Twitter Bot Or Not

Vamsi Mohan Ramineedi
CSE, New York University
vmr286@nyu.edu

Parinita Bhandary
CSE, New York University
pvb232@nyu.edu

***Abstract*—**This project aims to analyze existing twitter accounts using Machine Learning Algorithms and predict a new account as a bot or human account.

I. Introduction

In this modern day era, Twitter is one of the top social media application. It is widely used by millions of people to express their day to day feelings. All the companies maintain a twitter account to announce their latest releases and consider the retweets as a review of the product. We find a lot of people posting both positive and negative tweets. Out of these, few are being posted automatically by bots.

In this project, our aim is to find out if the twitter account is being controlled by a human or bot, using the popular Machine Learning algorithms.

II. Motivation

With the ever increasing use of Social Media and sudden decrease in privacy, it is important to identify the twitter bots that are increasing in number every day. While many bots can be harmless, most of the bots are created with the intention of creating a nuisance. From providing misleading or unverified information to the users to creating a negative, chaotic environment, most bots can have a negative impact on the people instead of providing a healthy platform for discussion.

With the intention of differentiating a genuine human twitter account from a bot, our project aims at developing an algorithm that can achieve this by classifying an account as a bot or not. This project will enable the users to verify if their followers are bots and in turn rid them of such followers.

III. Related Work

1. With the increase in the number of bots, there has been increased interest in identifying twitter accounts that are bots. For example, in the article 'The Rise of Social Bots' by Emilio Ferrara, Onur Varol, Clayton Davis, Filippo Menczer, Alessandro Flammini, the bot detection system was based on three strategies: social network information using graphs, crowdsourcing bot detection i.e. asking users to detect bots based on observed features and feature based bot detection based on behavioral patterns adopted with machine learning techniques. The method adopted in 'Shellman, Erin. Bot or not' involved using the features present on a user's profile page eg. follower/following counts, tweet etc. and also the last 200 tweets of the user. The data was fit using the three classifiers. Naive Bayes, logistic regression and random forest classifier. In the paper 'Online Human-Bot Interactions: Detection, Estimation, and Characterization' by Onur Varol,Emilio Ferrara, Clayton A. Davis, Filippo Menczer, Alessandro Flammini, the machine learning system was trained on more than a thousand features in different classes eg. users, tweet content and sentiment, activity time series etc. Other related work includes Lee, et al. Who Will Retweet This? Automatically Identifying and Engaging Strangers on Twitter to Spread Information.

IV. Data

The dataset for this project was obtained using the Twitter API. The dataset contains the details of approximately 3500 twitter accounts - half the accounts are bots while the other half are not (i.e operated by humans). Approximately 2800 twitter accounts from this dataset (containing both bot and non-bot accounts) form the training dataset while the other 575 accounts form the test set. Among other fields, it contains details such as the account ID, screen name, account name, description, status count, number of followers and friends and a bot field specifying if the account is a bot or not. The human accounts' data was collected from the few of the known twitter accounts. The bots data was collected analyzing the words like bot in screen_name; similar retweets and count of tweets posted.

Packages required: We have used the following packages in designing our algorithm.

1. Pandas
2. Numpy
3. Regular Expressions(re)
4. Datetime
5. sklearn

Loading the Data: We have used "Pandas" to load the dataset into a dataframe. Pandas provides the easiest way to play with data. The read_csv method allows us to load the dataset.

Preprocessing: The preprocessing of the dataset consists of the following steps:

Cleaning the text data: Usually data obtained through Twitter API is not clean. It will contain quotations, html type words, exclamation marks, numbers, strings, symbols etc. We need to clean the data to transform it to a format that can be used in training the algorithm. The data can also contain missing values which needs to be dealt with before it can be used in modeling. For the numeric fields, we have replaced the missing values ('None' values) with 0 while for the boolean or categorical fields, we have replaced the missing values with the mode value of that feature. Also for simplicity, we have extracted the required features that we plan to use for training the algorithm.

