Mary Washington

CSCI 390

Dr. Ghosh

09 December 2018

Report

1. Introduction

The goal is to determine an algorithm that can be used to try to classify an account as either real or fake, using statistical analysis. Social networks have a rampant number of fake accounts, which lead to the degradation of user experience. I am working with Twitter specifically, and I am interested in finding spam accounts. There exists, several different types of fake accounts, but I am focusing on spam accounts. They ruin the user experience and have been known to tarnish the reputation of Twitter. Twitter is trying to rebrand itself as a legitimate news source and they need to ensure that they are providing credible information. Spam can also compromise the user's security and privacy [3]. I have found several works that have had similar ideas. One focuses on understanding the different types of Twitter users[3]. One focuses on detecting spambots[4]. Another focuses on the spam itself[5]. One looks at which features are useful in detecting who is human, bot, or cyborg[1]. The last article looks at using sentiment analysis to determine bots on twitter[2]. The idea of spambots is troubling because it poses the question of "who to trust". In any network trust is a core feature, and once trust is lost it is hard to regain it. Here are the summaries of each article:

Detecting Automation of Twitter Accounts

This article was trying to determine the features use for detecting humans, bots, and cyborgs (human-aided bots/bots-aided humans) on twitter, using Twitter API. While some bots simply post things like status updates and news articles, a lot are used for spamming and creating an unpleasant experience for the community at large. The data was more than 500,000 twitter users generating more than 40 million tweets. I liked that they broke up the classification into determining: entropy, spam detection, account properties, and decision making using random forest. They noticed that a classic spamming behavior was that they had a high follower-to-following ratio. Depth First Search and Public Timeline API was used to collect the data. They used a ground-truth set, which they developed by taking 6,000 samples and manually labeled them as bot, human, or cyborg (2,000 per class). They use the cumulative distribution function (cdf) to determine bots from humans as well, with [#followers/(#followers+#friends)]. Humans have the highest value, cyborgs less than that, and bots less than 0.5. It is also interesting to me that they ranked the top 10 ways that the classes tweet, with humans (web), bots (API), and cyborgs (Twitterfeed), being number 1. They used Weka (I need to investigate this) to implement Random Forest and used 10-fold cross validation to test accuracy. They ended up with a 96% accurate model.

Using Sentiment to detect bots on Twitter

SentiBot is a sentiment-aware architecture for identifying bots on Twitter. It uses tweet sentiment to draw features for its analysis. These include: sophisticated sentiment analysis techniques, neighborhood-aware semantic metrics, syntactic tweet metrics, linguistic models, and graph-theory. The dataset used was the "India Election Dataset" (IEDS). The training set was selected randomly from the 897-user sample set (population was a lot higher). I thought it was interesting that out of the 145 variables used the study, 135 were sentiment-related, and 128 were topic of interest related. The use of sentiment variables improved the accuracy of their classifier. They learned that bots flip-flop less frequently than humans, humans express stronger positive and negative sentiments than bots, and humans disagree more with the general sentiment of the application of Twitter population than bots.

Understanding Types of Users on Twitter

The researchers used supervised learning and performed 10-fold cross validation to determine the different types of users. They performed a classification between 3 types of real accounts (personal, professional, and business) and fake accounts (spam users, feed/news, and virtual/marketing services) [1]. This is a good idea because if one can classify these different types of users, especially as a business model, one can determine the best approach to interacting with these users. The real accounts didn't have many features in common but the common features amongst the fake accounts were highly frequent tweeting, no or less interactivity, and followers either increased over time (feed/news users) or decreased (spam users) [1]. These different cases lead to a class imbalance which can lead to overfitting, and an unbiased classification system was used. The authors chose to use Random Forest with bagging approach, using 10-fold cross validation with 90/10 training and testing set. Overall the approach worked well except for the feed/news users.

