Twitter Account - Bot or Not

Vatsal Biren Gopani
Tandon School of Engineering, NYU
New York, USA
vbg221@nyu.edu

Manthan Mitesh Shah
Tandon School of Engineering, NYU
New York, USA
mms853@nyu.edu

Abstract—Predict accuracy of K-Nearest Neighbor algorithm and Random Forest algorithm on predicting whether a Twitter Account is a computer operated (bot) account or not.

Keywords—machine learning, classification, Random Forest, k-NN, accuracy

I. INTRODUCTION

The basic idea of this project is to successfully identify twitter bots using machine learning algorithms. After our primary research, we came up with the conclusion that using the data fetched from twitter, we can use K-nearest neighbor and Random Forest algorithm for more than fairly decent accuracy. How to reduce error rate will be the problem of optimizing model by preprocessing the testing data. Optimization will be introduced after initial implementation. We are planning to use followers count, retweeted count, friends count, analyses of all the tweets by user, and few other factors to get the initial result.

II. MOTIVATION

Twitter bots can be the cause of big change triggered without any notice and knowledge of any human. The fact that people believe the authenticity of news and comments from other twitter user is the problem here. If we can find which are the bots operating as human user, we can monitor and keep track of unwanted tweets and reduce the damage it can cause to the world. Also, finding these smart bots is a great way to learn and explore our knowledge of machine learning.

III. RELATED WORK

We used many research papers to get idea about which features are strong and which model can get us better results. These researches have proven to be groundbreaking at the time of their release. But the bots are becoming smarter day by day as well. You can get the details of those researches from the links given below.

1. Ferrara, et. Al. The Rise of Social Bots.
2. Lee, et al. Who Will Retweet This? Automatically Identifying and Engaging Strangers on Twitter to Spread Information.
3. Shellman, Erin. Bot or not.

IV. DATA

We obtain preliminary training dataset from the social networking portal - Twitter. The critical work in doing so was to obtain data (twitter accounts) for actual bots as we did not have any algorithm at the moment that can define with confidence whether an account as a bot or not. So, we extensively went through online articles[1] to find data on twitter accounts which are bots, explicitly created by humans, performing specific automated tasks using online resources. The bot accounts gathered by us are legitimate computerized programs which fetch materials through internet and send out a tweet using those materials in real time. Using 100 accounts (50 bot accounts and 50 not), we contributed to a larger pool of data of 2797 accounts out of which 1321 are bot accounts while the rest 1476 are accounts handled by humans. Now, this data is collected from number of sources, whose authenticity is questionable. But regardless, this pool of data is used to train our algorithms. The data consists of 2797 records each having 20 parameters, out of which multiple parameters of multiple records contain useful data, NaN, None or blank values.

V. ALGORITHM(S) USED

As the problem is a classification problem where we have to predict, from the values provided from 20 parameters, whether a given record should be considered a bot or not, we used two machine learning approaches in solving it. The algorithms used to devise the machine learning model are:

- K-nearest neighbor algorithm
- Random Forest Trees algorithm.

Both of these approaches work exclusively to derive a decision whether to consider a given account as a bot or not based on multiple parameters on which the model is being trained. The accuracy provided in the result is the accuracy of the entire testing dataset on how effectively the model predicts a bot account as a bot and vice-a-versa. From initial research, it is found that these are the algorithms more suitable for the given classification problem.