Preprocessing the required features: Since most of the bot accounts have the word "bot" in their screen name and/or description and/or account name, we can use it as a feature. The feature screen_name, name and description being a string feature was converted into an integer feature since machine learning models cannot process strings. To check for the word bot, we need the data to be in string format. For this, we used Regular Expressions package to delete the unwanted text from the description, name and screen name fields. Then, we changed the words to lowercase so that it becomes easier to compare with the word "bot". We then assigned the value 1 to the rows of screen name, name and description fields, which have the word "bot" in them and 0 if they don't.

The data was preprocessed before using it to train any of the models, since the data was unclean and consisted of features not required to train our machine learning models. First we extracted the required features of the dataset into a new data frame which consisted only the following features: screen_name, description, name, followers_count, friends_count, listed_count, favourites_count, verified, statuses_count, created_at and lang. Different features were used at different points of training the algorithm to check and improve the prediction accuracy of our training algorithm.

According to an article on MIT Technology Review: How To Spot a Social Bot on Twitter (https://www.technologyreview.com/s/529461/how-to-spot-a-social-bot-on-twitter/), bots tend to have younger accounts as compared to human accounts. Also from the dataset it can be seen that even newer bot accounts have a significant number of status counts, almost comparable to older human accounts. To take advantage of these facts, we found the average number of tweets an account has made since it's creation. For this, we found the number of days since the account was created and the current date using the datetime package offered by Python. The status count was then divided by the number of days to get the ratio of average number of tweets for an account per day. This was one of the features used to train our model (stored in the created_at field).

The features verified, default_profile, default_profile_image and has_extended_image were boolean. Also the lang field was categorical, which can't be used directly in training the algorithm. Using the LabelEncoder module of the Scikit-Learn library of python, the categorical feature labels were transformed in order to be assigned numeric values. Once the dataset was ready to be trained, the dataset was split into the Training set and the Test set. The training set is used to train our machine learning models and the test set is used as the new set i.e our trained model is applied to the test set which then predicts the label for these new accounts. The split of the dataset was obtained using the train_test_split module of the Scikit learn cross-validation library. We assigned 80% of the original dataset to be used as the training set while the rest 20% will be used as the test set. The 'X_train' is the training data consisting of the features that will be used to train the model while 'y' has the corresponding labels (if it is a bit or not). Similarly, the 'X_test' has the test features while the 'y_test' has the corresponding labels.

V. ALGORITHM(S) USED

Since the project involves classification, we chose few of the classification Machine Learning algorithms, namely, **Naive Bayes, Decision Trees, Random Forests.**

Naive Bayes: Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set. It is not a single algorithm for training such classifiers, but a family of algorithms based on a common principle: all naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable.

Firstly, we trained our classifier with Naive Bayes algorithm using the training data. The results have been on the lower side with an accuracy of 68%. Hence we decided on implementing the classifier with Decision Trees and Random Forests and not proceed with Naive Bayes any further.

Decision Trees: 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.

In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. In data mining, a decision tree describes data (but the resulting classification tree can be an input for decision making). This page deals with decision trees in data mining.

The goal is to create a model that predicts the value of a target variable based on several input variables. Each interior node corresponds to one of the input variables; there are edges to children for each of the possible values of that input variable. Each leaf represents a value of the target variable given the values of the input variables represented by the path from the root to the leaf.

Some advantages of decision trees are:

- Simple to understand and to interpret. Trees can be visualised.
- Requires little data preparation. Other techniques often require data normalisation, dummy variables need to be created and blank values to be removed. Note however that this module does not support missing values.

The disadvantages of decision trees include:

- Decision-tree learners can create over-complex trees that do not generalise the data well. This is called overfitting. Mechanisms such as pruning (not currently supported), setting the minimum number of samples required at a leaf node or setting the maximum depth of the tree are necessary to avoid this problem.
- Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This problem is mitigated by using decision trees within an ensemble.