Detecting Spam Bots in Online Social Networking Sites: A Machine Learning Approach

The researchers attempted to use 3 graph-based features: number of friends, number of followers, and ratio and follower ratio (the number of followers divided by the number of followers + number of friends) and 3 content-based features: number of duplicate tweets, number of HTTP lines, and number of replies/mentions, to detect spam bots [2]. The best approach was deemed to be the Bayes Classifier, which is noise robust. In order to detect a duplicate tweet, from the same account the authors used the Levenshtein Distance or edit distance between two tweets. If the distance was zero then the two tweets were considered identical, hence a duplicate, which has a higher probability of being spam.

Detecting Spam in a Twitter Network

The authors focused on the spam surrounding the #robotpickuplines hashtag. Spammers may retweet and change legitimate links to illegitimate ones or are simply obfuscated by URL shorteners (i.e. bit.ly) [3]. It is well-known that in-degrees represent the followers and out-degree represents friends. The authors found that spam lifecycle during the meme mirrored the lifecycle of it, though it lagged behind. They also noticed that the temporal patterns varied over time with frequency, volume, and distribution. Popularity/legitimacy tends to have a high in-degree, while spam tends to have a high out-degree. Spammers also were shown to have three-times the total number of followers and friends than legitimate users, and they replied and tweeted slightly more frequently than legitimate users. The aspect that I found the most

fascinating is that spammers were likely found on the edges of a Twitter graph, rather in the center. This is because it is easier to follow many accounts at once if you are not at the center.

2. Data

The dataset that I am using is from www.kaggle.com and it is called The Fake Project dataset was used in the "The Paradigm-shift of social spambots: Evidence theories and tools for the arms race" [4]. It is a dataset of Twitter spambots and genuine accounts. The data is already classified as either fake follower, genuine accounts, social spambots, or traditional spambots. Within each category there are tweets and users. Since everything is in its own file, I will mix some from each into one file and randomize it. There is no legend and I must deduce what each column name means. When I trained the model, I used the library **CaTools** and did a 1/3 as a training set and 2/3 as a testing set.

The following are both headers I added and ones that were already there: spambot1, spambot, name, screen_names, statuses_counts, friends_count, ratio, favourites_count, listed_count, url, lang, time_zone, location, default_profile, default_profile_image, profile_image_url, profile_banner_url, profile_use_background_image, profile_background_image_url_https, profile_text_color, profile_image_url_https, profile_sidebar_border_color, profile_background_tile, profile_sidebar_fill_color, profile_background_image_url, profile_background_color, profile_link_color, utc_offset, is_translator, follow_request_sent, notifications, descriptions, contributors_enabled, following, created_at, timestamp, crawled_at, updated, order.

Since most of these were either null, empty, or I had no idea what they were, I decided to make the dataset smaller and get rid of most of these predictors. I chose the ones that were full and didn't have nulls or non-entries. The features that I used were spambot1 (the binary representation of being a spambot), spambot (ratio < 0.5 = 1, otherwise 0), statuses_count (number of status updates made by account), followers_count (number of other accounts that follow this account), friends_count (number of friends the account has), ratio (number of followers relative to the number of followers and friends combined), favourites_count (number of accounts this account has favorited), listed_count (number of accounts that has put this account on an organized list).

