Michael Lampard

Capstone Project

Detecting Fake Follower Accounts on Twitter

Project Overview

“When you give everyone a voice and give people power, the system usually ends up in a really good place. So, what we view our role as, is giving people that power.” Zuckerberg to ABC's Diane Sawyer in 2010 [1]

Mark Zuckerberg’s statement has aged like a pint of milk left out on a hot day. Although social media platforms have offered a voice to the masses, there is still a power imbalance and a dark underbelly that has come into focus recently with events such as Cambridge Analytica’s wonton use of Facebook user data (2015), Russian Twitter Troll bots attempting to impact 2016 US electoral results and a botnet used during the UK’s Brexit referendum by Vote Leave (2016)[2] attempting to push hyper-partisan views into public discourse.

The ability of social media to shape public opinion has led to a new field of bot detection where both academia and business aim to understand and detect non-human actors. bots can be defined as a simple static fake account used to boost follower numbers to complex networks acting in unison masquerading as well supported organic movements.

This Capstone project will focus on the popular micro-blogging platform Twitter and deploy a Machine Learning (ML) model using a Sagemaker endpoint to discern fake followers from real humans. This information is important for business as paying for sponsored tweets to bots is a waste of money and important to the user as a large follower count could be used to persuade others of one's legitimacy (“clout”). We, the user, the main product social media platforms sell need tools to help us gauge our digital interactions.

Problem Statement

In a world increasingly governed by user engagement metrics, one can add an air of legitimacy to their message by boasting a healthy follower count. This incentivises would-be influencers to purchase fake followers to increase their clout in this digital arena where all manner of third parties (politicians, think tanks, businesses, social movements etc) go head to head to gain public support to achieve their individual goals.

In a progressively polarised world where social media has become the preferred battleground of ideas, users should rightfully have the ability to assess the legitimacy of an account, a problem this project aims to help solve by training a model to detect fake followers to later be integrated into a follower quality report web application.

[1] https://newrepublic.com/article/147697/mark-zuckerberg-lost-genius#:~:text=%E2%80%9CThe%20power%20of%20democracy%20in,ABC's%20Diane%20Sawyer%20in%202010
[2] https://journals.sagepub.com/doi/pdf/10.1177/0894439317734157

Benchmark Model

Although there are commercially available bot detectors they seldom expose their methodologies, for this reason, the models examined in the paper “Detecting Fake Followers in Twitter: A Machine Learning Approach” [3] will serve as our benchmark alongside Null Accuracy[4].

Algorithm	Accuracy Score
SVM	60.48%
Simple Logistic	90.02%
Instance-based classifier using 1 nearest neighbour	98.74%
Null Accuracy	50.90%

Evaluation Metrics

To allow a fair comparison with the benchmark models Accuracy Score will be the primary evaluation metric calculated using the below formula.

$$accuracy\_score = \frac{correct\ predictions}{number\ of\ predictions}$$

Analysis

Data Exploration

The dataset for this project consisted of a subset of data from the Cresci-2017[5] dataset, a collection of genuine users (human) , fake followers, social spambots and traditional spam bots in .csv format. The subsets described below were used in training, validating and testing our model.

Dataset name	Description	Size	Target
fake_followers.csv	A collection of fake accounts bought from three different providers in April 2013 - fastfollowerz.com, intertwitter.com, and twittertechnology.com	3,351	1

[3] http://ijaema.com/gallery/69-july-2162.pdf
[4] Accuracy when we always predict the most frequent class.
[5] https://botometer.osome.iu.edu/bot-repository/datasets.html

genuine_accounts.csv	A random sample of human accounts generated by contacting users with a simple question in natural language, verified by a human.	3,474	0

For the purpose of this study, Fake Followers (bots) were assigned a target of 1 and Genuine Accounts were assigned a target of 0. Once we had concatenated the two subsets we were left with 6835 observations with 38 features and one binary target to predict against.

Data Cleaning

The Cresci-2017 dataset was built using an older version of the Twitter API, specifically v1 from 2013. Since 2013 Twitter has changed the structure of its user object, deprecated and renamed a number of features, to ensure our model is forward compatible with the new Twitter API fields the data cleaning steps described below were performed.