The Information Gain feature of attributes is used to determine the attribute to be selected at each level. The attribute with highest information gain is selected.

Scikit-learn (formerly scikits.learn) is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy. We made use of Scikit-learn to implement Decision Trees algorithm in our program.

We used the tree module from Scikit-learn library and imported the DecisionTreeClassifier class into our python program. Supported criteria are “gini” for the Gini impurity and “entropy” for the information gain. We chose Entropy as the criterion to model our classifier.

The classifier has been fitted with the X_train and y_train values obtained during the train_test_split implementation. The classifier predicts using the predict function with the Z value as an argument.

Random Forest Classifier:

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting.

In order to use Random Forest Classifier in our python program, we again need to use Scikit Learn library. We have used the ensemble module from Scikit Learn and imported the class RandomForestClassifier. The object of the class was created using the parameters “n_estimators”, “criterion” and “random_state”.

VI. RESULT

Naive Bayes: The initial model trained using the Naive Bayes Classifier has an accuracy of 68%. Since the accuracy was low as compared to Decision Tree and Random Forest algorithms, we decided not to proceed with Naive Bayes algorithm to train our model.

Decision Trees: The initial prediction results from Decision Tree Classifier were largely better than that of Naive Bayes classifier. In order to check and improve the accuracy of the Decision Tree model, we trained the model selecting different features at different times and calculating its accuracy. In our initial attempt we selected 13 features to train our model consisting of the following features:

Screen name	Status count
Description	Ratio of status count to number of days since account was created
Followers count	Lang
Friends count	Default Profile
Listed count	Default Profile Image
Favourites count	Has Extended Profile
Verified	

It yielded the best accuracy score on the training set (~96%) but the actual accuracy score was really low (for the test_data_4_students.csv file). This was largely due to overfitting as selecting too many features for training the model can lead to it.

Similarly, some of our other attempts included the following features

Description	Screen name
Followers count	Ratio of status count to number of days since account was created
Friends count	Lang
Listed count	Verified
Favourites count	Status count

It yielded good accuracy score on the training set (~89%) and the actual accuracy score improved too (for the test_data_4_students.csv file).

Another feature set

Description	Screen name
Followers count	Ratio of status count to number of days since account was created
Friends count	Lang
Listed count	Verified
Favourites count	Status count

It yielded similar accuracy score on the training set (~88%) as the previous one and the actual accuracy score improved too (for the test_data_4_students.csv file).

Random Forests: Comparatively, this is the best classifier that we have built until now. It's accuracy is better than Decision Tree Algorithm too as it is better at controlling overfitting. Our initial attempt involving 13 features to train the model yielded an accuracy score of 98% on the training set but the actual accuracy score was really low (~68%). This was largely due to overfitting.

For the second feature set, our model yielded good accuracy score on the training set (~91.5%) and the actual accuracy score improved(~96). Similarly for the third feature set our accuracy on the training set was 90% and the actual accuracy score was similar to the previous one(~96%).

VII. CODE

The python program for our project can be found on the below mentioned GitHub site.

https://github.com/VamsiMohanRamineedi/Twitter_Bot_Or_Not

VIII. VIDEO LINK

The project presentation can be found here *https://youtu.be/aK4DKJyitcA* .

IX. EVALUATION

What went well: Our classifier worked well both with Decision Trees and Random Forests. One point to be mentioned is, the accuracy has increased nearly 3% when we used statuses_count and created_at fields together.

What didn't go well: Our classifier didn't work well when trained using Naive Bayes algorithm. This might be because of using too many features.

Future improvements: If we had preprocessed the tweets from status field of each account, it would be easier to predict with the use of nltk and countvectorizer packages. Also, we could have handled the description field in an efficient way.

X. CONCLUSION

1. Selecting too many features led to overfitting and resulted in less accuracy.
2. Random Forest algorithm gave better predicted results than Decision Tree algorithm as it is better at controlling overfitting than Decision Tree.