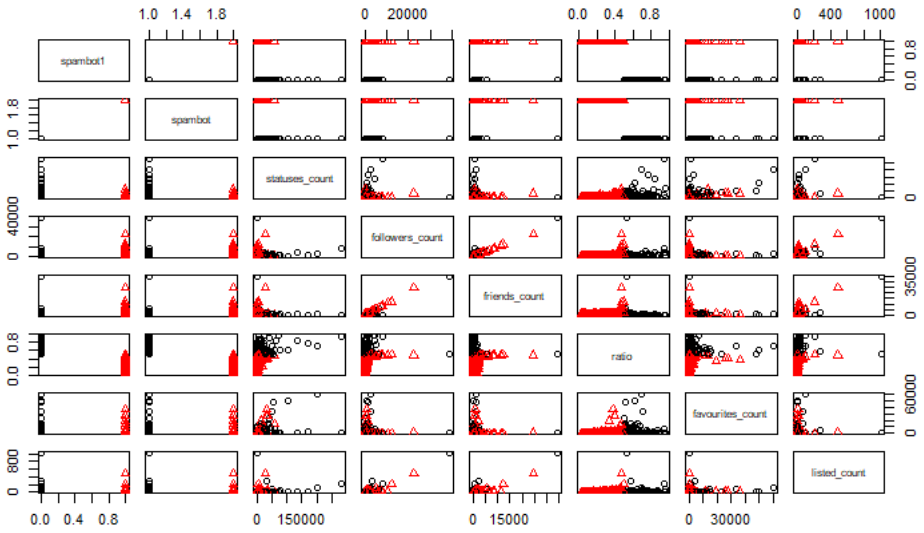
Here is the summary for the model with all the predictors:

```
glm(formula = spambot1 ~ statuses_count + followers_count + favourites_count +
    listed_count + friends_count, family = binomial, data = project)

Deviance Residuals:
      Min         1Q     Median         3Q        Max
-1.974e-03  -2.000e-08   2.000e-08   2.000e-08   1.863e-03

Coefficients:
                  Estimate Std. Error z value Pr(>|z|)
(Intercept)       9.049e+00  4.937e+02   0.018    0.985
statuses_count    5.214e-04  5.717e-02   0.009    0.993
followers_count  -6.331e+00  1.067e+02  -0.059    0.953
favourites_count -3.124e-04  1.955e-01  -0.002    0.999
```

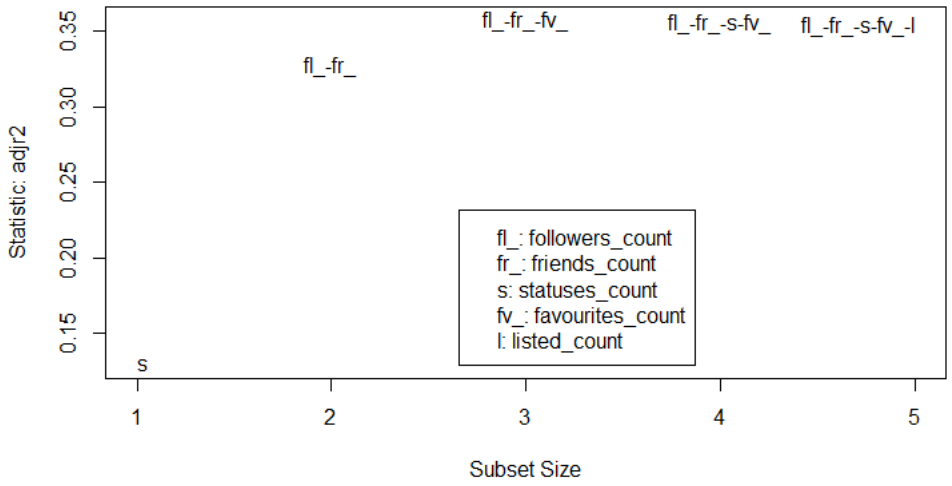
```
listed_count     -1.937e-01  2.110e+02  -0.001    0.999
friends_count     6.273e+00  1.057e+02   0.059    0.953

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 4.1368e+02  on 339  degrees of freedom
Residual deviance: 1.4206e-05  on 334  degrees of freedom
AIC: 12

Number of Fisher Scoring iterations: 25
```

None of these predictors are significant based on the p-value of all of them being > 0.05

I extracted features by first deciding which ones would work for the problem at hand. I did a pairs plot to see how all the features correlated towards each other. I also made the response "spambot" = 1, red to differentiate from when it was 0. The pairs plot is below:



There is clearly a "line" that separates this dataset. It was already classified, and I did my best to try to separate it and make it less contrived. After this I decided to use the subset selection process from the library **leaps**. It chose followers_count, friends_count,and favourites_count as the best predictors for the model according to the highest adjusted R^2 between the model with only 1 to the model with all of the predictors. The only linear relationship that exists is between followers_count and friends_count. These will prove to be the best predictors.



