Allison Tumas
IMSE 586 – Final Project
Fall 2020

Project Title: Detecting Predictors of Social Media Bots

1. Abstract:

This project focuses on supervised machine-learning and aims to determine which Twitter account features might aid in differentiating between authentic human accounts and bot accounts. The project starts with cleaning and normalizing the data and applying multiple linear regression to determine feature selection. Then, supervised learning models are applied to a training set to determine the accuracy of predicting bot accounts from these features.

2. Introduction

Social media platforms have recently been scrutinized by both the public and Congress for their inability to control "bot behavior", which is the automation of social media activity. While not all automated social media activity is bad i.e. customer service bots, automation was also used to manipulate data and news to mislead the public on important matter such as elections and the stock market. Detecting bot behavior has been a challenge for many years and many techniques have already been developed; However, both bot behavior and social media platforms are increasingly dynamic and new practices need to continually be developed to keep up with automated online behavior.

This project serves as a starting point to apply data wrangling, analysis, visualization and modeling techniques learned over the course of the semester. This project is hopefully only part one in a series of projects that focus on more advanced methods of machine learning and classification, application of models to applied natural language processing and social network analysis.

3. Research Questions

The project will focus on the following research questions:

- Which social media features indicate bot behavior?
- Can a social media bot be identified from only profile information?
- Can ML be applied to bot-identified actions on Twitter in order to detect new bot accounts or changing patterns of bot-behavior?

4. Methods:

The objective of the methods used was to build a classifier using a supervised machine learning approach that could be applied to a set of Twitter account features to determine whether that account is a bot. Supervised learning means training a model to determine based on labeled data.

4.1 Interactive Development Environment and Data Import

The project was developed in a Jupyter Notebook and programmed in Python. The main python library used for data analysis was pandas. Other important libraries in this project include: numpy, sklearn, matplotlib.pyplot and seaborn

4.2 Pre-Processing Data

To pre-process the data, we remove the ID, and standardize the date/time columns. We then convert the columns that have too many features for categorical encoding into Boolean operators (True/False). At this stage we can apply categorical encoding in which we multiply those columns by 1 in order to convert values to numerical data. We then One Hot Encoding to any categorical columns that are left, in this case it was only the target variable: account type.

One Hot encoding is a simple way to convert categorical data (objects) into numerical data as it creates a new column for each value in the feature. Label encoding can also be used in this instance but is more commonly used with features that have a higher number of set variables. After applying one hot encoding it is important to take away one feature, in this case we took away the column account_type_human because intuitively if the account is marked as a non-bot type (0) in the corresponding column, it is marked as human account type (1). This ensures there is no unnecessary duplication to skew the data.

While we could do some exploratory analysis before moving to feature selection, it seemed more worthwhile to explore the data that we already know is important from statistical feature selection methods. Still, in order to think through the data a bit more and come up with my own hypothesis, I grouped the data frame by account type and looked at the averages for each feature. Below are my observations at first-glance for bot accounts:

- Default profile/image is more prominent among bot accounts
- Bot accounts have a higher average tweet per day
- Favourites count is significantly lower among bot accounts
- Followers count is significantly lower among bot acounts
- Status count is lower among bot accounts

4.2 Feature Selection and Dimensionality Reduction

The next piece of this project focused on statistically based feature selection. Feature selection is an important process in machine learning because it helps to reduce overfitting and underfitting of the mode. It's primarily focused on removing non-informative or redundant predictors from the model. Since we are using a supervised learning technique, we use the target variable to help remove irrelevant variables. However, our dataset was a bit more complex because it contained both numerical and categorical data. To remedy this, we ran two different methods for feature selection.

<u>Pearson Correlation on Numerical Data</u>

The Pearson correlation is the standard correlation of coefficients and computes the pair-wise correlation of columns (excluding null values). In other terms, it divides covariance by the standard deviation of x and y. The resulting correlation can be anywhere between:

- 1 (total positive linear correlation)
- 0 (no correlation)
- -1 (total negative linear correlation

It may seem counter-intuitive, but in this step we filter out one of the columns for two features that have a high correlation (in this case an correlation of .9 or greater). This high correlation implies that one of the features is unnecessary because we can obtain the same information from the highly-correlated feature. This is one of the easiest and simplest feature selection models, but it only works with continuous variables.

Logit Model on Categorical Data

After running the Pearson correlation on the numerical categories, we can use the logit regression model on the target variable, which allows for multiple predictors. We want to run a statistical summary and determine irrelevant features based on the P-value. The p-value gives us some insight into the null hypothesis for each feature:

- **Null hypothesis** is a general statement that there is no relationship between two measured phenomena.
- **P-Value:** Probability value testing if the null hypothesis is true; the feature has no correlation with the dependent variable.

It's important to note that removing different features from the dataset will change the p-value, therefore after removing features from the dataset, the model should be run another time. Another important calculation we receive from the regression model is the coefficient, which portrays how the mean of the target changes for each one-unit shift in the feature. For categorical features, the coefficient tells us the odds that the target feature will shift.

