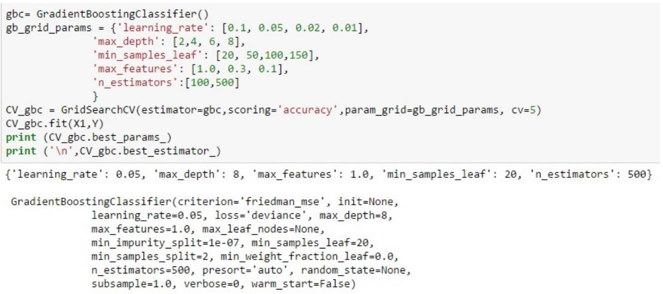
```
## As all the above parameters contribute significantly in the decision, we train our model with them.

X_train=train[['followers_count','friends_count','favourites_count','statuses_count','verified',
               'default_profile','default_profile_image','has_extended_profile','location','des_bot','name_bot']].astype(int)
Y_train=train.bot

X_test=test[['followers_count','friends_count','favourites_count','statuses_count','verified',
             'default_profile','default_profile_image','has_extended_profile','location','des_bot','name_bot']].astype(int)
Y_test=test.bot
```

The various classifiers were used for classification and were compared amongst themselves using the performance characteristics mentioned in Results to choose the algorithms to be used for final Bot prediction. Based on all these attributes and highest accuracy obtained which is 88%, we plan to choose Random Forest Classifier, Decision Tree Classifier, Gradient Boosting Classifier and Logistic Regression Classifier our final classifiers.

Finally, in one of the approaches, the predictions were made by hybridizing the prediction results of all the four chosen algorithms. It was done by using the four algorithms and final prediction was a result of the majority vote of the four algorithms.

In another approach to predict, GridSearchCV was used to fine-tune the predictions for the chosen algorithms by optimizing feature selection.

GridSearchCV with Gradient Boost Classifier

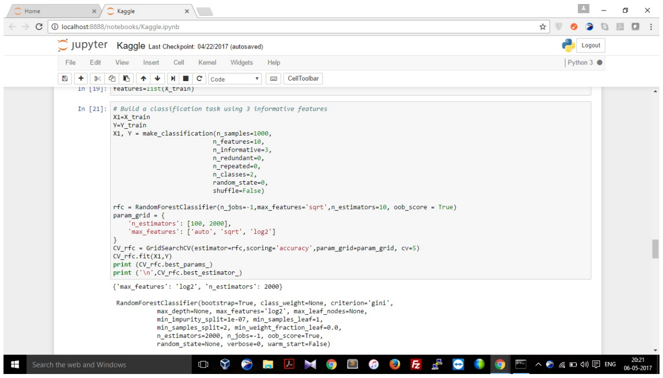


```
gbc= GradientBoostingClassifier()
gb_grid_params = {'learning_rate': [0.1, 0.05, 0.02, 0.01],
              'max_depth': [2,4, 6, 8],
              'min_samples_leaf': [20, 50,100,150],
              'max_features': [1.0, 0.3, 0.1],
              'n_estimators':[100,500]
              }
CV_gbc = GridSearchCV(estimator=gbc,scoring='accuracy',param_grid=gb_grid_params, cv=5)
CV_gbc.fit(X1,Y)
print (CV_gbc.best_params_)
print ('\n',CV_gbc.best_estimator_)
```

```
{'learning_rate': 0.05, 'max_depth': 8, 'max_features': 1.0, 'min_samples_leaf': 20, 'n_estimators': 500}

 GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.05, loss='deviance', max_depth=8,
              max_features=1.0, max_leaf_nodes=None,
              min_impurity_split=1e-07, min_samples_leaf=20,
              min_samples_split=2, min_weight_fraction_leaf=0.0,
              n_estimators=500, presort='auto', random_state=None,
              subsample=1.0, verbose=0, warm_start=False)
```

By using GridSearch CV on GBC we found that loading the model with learning rate= 0.05, max_depth=3 and n_estimators =500 would yield optimum results on our Data.

GridSearchCV with Random Forest Classifier



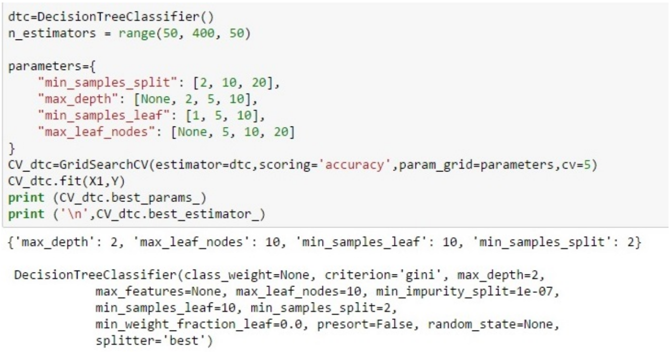
Home
Kaggle
localhost:8888/notebooks/Kaggle.ipynb
jupyter
Kaggle Last Checkpoint: 04/22/2017 (autosaved)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Python 3
Code
CellToolbar
Search the web and Windows
ENG
20:21
06-05-2017

```
In [19]: features=list(X_train)

In [21]: # Build a classification task using 3 informative features
         X1=X_train
         Y=Y_train
         X1, Y = make_classification(n_samples=1000,
                                     n_features=10,
                                     n_informative=3,
                                     n_redundant=0,
                                     n_repeated=0,
                                     n_classes=2,
                                     random_state=0,
                                     shuffle=False)

         rfc = RandomForestClassifier(n_jobs=-1,max_features='sqrt',n_estimators=10, oob_score = True)
         param_grid = {
             'n_estimators': [100, 2000],
             'max_features': ['auto', 'sqrt', 'log2']
         }
         CV_rfc = GridSearchCV(estimator=rfc,scoring='accuracy',param_grid=param_grid, cv=5)
         CV_rfc.fit(X1,Y)
         print (CV_rfc.best_params_)
         print ('\n',CV_rfc.best_estimator_)
```

```
{'max_features': 'log2', 'n_estimators': 2000}

 RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='log2', max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=2000, n_jobs=-1, oob_score=True,
            random_state=None, verbose=0, warm_start=False)
```

By using GridSearch CV on RF we found that loading the model with max_features = ‘log2’ and n_estimators = 2000 would yield optimum results on our Data.