I then decided to do backwards selection according to Akaike's Information Criterion (AIC) and Bayesian Information Criterion (BIC) using the step function from the library **MASS**. Below is the AIC model:

```
tart:  AIC=12
spambot1 ~ statuses_count + followers_count + friends_count +
    favourites_count + listed_count

                Df Deviance    AIC
- listed_count      1     0.00  10.00
- favourites_count  1     0.00  10.00
- statuses_count    1     0.00  10.00
<none>                    0.00  12.00
- followers_count   1   279.87 289.87
- friends_count     1   329.46 339.46

Step:  AIC=10
spambot1 ~ statuses_count + followers_count + friends_count +
    favourites_count

                Df Deviance    AIC
- favourites_count  1     0.00   8.00
- statuses_count    1     0.00   8.00
<none>                    0.00  10.00
- friends_count     1   329.46 337.46
- followers_count   1   329.53 337.53

Step:  AIC=8
spambot1 ~ statuses_count + followers_count + friends_count
```

```
                Df Deviance    AIC
- statuses_count   1     0.00   6.00
<none>                   0.00   8.00
- friends_count    1   329.51 335.51
- followers_count  1   329.53 335.53

Step:  AIC=6
spambot1 ~ followers_count + friends_count

                Df Deviance    AIC
<none>                   0.00   6.00
- friends_count    1   408.87 412.87
- followers_count  1   411.78 415.78
```

The final model is **spambot1 ~ followers_count + friends_count**
Next is the BIC selection:

```
Start:  AIC=28.42
spambot1 ~ statuses_count + followers_count + friends_count +
    favourites_count + listed_count

                 Df Deviance    AIC
- listed_count      1     0.00  23.68
- favourites_count  1     0.00  23.68
- statuses_count    1     0.00  23.68
<none>                    0.00  28.42
- followers_count   1   279.87 303.55
- friends_count     1   329.46 353.14

Step:  AIC=23.68
spambot1 ~ statuses_count + followers_count + friends_count +
    favourites_count

                 Df Deviance    AIC
- favourites_count  1     0.00  18.94
- statuses_count    1     0.00  18.94
<none>                    0.00  23.68
- friends_count     1   329.46 348.41
- followers_count   1   329.53 348.48

Step:  AIC=18.94
spambot1 ~ statuses_count + followers_count + friends_count

                Df Deviance    AIC
- statuses_count   1     0.00  14.21
<none>                   0.00  18.94
```

```
- friends_count    1   329.51 343.71
- followers_count  1   329.53 343.74

Step:  AIC=14.21
spambot1 ~ followers_count + friends_count

                Df Deviance    AIC
<none>                0.00  14.21
- friends_count    1   408.87 418.34
- followers_count  1   411.78 421.25
```