4.3 Exploratory Data Analysis

After applying feature selection and dimensionality reduction, I visualized the relationships between the various features using scatterplots. Scatterplots tell us about covariance, which is the mean of the product of the differences between two features. Positive covariance signifies positive correlation.

4.4 Data Modeling

For data modeling, we can use scikit learn, a machine learning module in Python. This module has multiple classification and validation methods that all follow the same structure. Therefore, it is common practice to run multiple classification models and see which one performs the best. Before we can run the classification model we have to split the data, as models should never be tested on the same data they were trained. We also want to stratify the data when splitting, which ensures that the values for the target data in the training set is proportional i.e. the same number of bot to human accounts. Lastly we have to normalize the data by scaling it using either:

- Standardization = subtract the mean and divide by variance
- Min 0 and Max 1 = subtract minimum and divide by range

After we've taken these steps, we are ready to pass the split and scaled data into multiple classification models. I chose the models defined below:

- **Logistic Regression:** Ensures the probability of X is always bounded between 0 and 1, which works well for binary classification. However, proper selection of features is very important for this model.

- **K-Nearest Neighbors:** KNN works by classifying the data based on the n-nearest data points (using Euclidean and Manhattan distances). It is very important to scale your features in this model for fair value given among all of them. This supports a non-parametric model and non-linear solutions but can't derive the confidence level from its predictions.

- **Decision Tree Classifier:** Based on information gain and a greedy-based algorithm, Decision Trees are best for categorical values, but can't derive the significance of features as Logistic Regression does.

- **Random Forest Classifier:** Random forest utilizes the decision tree classifier because it is essentially a group of randomly classified decision trees and it combines the output of each individual tree as the total output. It's randomized feature selection often makes it more accurate than the decision tree classifier, but it is also more difficult to train.

After we've ran each classifier we can assess its predictions using a confusion matrix and a classification report. The confusion matrix give us the true positive, false positive, true negative and false negative prediction rates for the test set, based on our model. From this we can either manually calculate the classification report or run the scikit learn function. The classification report gives us the following metrics on our model:

Accuracy Score: The average amount of times the model predicted correctly
Misclassification Rate: The average amount of time the model predicted incorrectly
Recall: Sensitivity of the model; true positive rate
Precision: True positive rate over all predicted positives

We can also plot an ROC curve to show the false positive rate vs. the true positive rate and setting it to different thresholds. The closer the ROC curve is to the upper left corner, the better the classification model is.

4.5 Comparing Models

The classifier models for KNN, the Decision Tree Classifier and the Random Forest Classifier all have different parameters that can change the accuracy of the model. For K-Nearest Neighbors, you can determine the number of neighbors (close data points) that the model will calculate it's

distance from in order to classify the test set. For the Decision Tree and Random Forest Classifiers you can set the depth of the feature subsets that the model will calculate.

I tested created a function that would iterate through 20 different values and choose the best value, then I reset the value so that I knew the model was achieving its highest accuracy before comparing it with the other models.

I then reran the models with their best values and compared the accuracy between the models from their classification reports and using KFold cross validation. Cross-validation is simply resampling the data in order to evaluate the machine learning model based on the different random samples. KFold determine the number of groups that the data sample is split into or in other words, how many times the resampling will take place with unique data points. The output of these comparisons will be shown and discussed in the results section.

5. Description of the Data Set:
The data used in this project was downloaded from a Kaggle dataset (https://www.kaggle.com/davidmartngutirrez/twitter-bots-accounts) that provides key features of Twitter profiles for both human-identified accounts (25,013) and bot-identified accounts (12,425). The features used in this dataset include account profile information and are defined below:

- **created_at:** The UTC datetime that the user account was created on Twitter.
- **default_profile:** When true, indicates that the user has not altered the theme or background of their user profile.
- **default_profile_image:** When true, indicates that the user has not uploaded their own profile image and a default image is used instead.
- **description:** When true, indicates that the user has not altered the theme or background of their user profile.
- **favourites_count:** The number of Tweets this user has liked in the account's lifetime. British spelling used in the field name for historical reasons.
- **followers_count:** The number of followers this account currently has.
- **friends_count:** The number of users this account is following (AKA their "followings").
- **geo_enabled:** User allows geo-tagging on posts
- id: The integer representation of the unique identifier for this User.
- **lang:** When true, indicates that the user has not altered the theme or background of their user profile.
- **location:** The user-defined location for this account's profile.
- **profile_background_image_url:** When true, indicates that the user has not altered the theme or background of their user profile.
- **profile_image_url:** When true, indicates that the user has not altered the theme or background of their user profile.
- **screen_name:** The screen name, handle, or alias that this user identifies themselves with.