Firstly the following features in our training and test datasets were renamed.

Cresci-2017	Twitter API v2
screen_name	username
friends_count	following_count
statuses_count	tweet_count

Figure 1 shows the intersection between Cresci-2017 dataset and the new Twitter v2 API[6]

[6] https://developer.twitter.com/en/docs/twitter-api/data-dictionary/object-model/user

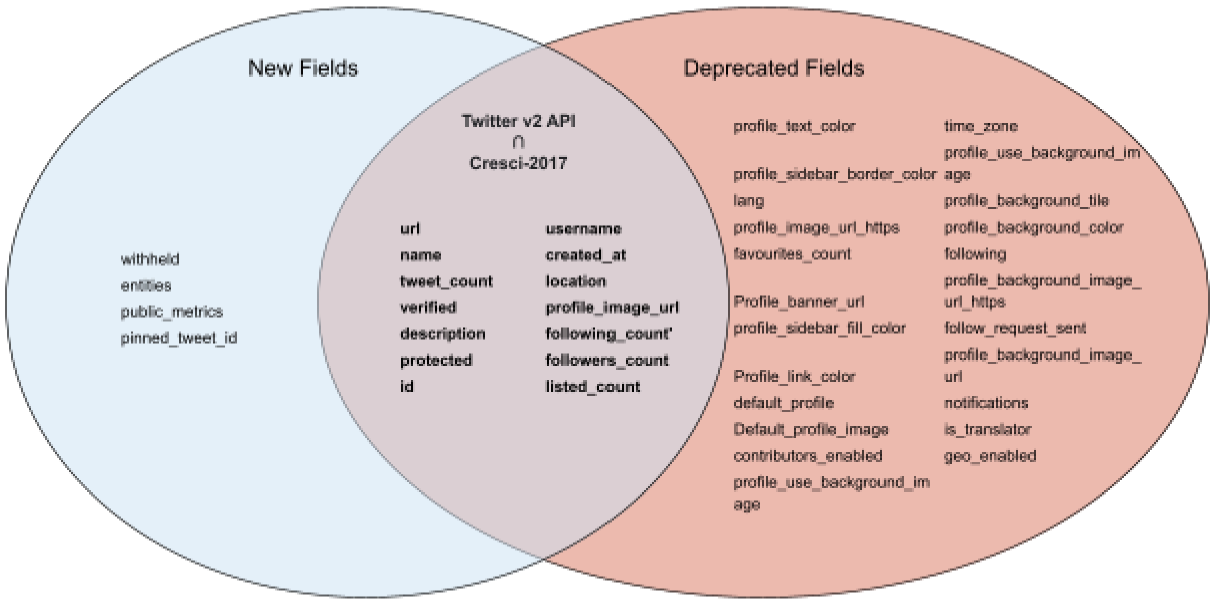


Figure 1: Venn diagram displaying intersection between Cresci-2017 dataset and Twitter v2 API

Since the dataset was built in 2013, 23 fields have been deprecated, 4 fields have been added and 14 fields have remained available. We will only keep the remaining 14 fields.

Further to the above, “id” was removed from the dataset as it is a unique identifier and not a suitable feature. “profile_image_url” was also removed as image classification is outside the scope of the study.

Feature Engineering

Text/String Features

We have the below text-based features in our dataset that must be engineered to be useful to our ML model. As Natural Language Processing (NLP) is outside the scope of this study ”'location”, “description” and “url” were converted to boolean values with 1 insinuating the feature is present.

Text/String Features:”'location”, “description”, “url”, “name”, “username”,

Name Containment

A username to name containment feature (Jaccard Similarity) that describes whether the “username” contains the same word tokens as the “name”. The assumption here is that for fake followers these two fields will be similar as generating a different value for each field could be harder to implement. Further to this, when you sign up for Twitter the “username” suggested is often a concatenation of first and last name. An easy way around this would be to introduce some random characters into the suggested name. To account for this a containment function was used as opposed to a Levenshtein distance approach that would not appreciate this quirk. The below algorithm was used.