Again, BIC selects **spambot1 ~ followers_count + friends_count**

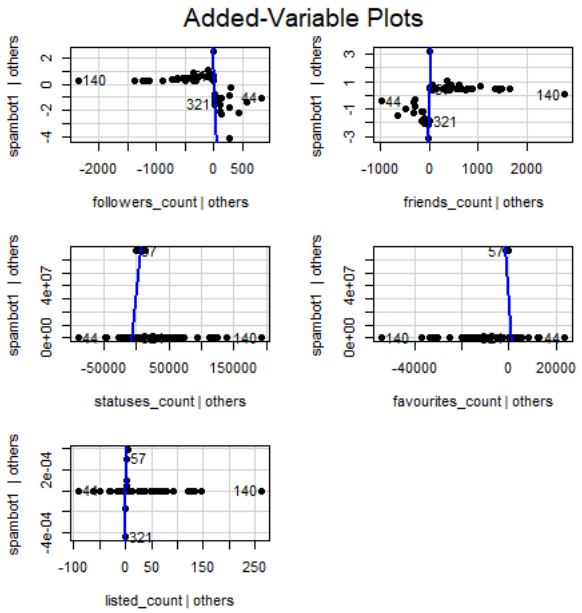
BIC penalizes the model more than AIC does, and normally is allows in less predictors. AIC balances the goodness of fit and penalizes based on model complexity. The lower the value of AIC the better the model. It starts with the full model and eliminates one variable at a time until it gets to the best model. BIC performs the same as AIC except it uses a log(n) penalization so it is stricter on what it allows into the model. Like AIC, the smaller the value the better the model. Since only 5 predictors are being used, they both agree on the same 2 variables. I could have done this by hand since there are only 5 predictors and I would have made 31 models since I wouldn't have the empty set because that would predict nothing.

The summary for the reduced model is below:

```
glm(formula = spambot1 ~ followers_count + friends_count, family = binomial,
    data = train.project)

Deviance Residuals:
      Min         1Q     Median         3Q        Max
-1.305e-03  -2.000e-08   2.000e-08   2.000e-08   1.208e-03

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept)      9.663    655.998   0.015    0.988
followers_count  -6.371    206.904  -0.031    0.975
friends_count     6.313    205.172   0.031    0.975

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1.3894e+02  on 113  degrees of freedom
Residual deviance: 3.4488e-06  on 111  degrees of freedom
AIC: 6
```

Again, nothing is significant, but the dataset is contrived.

I also decided to do an added variable plot from the library **cars.** This checks each variable's effect on the model independent from the other variables:



They all affect the model according to this because none are horizontal. If they were horizontal (or equaling 0) then that means they do have no affect on the model. This is odd seeing the p-values from the summaries, but I suspected multicollinearity. I chose to run the variance inflation factor (vif) again from the **cars** library.

```
 statuses_count  followers_count favourites_count     listed_count    friends_count
       3.012045     15041.082764         2.316840         1.871483     15034.149000
```

Clearly followers_count and friends_count are inflated because they are extremely large (larger than 5). Ideally one would take both of those out of model, so I took them out and decided to only run a model with statuses_count, favourites_count, and listed_count.

```
glm(formula = spambot1 ~ statuses_count + favourites_count +
    listed_count, family = binomial, data = project, subset = train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.14460  -0.00577   0.52859   0.53266   2.58908

Coefficients:
                  Estimate Std. Error z value Pr(>|z|)
(Intercept)      1.890e+00  3.271e-01   5.778 7.57e-09 ***
statuses_count  -1.795e-04  5.420e-05  -3.311 0.000929 ***
favourites_count  5.919e-06  9.767e-05   0.061 0.951679
listed_count      1.022e-02  2.658e-02   0.385 0.700520
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)
```

```
    Null deviance: 138.936  on 113  degrees of freedom
Residual deviance:  91.952  on 110  degrees of freedom
AIC: 99.952

Number of Fisher Scoring iterations: 7
```

Now we have one significant predictor and that is statuses_count. I decided to stick with this model.

When I did the data analysis portion of the project, I used observations (rows). For my final I did 340 observations so 242% more data. It originally caused problems as any new observations has a potential of causing a conflict with the previously written code.