- **statuses_count:** The number of Tweets (including retweets) issued by the user.
- **verified:** When true, indicates that the user has a verified account.
- **average_tweets_per_day:** Average number of tweets posted by an account per day
- **account_age_days:** Number of days the account has been active
- **account_type:** Identified as bot or human (not by Twitter)
  Source: https://developer.twitter.com/en/docs/twitter-api/v1/data-dictionary/overview/user-object

6. Detailed Technical Description:

6.1 Data Treatment

In order to become familiar with the dataset, it was read into Jupyter Notebook using the pandas function pd.read_csv. I then previewed the data and checked the data types by using df.info and visualized the count target variable using seaborn's sns.countplot function.

Some columns were declared as objects instead of float or integers, which can be difficult data types to work with when applying regression analysis or classification models. I chose the following methods to prepare each column:

1) Unnamed 0: I dropped this unneeded index column using pandas df.drop function
2) Created_at: converted the standard datetime and dropped the time stamp
3) Description: Changed to bool type (True: the account provided a description; False: NaN)

td['description'] = pd.notna(td['description'])

4) Language: Changed to bool type (True: Language was provided; False: NaN)

td['lang'] = pd.notna(td['lang'])

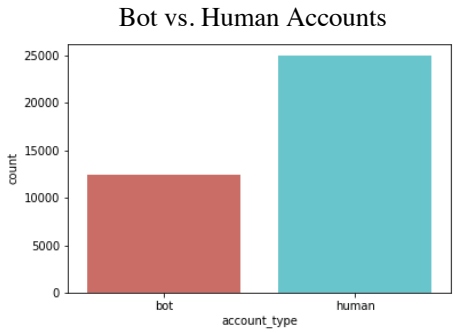
5) Profile_Backgound_image_url: dropped column; redundant due to correlation with default profile image 6oolean column
6) Profile_image_url: dropped column; redundant due to correlation; redundant due to
7) The following columns were dropped because they would be too time intensive to clean for the purpose of this project: id, screen_name, location
8) Missing values were also dropped using df.dropna()

After rechecking our datatypes we want to prep for categorical encoding, which means all variables that were changed to bool types need to be converted to int (numerical data). "True" is marked as 1 and "False" is marked as 0, which we accomplish by multiplying the dataframe by 1.

*Df [['default_profile', 'default_profile_image', 'description', 'geo_enabled', 'lang', 'verified']] *=1*

Due to the data preparation we will only have to categorically encode the "account_type" column which will effectively create two new columns (**account_type_bot, account_type_human**) in which is

the value is marked by 1 if the column variable is true. We then drop “obvious” column so data isn’t duplicated i.e. if we have a null value of 0 for account_type_bot, we know it is a human so we don’t need the account_type_human column. We then visualized the counts for each:



33% of accounts were bot accounts

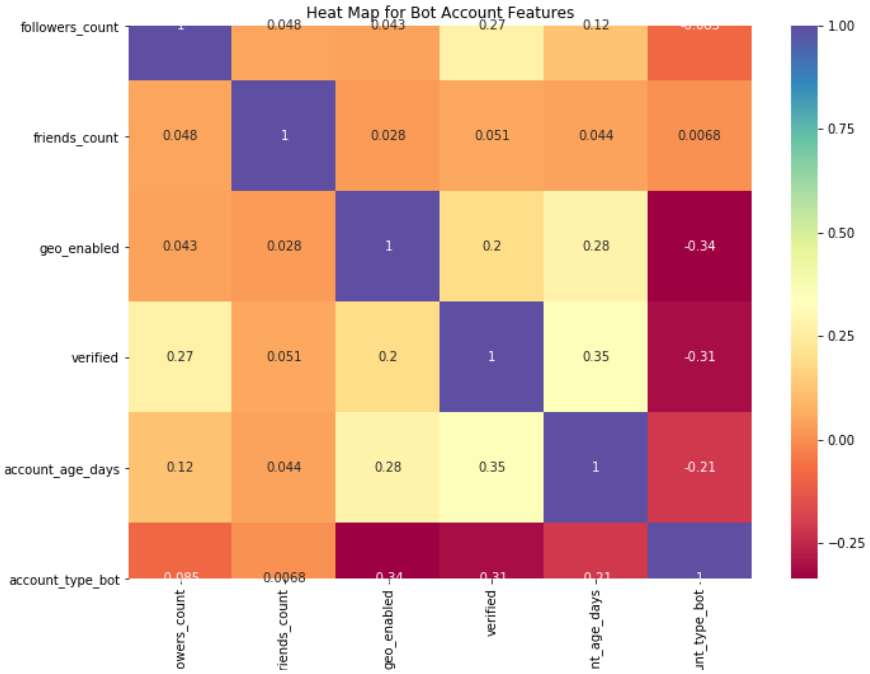
6.2 Feature-Selection

As described in the methods, Pearson Correlation was first applied to the dataset. First, we had to determine what was numerical data. Even though we had provided encoding to some of the features and their categories were represented with 1s and 0s, they were still considered categorical (or nominal inputs), so we instead focused on the numerical continuous features. In order to accomplish this we split the data and created new variables to store it:

```
#fs_cat = td_dum.iloc[:, [1,2,3,7,8,10,13]]
fs_num = td_dum.iloc[:, [4,5,6,9,11,12]]
```