GridSearchCV with Decision Tree Classifier.

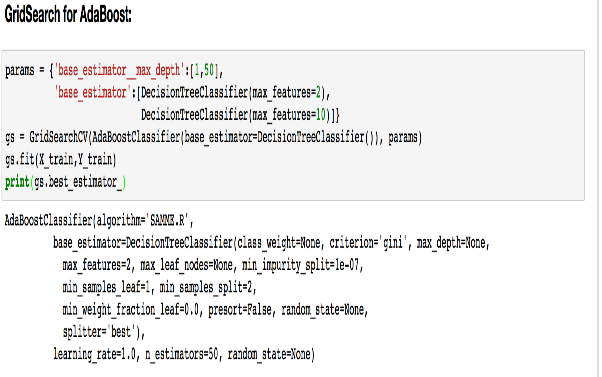


```
dtc=DecisionTreeClassifier()
n_estimators = range(50, 400, 50)

parameters={
    "min_samples_split": [2, 10, 20],
    "max_depth": [None, 2, 5, 10],
    "min_samples_leaf": [1, 5, 10],
    "max_leaf_nodes": [None, 5, 10, 20]
}
CV_dtc=GridSearchCV(estimator=dtc,scoring='accuracy',param_grid=parameters,cv=5)
CV_dtc.fit(X1,Y)
print (CV_dtc.best_params_)
print ('\n',CV_dtc.best_estimator_)
```

```
{'max_depth': 2, 'max_leaf_nodes': 10, 'min_samples_leaf': 10, 'min_samples_split': 2}

 DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=2,
            max_features=None, max_leaf_nodes=10, min_impurity_split=1e-07,
            min_samples_leaf=10, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
```

By using GridSearch CV on RF we found that loading the model with max_depth = 2, max_leaf_node=10,min_samples_leaf=10, min_samples_split=2 would yield optimum results on our Data.

GridSearch CV with AdaBoost Classifier:



GridSearch for AdaBoost:

```
In [14]: params = {'base_estimator__max_depth':[1,50],
                   'base_estimator':[DecisionTreeClassifier(max_features=2),
                                     DecisionTreeClassifier(max_features=10)]}
         gs = GridSearchCV(AdaBoostClassifier(base_estimator=DecisionTreeClassifier()), params)
         gs.fit(X_train,Y_train)
         print(gs.best_estimator_)
```

```
AdaBoostClassifier(algorithm='SAMME.R',
          base_estimator=DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=2, max_leaf_nodes=None, min_impurity_split=1e-07,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best'),
          learning_rate=1.0, n_estimators=50, random_state=None)
```

By using GridSearch CV on AdaBoost we get the optimum AdaBoost model for our data.

FURTHER WORK

There are other improvements possible which have currently not been implemented to improve the prediction of the code. Currently only limited number of attributes have been used for classification. Further few more attributes can be used to improve the process as follows:

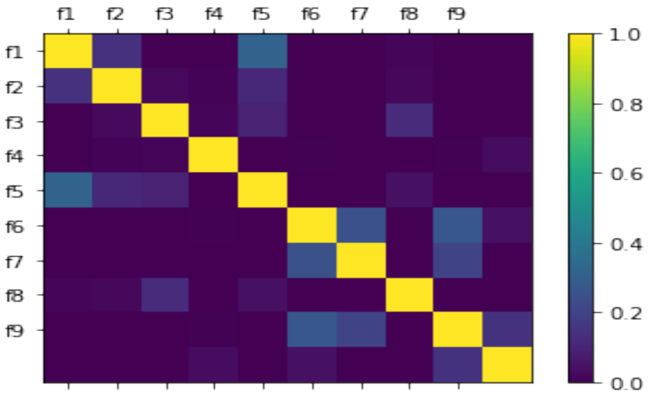
1. Status count rate can be used for prediction.
2. NLP can be performed on the text fields like description, statuses, etc.
3. NLP on description
4. Larger amount of Data with greater variation can be used to train the model.
5. Syntax and sentiment analysis can be performed.
6. Behaviour Analysis can be used to determine bot profiles.

CONCLUSION

The prediction scores of the various models can be seen to be as follows:

Model	Cross Val Accuracy
Random Forest	0.9002516
Gradient Boost	0.90016802
Decision Tree	0.87235855
SVM	0.72717485
AdaBoost Classifier	0.88629999

The effectiveness of the features in prediction are in the order mentioned below and can be seen from the correlation matrix as follows:



Also, the best way to choose Models is by running an optimization algorithm like GridSearchCV which helps better fit the data.

REFERENCES

1. Shellman, Erin. Bot or not. http://www.erinshellman.com/bot-or-not/
2. Ferrara, et. Al. https://arxiv.org/abs/1407.5225
3. Who Will Retweet This? Automatically Identifying and Engaging Strangers on Twitter to Spread Information. https://arxiv.org/ftp/arxiv/papers/1405/1405.3750.pdf
4. http://jpdickerson.com/pubs/dickerson14using.pdf
5. DARPA twitter bot challenge https://arxiv.org/ftp/arxiv/papers/1601/1601.05140.pdf