1. Make all characters lowercase in “username” (PatLam99 -> patlam99) and “name” (Pat Lam -> pat lam).
2. Split “name” into tokens e.g “Pat Lam” -> name_list = [‘pat’, ‘lam’]
3. For each token in the name_list
    a. Check if that sequence of characters is present in the “username”.
        i. If present count += 1
4. Calculate containment:

$$\frac{\Sigma count(ngram_{name}) \cap count(ngram_{username})}{\Sigma count(ngram_{name})}$$

Numerical Features

Count features

Our count features were converted to per_day ratios. This is done because a simple count can be misleading. Younger accounts will tend to have fewer tweets, followers and friends because they have not had time to perform these actions. For example, an account may only have 10 tweets but is 10 days old meaning it sends 1 tweet per day, conversely an old account could also have 10 tweets but be 100 days old giving us 0.1 tweets per day. In this scenario, although tweet counts are the same, it is evident the younger account tweets more regularly.

"followers_to_friends" ratio

As pointed out in (Z. Chu et al, 2010)[7], we can often gain more information from engineered features such as a “follower to friends ratio” where humans exhibit a ratio closer to 1, compared to fake followers that tend to 0. This difference could be attributed to bots not posting regularly, not posting organic and engaging content, meaning they garner less followers.

Final Features

Having cleaned our data and engineered features thought to be useful in detecting fake followers we are left with the below feature space consisting of 7 numerical features, 5 boolean features and our target (1,0)

[7] Z. Chu, S. Gianvecchio, H. Wang, and S. Jajodia. Who is tweeting on twitter: Human, bot, or cyborg? In Proceedings of the 26th Annual Computer Security Applications Conference, ACSAC ’10, pages 21–30, New York, NY, USA, 2010. ACM.

Feature	Type	Description
target	boolean	The target we are predicting against i.e fake followers (1) and genuine accounts (0)
listed_count	int	Number of lists an account has created
location	boolean	1 if location is provided
description	boolean	1 if account has a description
protected	boolean	1 if accounts tweets are protected
url	boolean	1 if account has a URL
verified	boolean	1 if an account is verified
name_containment	float	Jaccard similarity between “name” and “username”
account_age_days	int	Account “created_at” - “updated” (day account was scraped
followers_per_day	float	follower_count/account_age_days
tweets_per_day	float	tweet_count/account_age_days
following_per_day	float	following_count/followers_count
followers_to_friends	float	following_count/friends_count

Exploratory Analysis

The first step was to check if we have a class imbalance i.e where one target class represents a significantly larger proportion of our observations. A class imbalance can result in a model with poor predictive performance on the monor class.

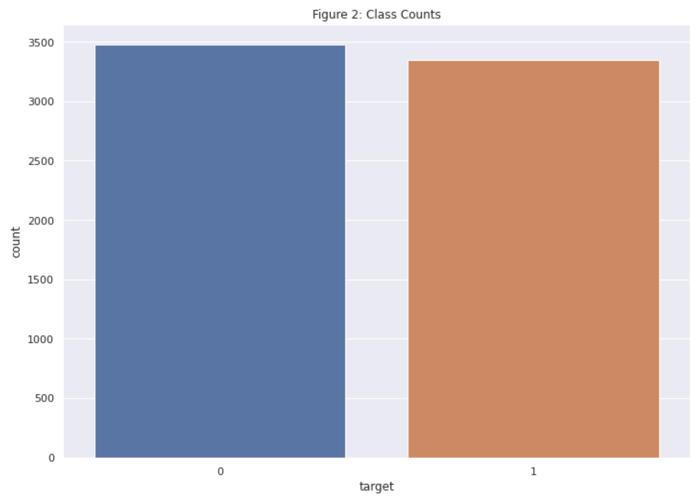


Figure 2 shows a slight imbalance between the classes genuine account (0) and fake follower (1), which is not a concern meaning we shouldn't have to employ oversampling or resampling techniques.