Next, we visualized the correlation between the different features using seaborn’s heatmap and annotated the data with its correlation metric:

```
plt.figure(figsize=(11,8))
plt.title('Heat Map for Bot Account Features')
sns.heatmap(fs_num.corr(), cmap='Spectral', annot=True)
plt.savefig('correlation_heatmap.png')
```



We can drop either "average_tweets_per_day" or "statuses_count" because the correlation is higher than .9:

```
td_dum.drop(columns=['average_tweets_per_day'], inplace=True)
```

Next, we applied the multiple linear regression model in statsmodel. Given the categorical data in our dataset, we wrapped categorical predictors in C() to let the formula know the numbers represent categories and are not numerical.

```
import statsmodels.formula.api as smf

result = smf.logit(formula = 'account_type_bot ~ C(default_profile) + C(default_profile_image) +
C(description) + favourites_count + followers_count + friends_count + C(geo_enabled) + C(lang) +
statuses_count + C(verified) + account_age_days', data=td_dum).fit()

print(result.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.482483
         Iterations 7
                           Logit Regression Results
==============================================================================
Dep. Variable:       account_type_bot   No. Observations:                37438
Model:                         Logit   Df Residuals:                    37426
Method:                          MLE   Df Model:                           11
Date:                Fri, 11 Dec 2020   Pseudo R-squ.:                 0.2408
Time:                        00:01:51   Log-Likelihood:                -18063.
converged:                       True   LL-Null:                       -23792.
Covariance Type:            nonrobust   LLR p-value:                     0.000
=================================================================================================
                                    coef    std err          z      P>|z|      [0.025      0.975]
-------------------------------------------------------------------------------------------------
Intercept                         0.6712      0.058     11.488      0.000       0.557       0.786
C(default_profile)[T.1]           0.6317      0.029     21.738      0.000       0.575       0.689
C(default_profile_image)[T.1]    -0.1031      0.099     -1.042      0.298      -0.297       0.091
C(description)[T.1]              -1.1285      0.105    -10.708      0.000      -1.335      -0.922
C(geo_enabled)[T.1]              -1.1655      0.029    -40.422      0.000      -1.222      -1.109
C(lang)[T.1]                      0.1837      0.103      1.788      0.074      -0.018       0.385
C(verified)[T.1]                 -2.4045      0.067    -35.667      0.000      -2.537      -2.272
favourites_count              -2.641e-05   9.93e-07    -26.607      0.000   -2.84e-05   -2.45e-05
followers_count               -1.544e-08   1.22e-08     -1.261      0.207   -3.94e-08    8.55e-09
friends_count                  2.564e-06   3.29e-07      7.785      0.000    1.92e-06    3.21e-06
statuses_count                 7.908e-06   3.48e-07     22.747      0.000    7.23e-06    8.59e-06
account_age_days              -4.921e-05   1.43e-05     -3.440      0.001   -7.72e-05   -2.12e-05
=================================================================================================
```

We can run one more time after dropping the insignificant features to see if the p-values change, in this case, they do not.

6.3 Dimensionality Reduction

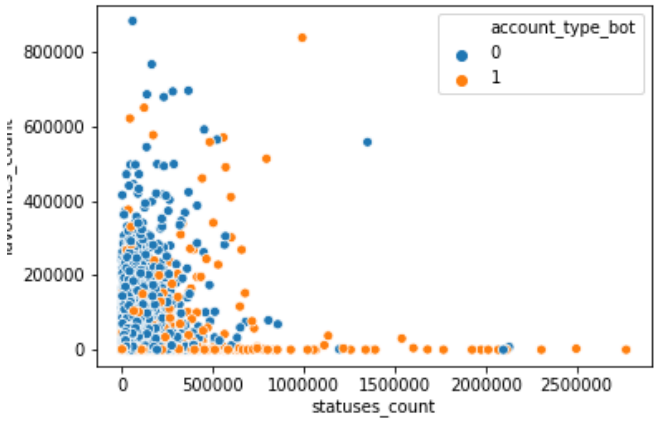
To reduce the dimensionality of our dataset and make it easier for modeling, we drop any irrelevant features with a high p-value. We then save this into a new data frame for modeling.

```
td_dum.drop(columns=['default_profile_image', 'lang', 'followers_count'], inplace=True)
df = pd.DataFrame(td_dum)
```