k-Nearest Neighbor Algorithm

In k-NN classification, the output is a classification decision result. An instance is classified based on majority vote of its neighbors, with the instance being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor. Now the neighbors are calculated on a 2-D graph space using distance formula. There are multiple ways in which the distance can be calculated, but for this project Euclidean distance has been used to calculate the distance between the instance and the neighbors. Now, these distances are stored in a list and sorted in ascending order. Based on the value of k decided, the first k-distances are selected from the list and the instance is assigned a class based on the majority of neighboring classes in the first k-distance list. An inbuilt library method is used to perform all the specified tasks. Pandas framework and SciKit.learn's library packages are used to implement the K-nearest neighbor algorithm in JuPyter environment to train the model. Before training the model on the data, the data has to be cleaned so that the model can use it to train itself. First the data is read and all the NaN values are replaced with empty strings. Then the data is passed through cleaning function, which can be viewed from my github link provided. The data returned from the cleaning function contain normalized continuous values and categorical values for parameters: 'screen_name', 'location', 'description', 'listedcount', 'friends_count', 'followers_count', 'favorites_count', 'verified', 'statuses_count', 'default_profile', 'default_profile_image', 'has_extended_profile', 'name', 'bot'. Next, KNN model is fitted for neighbor values 1-50 and accuracy of each prediction is noted. A graph is plotted for neighbor vs AUC score. We achieve maximum accuracy when we fit KNN model using 12 neighbors. To better fit the model, 10-Fold Cross Validation is performed and accuracy scores are noted and plotted vs neighbor value. After CV max accuracy is achieved at neighbor value 6. After noting the max accuracy neighbor value, the model is refitted for testing purpose and accuracy at each fold is calculated and plotted. Results are calculated which include: Confusion Matrix, Classification report containing - Precision, Recall, F-1 Score and ROC curve.

Random Forest Algorithm

In Random Forest algorithm, multiple decision trees are created when the model is trained. These multiple decision trees are the major cause of the high accuracy of Random Forest algorithm. Due to these trees, the algorithm is guaranteed not to overfit. Moreover, the average result of these trees gives high accuracy as decision trees are used to give. For our project, we have used tweeter data with 20 fields. Much of this data was unnecessary. The test conducted by other researchers who worked on the same problem before, have said in their research papers that only few fields yield significant information gain which can lead to the detection of bot account. Hence, I have filtered the data and kept only 6 fields that I found good enough to give the accuracy I need.

For implementing this model, I have used SciKit package available in python. From the whole dataset, I have filtered unnecessary data and from the remaining data of 2797 accounts to train my Random Forest algorithm with number of estimators 100. As I have explained in results, I have used 6 most significant features for my dataset to train the model and hence, I have acquired 93.5% of training accuracy.

VI. RESULTS

k-NN Results:

After cleaning is performed and the data is obtained in a format which can be used to train the model, the data is split into training and testing sets in 80:20 ratio. The k-NN model is trained on 80% of the data (training set) and after the model is trained, the model is tested on the rest 20% of the data (testing set).

During testing, the model is trained for each k (neighbor) value and its false positive rate, true positive rates and area under the curve (AUC) scores are calculated. The AUC scores are stored and are plotted against k-neighbor values.

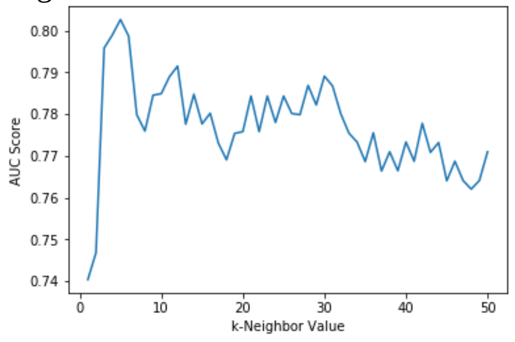


Figure 1- K-neighbor vs AUC score

To improve the decision power of the model 10-Fold cross validation is performed and model is fitted and accuracy is calculated for each neighbor. The resultant accuracy vs neighbor graph is plotted.

```
Maximum accuracy achieved at Neighbor value(k): 6
Mean Accuracy: 0.842895972009
```

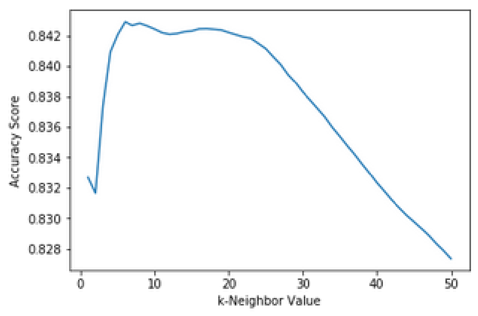


Figure 2- Neighbor value vs Accuracy score

Maximum accuracy is achieved at neighbor value 6, 84.289%. After this the model is trained for neighbor value 6 and following results are calculated:

Confusion Matrix

- **True Positive (TP) : 1293**
- **True Negative (TN) : 1076**
- **False Positive (FP) : 246**
- **False Negative (FN) : 183**

Based the values obtained above we calculate precision and recall of the model for each class:

```
Classification report:
             precision    recall  f1-score   support

          0       0.84      0.88      0.86      1476
          1       0.85      0.81      0.83      1321

avg / total       0.85      0.85      0.85      2797
```

Figure 3 – Classification Report

ROC curve is plotted for neighbor value 6 (when the accuracy score was maximum).

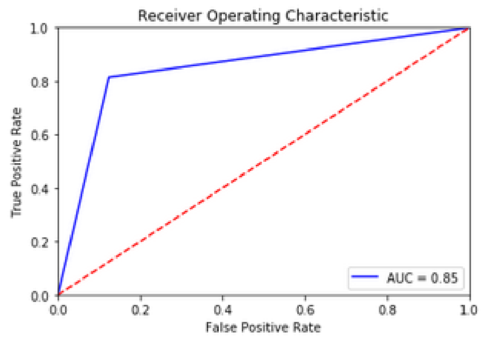


Figure 4 – ROC Curve

Random Forest Results:
After cleaning is performed, the data is randomly split into 80:20 ratio for training and testing accordingly. The Random Forest algorithm then uses the 80% training data to train the model and then fits the model on the remaining 20% dataset. Outcome of this is the predictions from Random Forest algorithm about the testing dataset. This algorithm gives fairly accurate result of 93%

For the final test on the kaggle, we used the whole training data to fit the model using 6 features. These are the features that were most significant. We performed Feature Significance Analysis to get these features. Anything less than 6 and the accuracy drops 4-5% and anything more will turn into overfitting, again decreasing the testing accuracy by 2-3%.

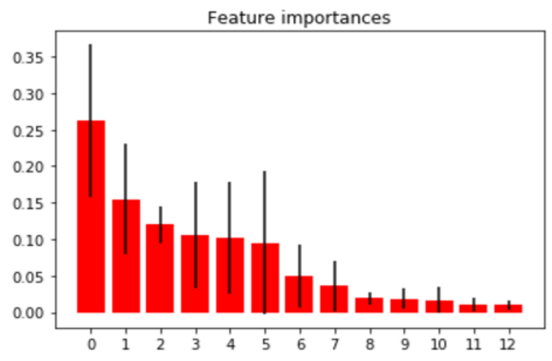
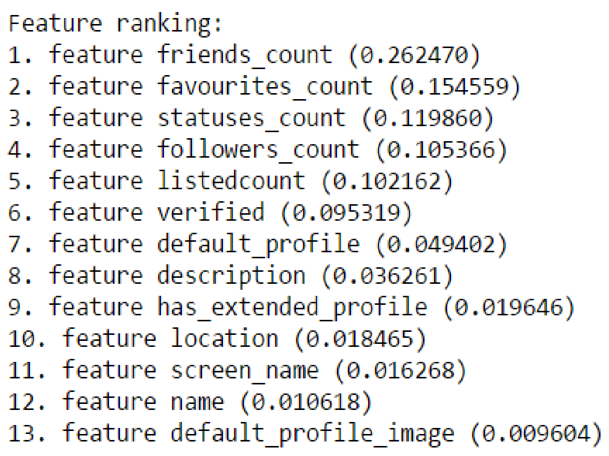


Figure 5 – Feature Importance


```
Feature ranking:
1. feature friends_count (0.262470)
2. feature favourites_count (0.154559)
3. feature statuses_count (0.119860)
4. feature followers_count (0.105366)
5. feature listedcount (0.102162)
6. feature verified (0.095319)
7. feature default_profile (0.049402)
8. feature description (0.036261)
9. feature has_extended_profile (0.019646)
10. feature location (0.018465)
11. feature screen_name (0.016268)
12. feature name (0.010618)
13. feature default_profile_image (0.009604)
```
Figure 6 – Feature Importance Values

Also, previously we used the different method for cleaning the data which restricted in gaining high accuracy. Hence, we used our own function for cleaning the data specifically for this dataset. This resulted in significantly high and stable accuracy while both training and testing.