3. Analysis

I used 3 different machine learning algorithms. I used Linear Discriminant Analysis (LDA), Logistic Regression, and K-Nearest Neighbors (KNN). Logistic Regression was used, and I ended up with a 24% training error, which means that I only classified 76% correctly.

```
         spam.test
glm.pred2   0   1
       No   23   8
       Yes  44 151
> 1-sum(diag(g2))/sum(g2)
[1] 0.2300885
```

Logistic Regression from the full model

```
         spam.test
glm.pred   0   1
     No   66   5
     Yes   1 154
> 1-sum(diag(g1))/sum(g1)
[1] 0.02654867
```

97.4% of the data is classified correctly

I then decided to use LDA from the library **MASS.** With this I ended up with a 27% training error, it was getting worse.

```
          spam.test
lda.class   0   1
        0  13   3
        1  54 156
> 1-sum(diag(g3))/sum(g3)
[1] 0.2522124
```

75% was modeled correctly for reduced model

LDA for the full model was

```
        spam.test
lda.class   0   1
        0  34   2
        1  33 157
> 1-sum(diag(g5))/sum(g5)
[1] 0.1548673
```

84.6% of the data is classified correctly

Lastly, I did KNN from the library **Class**. For k=1 the training error was 40%

```
     spam.test
knn.pred   0   1
       0  42  26
       1  25 133
> 1-sum(diag(g4))/sum(g4)
[1] 0.2256637
```

I played around with multiple values of k

```
#k=25
> knn.pred=knn(train.x,test.x,train.spam,k=25)
> table(knn.pred,spam.test)
        spam.test
knn.pred   0   1
       0  37  12
       1  30 147
> 1-mean(knn.pred==spam.test)
[1] 0.1858407
> #k=10
> knn.pred=knn(train.x,test.x,train.spam,k=10)
> table(knn.pred,spam.test)
        spam.test
knn.pred   0   1
       0  34  14
       1  33 145
> 1-mean(knn.pred==spam.test)
[1] 0.2079646
> #k=50
> knn.pred=knn(train.x,test.x,train.spam,k=50)
> table(knn.pred,spam.test)
        spam.test
knn.pred   0   1
       0  11   2
       1  56 157
> 1-mean(knn.pred==spam.test)
[1] 0.2566372
> #k=20
> knn.pred=knn(train.x,test.x,train.spam,k=20)
> table(knn.pred,spam.test)
        spam.test
knn.pred   0   1
       0  43  16
       1  24 143
> 1-mean(knn.pred==spam.test)
[1] 0.1769912
> #k=15
> knn.pred=knn(train.x,test.x,train.spam,k=15)
> table(knn.pred,spam.test)
        spam.test
knn.pred   0   1
       0  44  18
       1  23 141
> 1-mean(knn.pred==spam.test)
[1] 0.1814159
```

The best training error I could achieve was using k=22

```
         spam.test
knn.pred   0   1
       0  44  14
       1  23 145
> 1-mean(knn.pred==spam.test)
[1] 0.1637168
```

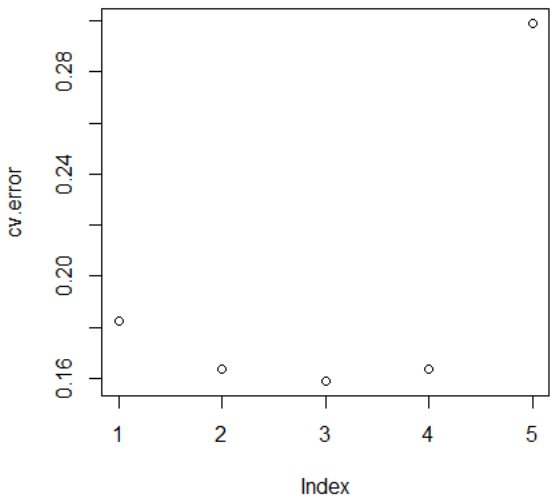
84% of the data was correctly classified and k=22 was the smallest value of k that could be used before the error rose again.