7. Data Analysis:

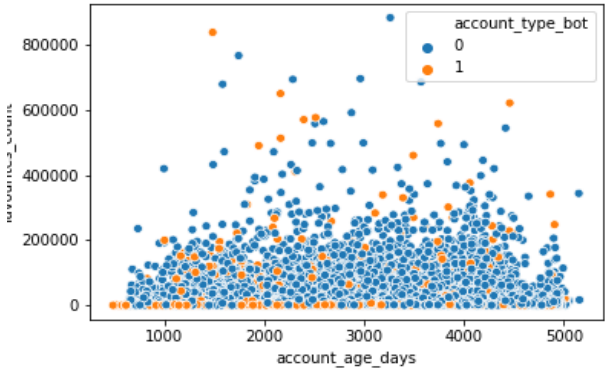
After applying feature selection and dimensionality reduction we can do some exploratory data analysis before modeling. I used seaborn's pairplot function to see the relationships between that features, then picked a few to visualize:

```
#plot_X = td_dum.iloc[:,[1,2,3,4,5,6,7,8]]
#sns.pairplot(plot_X)

sns.scatterplot(x='statuses_count', y='favourites_count', data=td_dum, hue='account_type_bot')
plt.savefig('statuses_favourites.png')
plt.show()
```



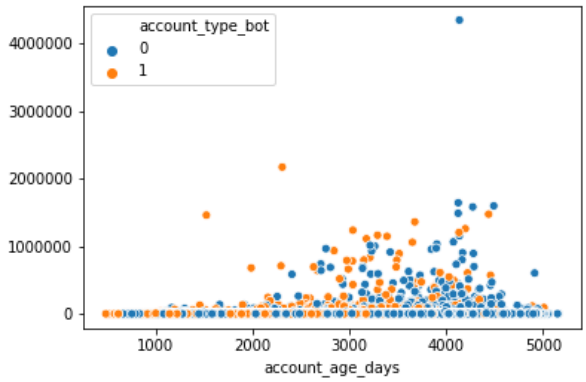
"Bot accounts have higher status counts with less favourites."

```
sns.scatterplot(x='account_age_days', y='favourites_count', data=td_dum, hue='account_type_bot')
plt.savefig('accountage_favourites.png')
plt.show()
```



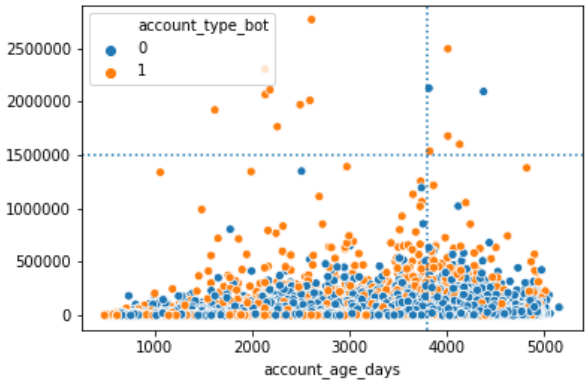
"Bot accounts receive much less favourites in general, regardless of account age"

```
sns.scatterplot(x='account_age_days', y='friends_count', data=td_dum, hue='account_type_bot')
plt.savefig('accountage_friends.png')
plt.show()
```



"Correlation between account age days and friends is similar bot and human accounts"

```
sns.scatterplot(x='account_age_days', y='statuses_count', data=td_dum, hue='account_type_bot')
plt.axvline(x=3800, linestyle='dotted')
plt.axhline(y=1500000, linestyle='dotted')
plt.savefig('accountage_statuses.png')
plt.show()
```



"Correlation is fairly similar with the exception that a small percentage of outlier bot account have over 1.5 million tweets with their account open for less than 3800 days"

7. Data Modeling:

We begin by creating a variable for the data and the target feature.

```
X = df.iloc[:, [1,2,3,4,5,6,7,8]]
y = df.iloc[:, 9]
```

We then split the data using the scikit learn function train_test_split. Here we determine the training size, which I have set to a common training size (.75). We can also determine the random state and we can stratify the data.

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=.75, random_state=11, stratify=y)
```

Next we scale the data using the StandardScaler function from scikit learn:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

We can now build our first model based on Logistic Regression. We do this with scikit learn's Logistic Regression library. We fit the model to our training sets for both X and y. Then we predict the probability that our test set is a bot account or not, and we also predict the probability for that prediction.

```
from sklearn.linear_model import LogisticRegression
log = LogisticRegression(solver='lbfgs')
logreg = log.fit(X_train, y_train)

y_proba_log = log.predict_proba(X_test)[:,1].round(3)
y_pred_log = log.predict(X_test)
```

We then want to use the confusion matrix and classification import from scikit learn to obtain additional metrics about our predictions. The confusion matrix reads in the test data and predicted data for y and output the true positive, true negative, false positive and false negative rate for the predictions. Scikit learn's metrics function will tell us the overall accuracy of the model and the classification report gives us additional metrics on precision, recall, f-1 score and support.

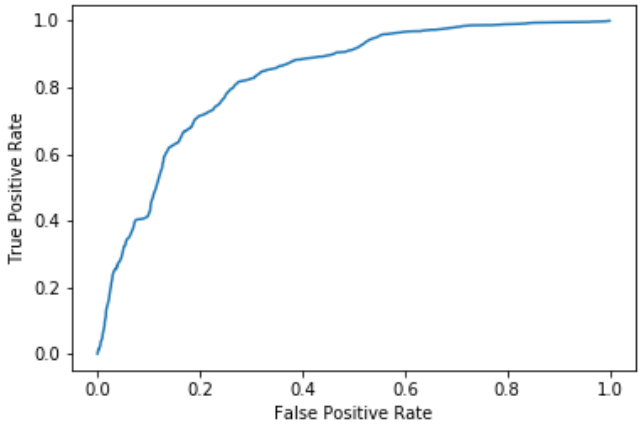
```
from sklearn.metrics import confusion_matrix, classification_report
from sklearn import metrics