Categorical Features

feature_name	genuine accounts	fake followers
location	0.68%	0.83%
description	0.89%	0.68%
url	0.36%	0.03%
verified	0.00%	0.00%
protected	0.02%	0.00%

From the table above it is apparent that fake followers are more likely to have a location, whereas genuine users are more likely to have a description and a URL in their profile. It is also apparent that “verified” and “protected” users are underrepresented in our dataset, a total of 11 and 78 respectively, exclusively found in genuine users.

Continuous Features

To check for multicollinearity we plotted the correlations between continuous features.

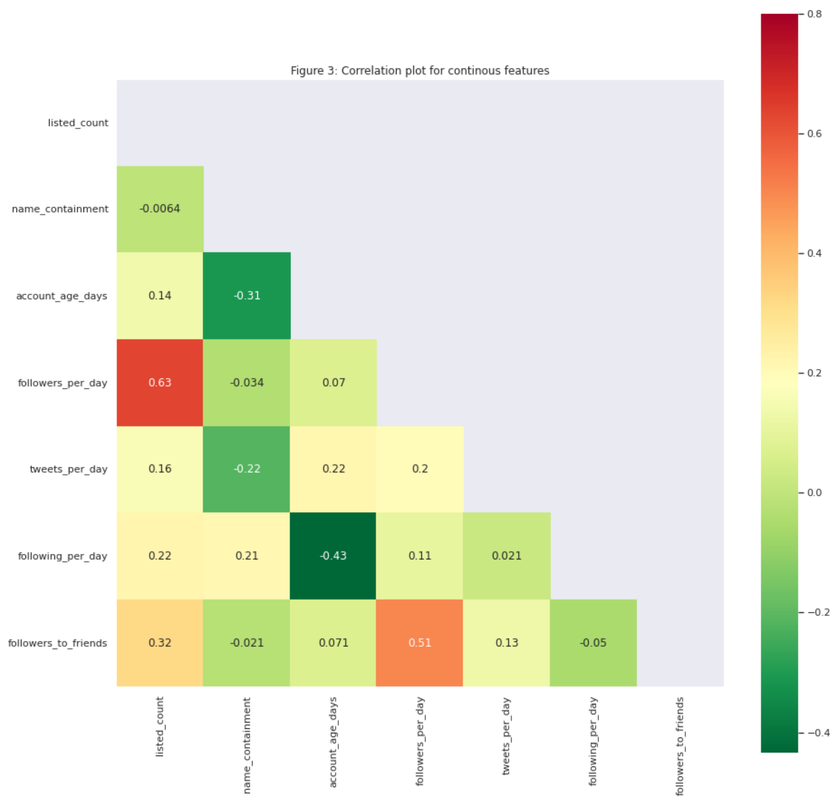
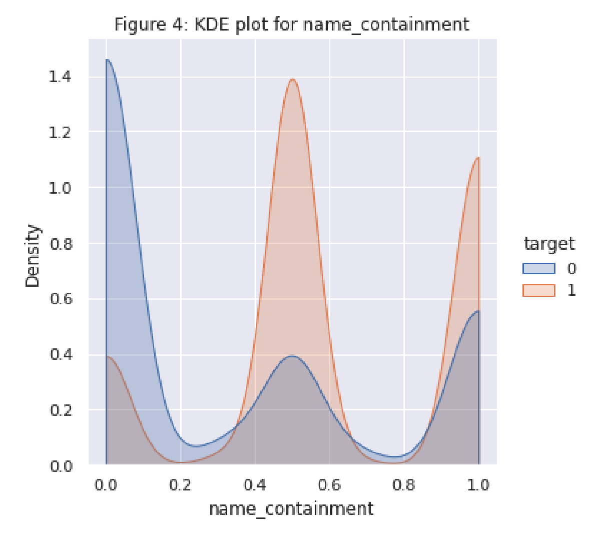
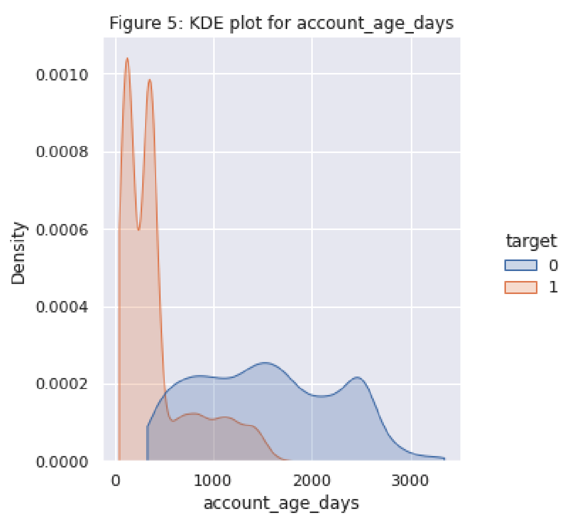