Logistic Regression gave the best accuracy. As for cross-validation I chose to use leave-one-out and used the library **boot**. The below deltas are the raw leave-one-out error rate and the other is the bias-corrected rate, which you can see that they are nearly identical . The deltas for this were

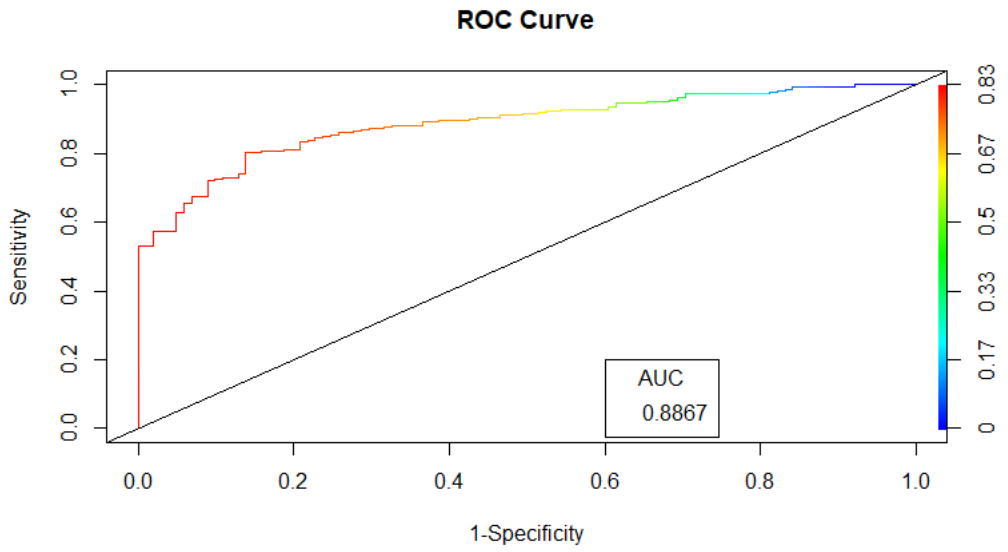
```
cv.err$delta
[1] 0.1625351 0.1625373
```

I also used k-folds validation with k=5, below are the deltas and the plot

```
> cv.error
[1] 0.1826122 0.1636223 0.1590807 0.1635463 0.2989844
```



The Receiver operating characteristic (ROC) plots the true positive rate against the false positive rate. I used the libraries **ROCR** and **nnet** to plot this curve.



Sensitivity is the true-positive rate, and 1-Specificity is the false-positive rate. The area under the curve (AUC) is large and almost 1, so it shows that the model is performing well. I couldn't do it with multiple models because I kept getting errors that said

```
Error in prediction(pred, project$spambot1) :
  Format of predictions is invalid.
```

4. Conclusion

The inferences that was reached by the machine learning algorithms were that logistic regression performed the best on this particular dataset. Using all predictors gave the best training error but this dataset was pieced together to make it work. The number of followers and friends are inner-correlated. There might be endogeneity as they can be chosen at the same time. Though status, favorites, and listed counts are not bad to use they do not produce the same results as if all of the predictors were used together. However, when all were used together the model itself wasn't valid.

Statuses_count is the most significant predictor with a p-value = 0.000929. The summary says that for every status that the account tweets, the likelihood of being a spambot drops by -1.8, so the more statuses an account has the less likely it is to be a spambot, according to the model. This was the only predictor that was significant, so one can infer that this is what needs to be examined, when determining the legitimacy of a Twitter account. However, this is in direct opposition of all of the research that has been done on this subject.

5. Reflections

While working independently, I learned that I need to get more proficient and comfortable with R. If I wasn't currently taking another class in conjunction with this one, where I used R, I would have been more behind. I learned to get better at Google to figure out what my what most of my errors meant. I also learned that sometimes I need to take a step back and think through the problems to figure out where I am going wrong. This project was fun as I never

really thought about the concept of classifying accounts in any social network, as being fake. I know understand that this is not something that is only affects Twitter. This also doesn't just affect social networks. Any network can fall victim to fraudulent accounts.