cm_log = confusion_matrix(y_test, y_pred_log)
df_cm_log = pd.DataFrame(data=cm_log, columns=['predict: Respondent is human', 'predict: Respondent
is a bot'], index=['true: human', 'true: bot'])
print(df_cm_log)
#(7+18)/(7+9+18+1)
accuracy_log = metrics.accuracy_score(y_test, y_pred_log)
misclass_log = (1 - metrics.accuracy_score(y_test, y_pred_log))
print('accuracy: ', accuracy_log, "      ", 'misclassification: ', misclass_log)
```

```
#sklearn also provides a classification report
print('Classification Report: ')
print(classification_report(y_test, y_pred))
```

```
                predict: Respondent is human  predict: Respondent is a bot
true: human                             5289                           965
true: bot                               1143                          1963
accuracy:  0.7747863247863248       misclassification:  0.2252136752136752
Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.85      0.83      6254
           1       0.67      0.63      0.65      3106

    accuracy                           0.77      9360
   macro avg       0.75      0.74      0.74      9360
weighted avg       0.77      0.77      0.77      9360
```

In our case we find it doesn't often predict that the account is a bot leading to a lot of false negatives (account is human)

We then plot the ROC curve:



7.1 Data Modeling: Testing other Models

We then tested a few other models and ran the same analysis on each one, all using scikit learn.

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(11)

from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier(max_depth=8, criterion = 'entropy').fit(X_train, y_train)

from sklearn.ensemble import RandomForestClassifier
```

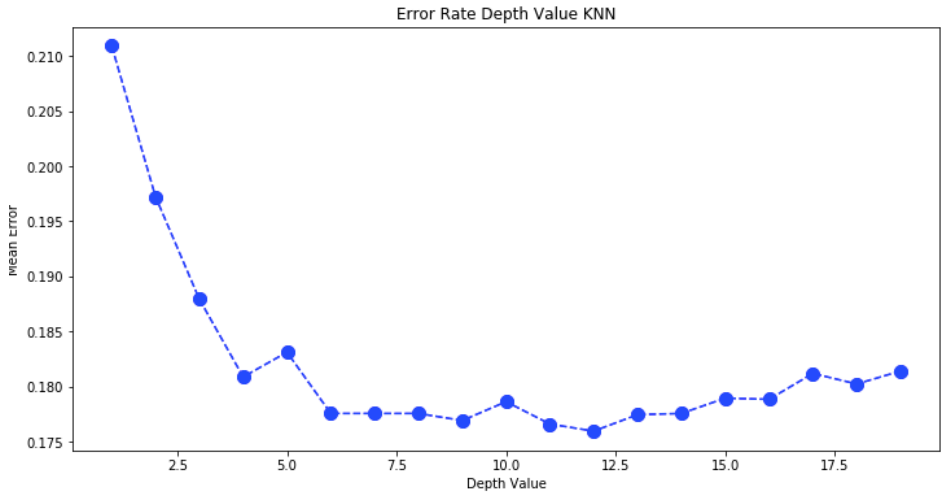
```
rfc = RandomForestClassifier(max_depth=15, random_state=0,n_estimators=10).fit(X_train, y_train)
```

8. Compare Models

The final step was to compare the models. I first chose the best value for the number of neighbors (KNN) and the max depth (DTC and RFC) within the models by logging the errors for a different set of values in a list and plotting it on a graph to visually see the lowest error.

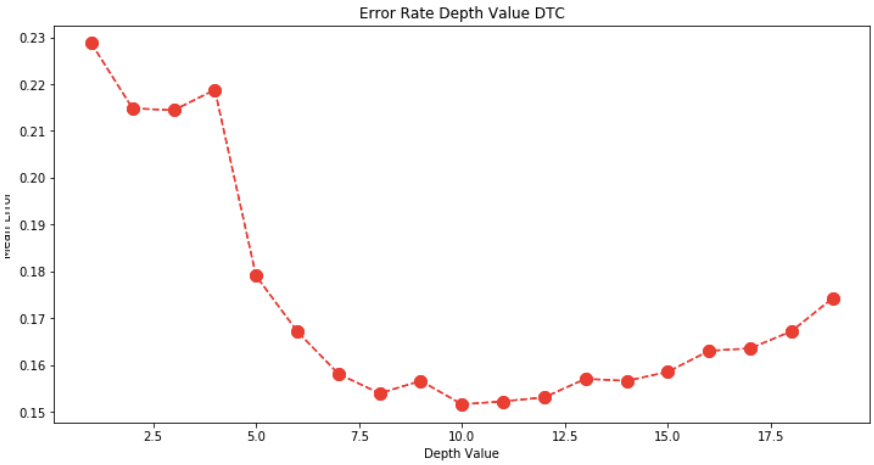
```
#K-Nearest Neighbors
error_knn = []
for i in range(1, 20):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    pred_i = knn.predict(X_test)
    error_knn.append(np.mean(pred_i != y_test))

#KNN
plt.figure(figsize=(12, 6))
plt.plot(range(1, 20), error_knn, color='blue', linestyle='dashed', marker='o', markerfacecolor='blue',
markersize=10)
plt.title('Error Rate Depth Value KNN')
plt.xlabel('Depth Value')
plt.ylabel('Mean Error')
```