Figure 3 shows that our strongest correlations are between followers_per_day vs listed_count and followers_to_friends vs followers_perday, 0.61 and 0.51 respectively.

A study by Dormann, C. F., J. Elith, S. Bacher, et al. 2013[8] suggest a correlation threshold of r>0.7 as the cut off for multicollinearity, with this in mind all continuous features where shown to have a satisfactory correlation.

Having analysed a pair plot describing relationships between continuous features the below distributions stood out.





[8] Dormann, C. F., J. Elith, S. Bacher, et al. 2013. Collinearity: a review of methods to deal with it and a simulation study evaluating their performance. Ecography 36:27–46.

Figure 4 suggests fake followers (0) tend to have higher name_containment, meaning the username and the name of the account contain the same strings. This is expected as Twitter suggests a username similar to the name provided when signing up. When making a large number of bots it would be a costly extra step to make both of these original.

Figure 5 shows fake followers (1) tend to have younger accounts when compared to genuine users (0). Twitter often purges bot traffic from their user base meaning bot accounts tend not to last as long.

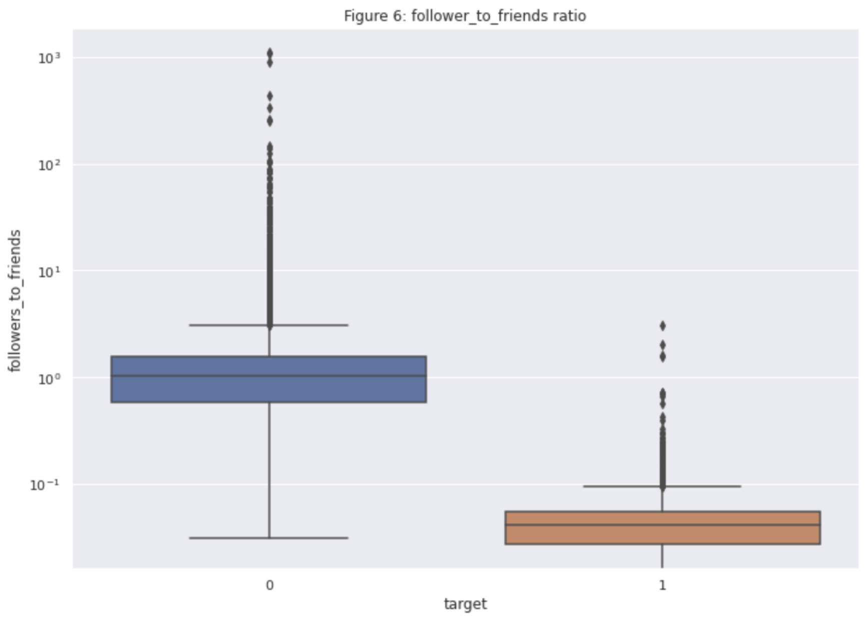


Figure 6 confirms our assumption that humans tend to have a follower_to_friends ratio closer to 1 where bots tend to 0.

Model Selection

A brief study using various models with default settings was used to assess which model will work best for our application.

model_name	accuracy_mean	accuracy_std	roc_auc_mean	roc_auc_std
RandomForestClassifier	0.989003	0.001922	0.999122	0.000547
SVC	0.982771	0.00364	0.99599	0.001578
LogisticRegression	0.980571	0.002556	0.996275	0.001375
KNeighborsClassifier	0.91954	0.007006	0.954054	0.005059

RandomForestClassifier (RF) was found to have the highest accuracy (99%) and lowest standard deviation in accuracy scores (0.0023), further to this RF exhibited the highest "roc_auc" score (area under receiver operator curve) meaning RF was able to maximise true positives whilst minimising false negatives.