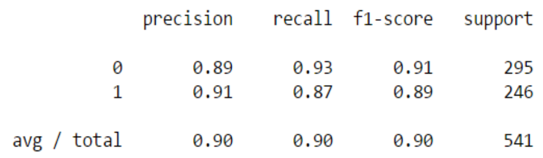
Here, the specific Testing data is available but it is on the kaggle and hence we can only get the accuracy of my model which is mentioned in statistics below. For the other measures of accuracy i.e. True Positive Rate, False Positive Rate, Receiver Operating Characteristics etc., we must rely on the available training data only. Hence the rest of the statistics are based on the total training data of 2797 accounts.

True Positive (TP) : 213
True Negative (TN) : 273
False Positive (FP) : 22
False Negative (FN) : 33

True Positive Rate : 0.8658
False Positive Rate : 0.0745

When tested on the kaggle competition, this model gives the accuracy of 93.5%

Classification Report for the same it given below:


```
             precision    recall  f1-score   support

          0       0.89      0.93      0.91       295
          1       0.91      0.87      0.89       246

avg / total       0.90      0.90      0.90       541
```
Figure 7 – Classification Report

Based on the information provided above, if we generate ROC curve, it looks like this.

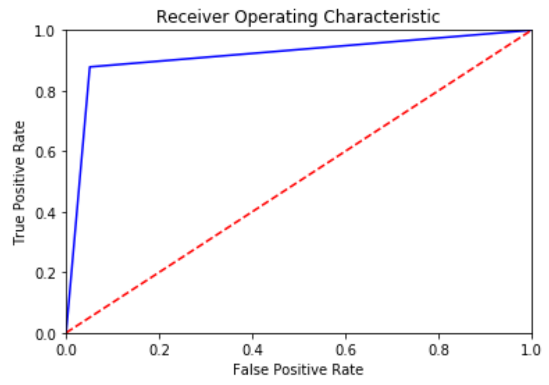


Figure 8 – ROC for Random Forest (Training)

VII. CODE

Please find below links to our GitHub accounts, which contain codes used to implement the algorithms mentioned.

- K- Nearest Neighbor Algorithm: https://github.com/mm-shah/Twitter-Bot-OR-Not
- Random Forest Algorithm: https://github.com/vbg221/Tweeter-Bot-Detection

VIII. VIDEO LINK

Here is the video link of the presentation that we uploaded to Youtube.

https://youtu.be/WYCZ6ZjfAJ0

IX. EVALUATION

For the future work, we can remove the garbage data from the status field and perform sentiment analysis over the actual meaningful statuses. its outcome can be used as a feature to improve the accuracy by observing the patterns of tweets by users. For example, we can determine if user is randomly generating tweets or not.

There are also some more possible feature transformations that we would like to test for e.g. friend vs followers' ratio, age of account etc. These features may also help in determining the Bot accounts efficiently.

Also, we would like to work on the raw data that we used for training because there was consequently large amount of garbage data in the status field of the training data throughout the whole dataset. There were also doubts about authenticity of the labels as many of the publicly available resources fail to identify a genuine account and mark it as a Bot and vice versa.

X. CONCUSION

Using machine learning methods KNN and Random Forest Trees, we were able to successfully identify a twitter account as a Bot or not a Bot with 95% and 93.5% accuracy respectively.

REFERENCES

- http://nyag.com/selectall/2015/11/12-weirdest-funiest-smartest-twitter-bots.html
- https://en.wikipedia.org/wiki/Twitterbot
- Scikit Documentation
- Project Truthy
- The rise of the social bots - by Ferrara, Varol, Davis, Menczer, Flammini link
- Who Will Retweet This? Automatically Identifying and Engaging Strangers on Twitter to Spread Information - by Kyumin Lee link
- Bot or not - by Shellman, Erin link
- Bot collection - link