K-Nearest Neighbor



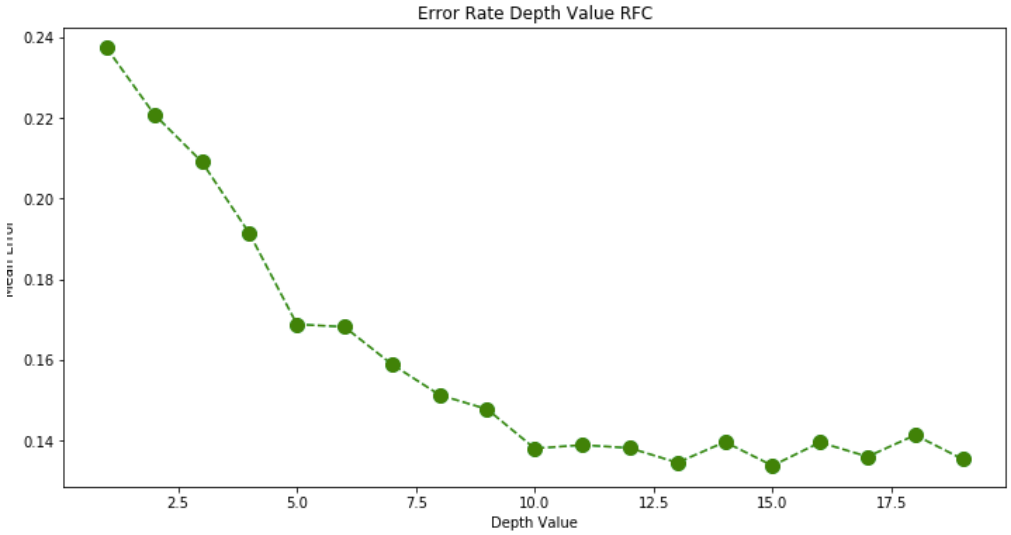
KNN k-value with the lowest error: 12

Decision Tree Classifier



DTC max depth value with the lowest error: 10

Random Forest Classifier



RFC Max depth value with the lowest error: 11

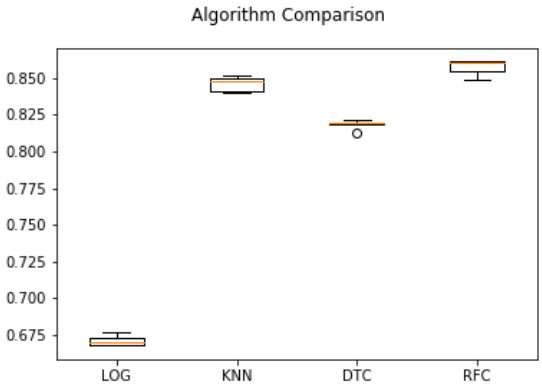
I then reran the models with their best values and compared the accuracy between the models from their classification reports and using KFold and cross validation. The output of these comparisons will be shown and discussed in the results section.

9. Results

I. Comparing Classification Reports

```
Logistic Regression:  accuracy:  0.7747863247863248      misclassification:  0.2252136752136752
K-Nearest Neighbors:  accuracy:  0.8240384615384615      misclassification:  0.1759615384615385
Decision Tree Classifer:  accuracy:  0.8483974358974359      misclassification:  0.15160256410256412
Random Forest Classifer:  accuracy:  0.8618589743589744      misclassification:  0.1381410256410256
```

II. Comparing Accuracy with KFold and Cross-Validation



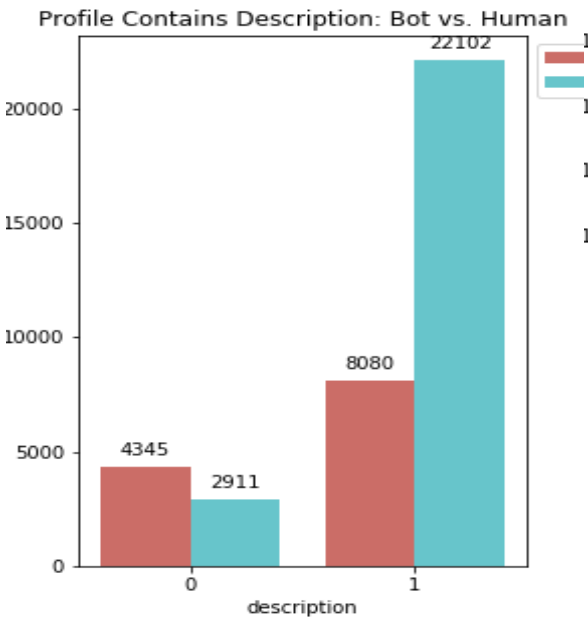
LOG: 0.671243 (0.003554)
KNN: 0.846039 (0.004858)
DTC: 0.818607 (0.003098)
RFC: 0.857418 (0.004847)