For this reason, the Random Forest model was chosen to proceed with.

Feature Selection

Recursive Feature Elimination (RFE) was chosen as our feature selection method, in particular, the Boruta algorithm was used. RFE has advantages over feature importance methods as it doesn't require the user to select features based on an arbitrary feature importance threshold.

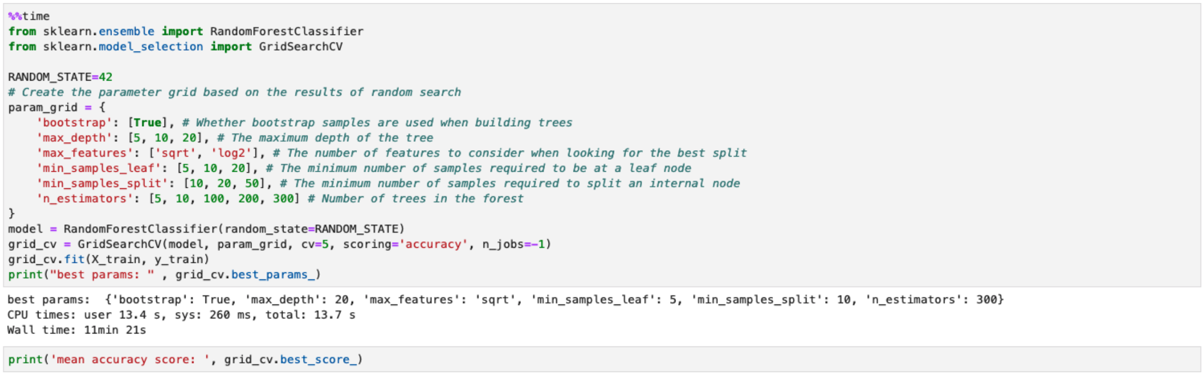
Results from Boruta suggest we can remove "verified" and "protected" as they do not influence our accuracy score. Both these features would usually be attributed to a real user. "Verified " users are accounts that command significant influence (influencers) where the verified blue tick confirms the account is who they say they are. Similarly, "protected" accounts can only be viewed by certain people, a trait one would associate with a real user. Both of these features are underrepresented in our dataset, to have the model learn the importance of these features we would need to oversample accounts containing them. In a production environment we could add a check for these two features before inferring from our model.

Results having removed "verified" & "protected"		
model_name	accuracy_mean	accuracy_std
RandomForestClassifier	0.989003	0.002389

The final feature set consisted of 10 features.

Model Implementation

Before training a SageMaker estimator our Random Forest model was first tuned to determine the best hyperparameters for future use. Here the Sklearn GridSearchCV library was used which exhaustively considers all parameter combinations defined in the param grid below.



```
%%time
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

RANDOM_STATE=42
# Create the parameter grid based on the results of random search
param_grid = {
    'bootstrap': [True], # Whether bootstrap samples are used when building trees
    'max_depth': [5, 10, 20], # The maximum depth of the tree
    'max_features': ['sqrt', 'log2'], # The number of features to consider when looking for the best split
    'min_samples_leaf': [5, 10, 20], # The minimum number of samples required to be at a leaf node
    'min_samples_split': [10, 20, 50], # The minimum number of samples required to split an internal node
    'n_estimators': [5, 10, 100, 200, 300] # Number of trees in the forest
}
model = RandomForestClassifier(random_state=RANDOM_STATE)
grid_cv = GridSearchCV(model, param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_cv.fit(X_train, y_train)
print("best params: " , grid_cv.best_params_)
```

```
best params:  {'bootstrap': True, 'max_depth': 20, 'max_features': 'sqrt', 'min_samples_leaf': 5, 'min_samples_split': 10, 'n_estimators': 300}
CPU times: user 13.4 s, sys: 260 ms, total: 13.7 s
Wall time: 11min 21s
```

```
print('mean accuracy score: ', grid_cv.best_score_)
```

Had our dataset been much larger and our grid search been more exhaustive we could have leveraged SageMaker's HyperparameterTuner wrapper and its parallelism, but with a wait time of only 10-20 minutes, this was not necessary.