I learned a lot through out this project. I liked the different iterations for assignments. It kept me on schedule and I was able to move forward after each step. Each step narrowed the scope. The first step was the preliminaries. The purpose of this assignment was to get an understanding of which cybersecurity concept I wanted to explore. I knew I wanted to do a project on social engineering. I had originally wanted to work with password strengths and the classification of them. This was an extremely broad topic.

The second one was the project background. Here I had to find articles and summarize them. This was when I decided to classify spambots using the friends and followers ratio. I had originally wanted to use a graph and look at the edges between nodes and their friends and followers, that's when I learned that Twitter does not like people having access to their data, especially the tweets themselves. I had to scour the web to find a dataset that would work. I found one from Kaggle.

The prototype phase was next. I had decided on my main goal and the sub-problems to get to that goal. I learned that feature extraction was a lot harder than it looked. I didn't have any results at this moment as all I had were errors. I should have made a synthetic dataset at this stage, but I was being stubborn and did not.

The next phase was the data analysis one. Finally, I had results to show. I was able to show plots. I knew what features I wanted to use. I dropped the rest. This was when I started understanding the difference between the classification models. I know which one is best based on the data that I had, but I know that best it relative. A different set of observations would give me a different algorithm that is best. The problems that I ran into was trying to do too much at once. Instead of me getting one thing working and then moving forward, I wanted to do it all at once. I would get multiple errors that were dependent upon something that had an error before it. That was I wasn't able to get things working before. I had to learn to work in segments. Once I was able to do the work piecewise it eventually came together.

Lastly was this final. I couldn't get the ROC curve to work during the data analysis section. I watched a few YouTube videos and figured out what I was doing wrong. My data wasn't formatted correctly. I also understood what the area under the curve meant for classifications. The higher the area the better the model. This intuitively made sense because the line goes through the origin is at 0.5 and this is a guess. I was also able to get the k-folds cross-validation to work. I was creating the method wrong for doing each degree for the polynomial.

If I had another 6 months, I would add in natural language processing (NLP) for the categorical predictors. The original dataset had lots of categorical data, but it needed to be cleaned as there were a lot of empty values. I would have also added in classification trees. I would have done more cross-validation, possibly bootstrapping or resampling. I would have found another dataset that wasn't pre-classified and tested my methods on it and then compared it to these results.

If I had to redo this project, the first thing I would change is the data itself. I would use Twitter's API, instead of relying on someone to gather my data. I don't know how the data was

gathered. I then would not have an already classified dataset. I would be one that is unbiased, and true unsupervised learning. I would then use k-means clustering and had k=2 since I only had 2 classes. I would have done a better job of selecting features and not just used continuous ones. I would have also use categorical predictors and used NLP. The model would have more than 5 predictors and it would have been more robust. I would also have asked for help sooner, and not waited, and if my original data wasn't working, I would have just created my own artificial dataset.

References:

1. Chu, Zi, et al. “Detecting Automation of Twitter Accounts: Are You a Human, Bot, or Cyborg?” *IEEE Transactions on Dependable and Secure Computing*, vol. 9, no. 6, 2012, pp. 811–824., doi:10.1109/tdsc.2012.75.

2. Dickerson, John P., et al. “Using Sentiment to Detect Bots on Twitter: Are Humans More Opinionated than Bots?” *2014 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2014)*, 2014, doi:10.1109/asonam.2014.6921650.

3. Uddin, Muhammad Moeen, et al. *Understanding Types of Users on Twitter*. 5 June 2014, arxiv.org/pdf/1406.1335.pdf.

4. Wang, Alex Hai. “Detecting Spam Bots in Online Social Networking Sites: A Machine Learning Approach.” *Lecture Notes in Computer Science Data and Applications Security and Privacy XXIV*, 2010, pp. 335–342., doi:10.1007/978-3-642-13739-6_25.

5. Yardi, Sarita, et al. “Detecting Spam in a Twitter Network.” *First Monday*, vol. 15, no. 1, 2009, doi:10.5210/fm.v15i1.2793.