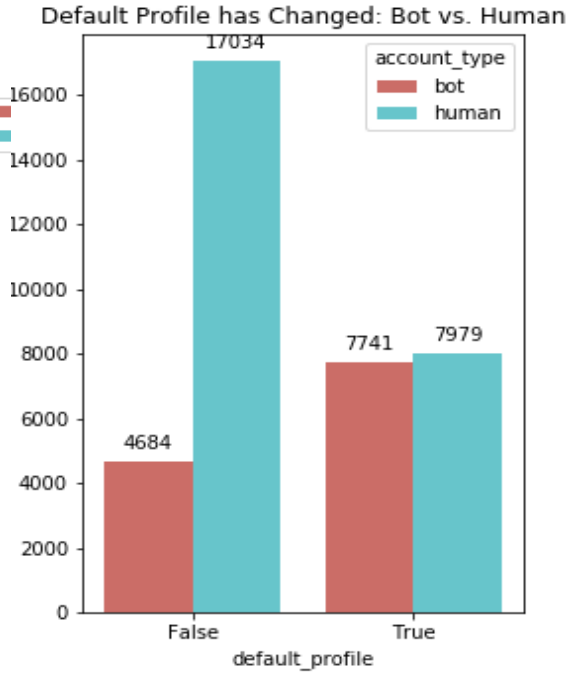
The result of the project was that we achieved an 86% accuracy score in predicting bot accounts based on only profile information when using the Random Forest Classifier on a single test. The decision tree classifier and K-Nearest Neighbors also achieved an accuracy score of greater than 80% on a single test. When we applied the KFold methodology and cross-validation, the Random Forest classifier remained at just around an 86% accuracy score. Meanwhile, the K-Nearest Neighbors accuracy score improved by .2, giving it a higher accuracy rating than the Decision Tree classifier; which dropped to 81%.

Summary Predictors for Detecting Bot Accounts

While these models can be applied to new datasets and accounts in the future, we were also able to use exploratory data analysis to answer the simpler question of "exactly which features could we use to qualitatively identify bot accounts?" The conclusions were as follows:

- Less than .03 bot accounts are verified compared to nearly 3% of human accounts.
- 78% of human accounts have changed their default profile while only 38% of bot accounts changed their default profiles.
- Only 2% bot accounts are geo enabled comapred to 58% of human accounts
- 88% of human account have a description in their profile vs. only 65% of bot accounts.





10. Discussion:

At the close of this project, I was able to answer nearly every research question I proposed. Social media features did, in-fact, indicate bot behavior. Moreover, profile information alone can be an indicator of bot behavior and we can detect it with the models used nearly 86% of the time.

This is an average accuracy score and while I wouldn't recommend shutting down accounts based on the model, it could certainly aide in flagging accounts for review by the Twitter Integrity team, or by one's self if they are curious about whether their followers are bots or not. Lastly, I think this confirms that machine learning can be applied to new bot accounts and can potentially keep up with changing patterns in bot behavior, so long as it is being fed new data and the errors and features are remaining logged. It did not surprise me that bot account receive less interactions that human accounts, i.e. less favourites albeit more status counts. Nor was I surprised to see that there were a few bot accounts that stood out as outliers for the high number of posts in a short period of account duration. However, I was surprised to see that bot accounts and human accounts we're fairly similar when comparing the friends that they'd acquired vs. the time the account had been opened for. It shows that bots can interact through likes and follows and comments and possibly create relationships online or mimic human behavior better than I had anticipated going into this project.

These results and the project in general confirmed my assumption that there is still a lot of work that can be done in this space and it is worth it to expand on this project. Some of the additions I have already started or would like to make to this project are as follows:

- Apply natural language processing to the description and tokenize words in order to give a value count for each type of account and see if there is a patter in the types of words bot accounts use.
- Apply natural language processing (NLP) to the lang field in order to parse out emojis and other outlier text in an attempt to standardize this field enough to encode and add to the original model.
- Use feature engineering to parse out date values that might give additional meaning to the data i.e. number of accounts during typical work hours for the account location.
- Connect to Twitter API and pull content for more in depth Time Series Analysis and NLP feature engineering and compare how the addition of this feature could improve the model.
- Test new accounts pulled from the Twitter API.
- Apply unsupervised learning methods to new Twitter accounts pulled from the API and see how these models compare.