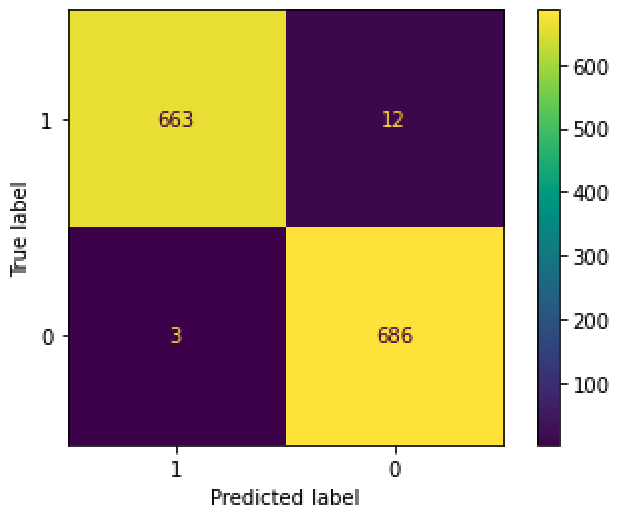
The best model and hence hyperparameters gave a mean accuracy score of 98.90%, which was above our null accuracy of 51%, meaning our model is able to differentiate between the two classes.

With our optimum hyperparameters, a SageMaker estimator was trained using the fit() method and deployed to an endpoint.

Model Accuracy

Having trained and deployed our RF model we were then able to test the accuracy of our model on the test.csv dataset that the model had not seen.

The final accuracy score of the model was 98.9%, similar to our training accuracy. The confusion matrix below displays the results.



From the above image, we can see our model resulted in 12 false positives and 3 false negatives, however, the vast majority of observations were correctly classified.

The table below outlines the sensitivity, the ability for our model to correctly identify a fake follower (1), and specificity, the ability of our model to identify a genuine account (0)

accuracy_metric	Formula	score
True_positive_rate/sensitivity	True Positives / (True Positives + False Negatives)	0.9955
False_Positive_rate	False Positives / (False Positives + True Negatives)	0.0044
Specificity	True Negatives / (True Negatives + False Positives)	0.9828

Both sensitivity and specificity were high enough to assume our model is capable of separating the two classes presented in the train and test data.

As our dataset was balanced further accuracy scores such as precision and recall were not considered.

Conclusion

During this study, we were able to train and deploy a Random Forest Classifier (RF) using AWS SageMakers Machine Learning infrastructure achieving an accuracy score of 98.90%

Algorithm	Accuracy Score
SVM*	60.48%
Simple Logistic*	90.02%
Instance-based classifier using 1 nearest neighbour*	98.74%
Null Accuracy*	50.90%
Random Forest	98.90%
* denotes baseline model	

From the above table, it is evident that our RF classifier achieved the highest accuracy closely followed by the KNN model from the baseline study.

Such high accuracy scores suggest the two classes used in training and testing are easily separable, a phenomenon we seldom see in practice. High test accuracy suggests our model is not overfitting to the training data.

Critical Analysis and Future Work

The high accuracy scores suggest this is an easy problem to solve using a binary classifier, however, the age of the Cresci-2017 dataset, constructed in 2013, could be the Achilles heel of this model were we to begin searching for fake followers in the current day. In 2022 fake followers still exist where a brief google search will bring you companies such as Boost Likes[9] offering 1000 “real” followers for £16.99, delivered within 24 hours of signing up. Social media companies such as Twitter are well aware of the presence of bots on their networks and as detection methods improve so do bots.

[9] https://boostlikes.co/

Unfortunately building the web app to allow users to generate a fake follower report turned out to be too large a scope for this study. Future work will involve building a web app and implementing a feedback mechanism where users can correct predictions were the model to predict a false positive. This kind of continuous learning from user input could help us to truly understand the pervasiveness of this problem.