CIS 563 - Introduction to Data Science

‘Fake Instagram Account Detection’

Name: Himanshu Avadhut Churi
SUID : 803477982

Introduction:

Social Media has had a great influence on people's lives in recent years. Various apps like Facebook, Twitter, Instagram are being used by a vast majority of people. Instagram has become an application found on almost every person's especially on every teenages mobile phone and is used for sharing thoughts, memories, personal moments and more and more new users are joining every day. It is estimated that Instagram has over 1.074 billion users. With this increasing user base people have started creating fake profiles and stalking genuine users, spreading hate, scamming genuine users, etc. Detecting fake accounts can help in preventing such crimes and requests from any such accounts can be blocked so that people can share their thoughts freely. So this project aims to create a model from real fetched data to detect fake accounts on instagram using information that is easily available to the user. I have used the XGBoost classifier on my fetched dataset to obtain a model with around 96% accuracy.

Prior Work:

1) Instagram Fake and Automated Account Detection [1]:

   The approach focuses on a lot of features of instagram accounts like, number of following and followers, positioned dates, frequency, comments, etc. On these features various algorithms like SVM, Naive Bayes, Logistic Regression,etc are applies. This provides an accuracy of 96%. Most of the features mentioned above are not available in case of private accounts and some of the fake accounts in order to pretend they are real have made accounts private so the availability of data to common users becomes a difficult problem in this case.

2) Methods for Detecting Fake Accounts on the Social Network VK [2]:

   In this paper the collection of the real users data from the VK was done manually. For collecting the data about the fake accounts, VK API was used. The various signs considered for detection are, user id, date of first post, if there is a profile photo or not? , number of subscribers, number of subscribed to, count of photos and video, etc. Various algorithms were implemented, such as the Decision tree method, Random Forest, Bernoulli naive Bayes, SVM, etc. Randomforest gave an accuracy of 97%, but it is not good at detecting fake accounts. Another better algorithm was Gaussian Networks, but it has a high margin of error. So we have to find another algorithm that is good at classification. One such algorithm is XGBoost.

Methodology:

A) Dataset and Preprocessing:

The data used for training the model is a supervised labeled data with 13 attributes (*follower_count, followed_count, bio, bio_Length, Bio_emoji_Count, #_tags_in_bio, Count_Uploads, isPrivate, Username_len, DigitsInUsername, FullName_len, has_ProfilePicture, Count_Mutual, isVerified*) and is divided into two parts. The first part consists of 693 json files of fake accounts obtained from this ***Link*** and the second part consists of 125 json files of real accounts gathered from my and my followers' instagram following list data from these accounts were extracted and formatted into numerical format as needed and the data sets were converted to CSV files.

1) Extracting data,transforming into creating CSV :

Important features for classifying an account are selected out of various features available in the json file and are converted and stored as a CSV file. These features include the ones listed above. These features were selected as they were easily available/ visible to any user and also could be easily parsed. The fact that that user Bio was not included because many users contain bio in different languages , use symbols (emojis) for bio and some don't even have a biography for both fake as well as real accounts.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
follower_cou	followed_cou	bio	bio_Length	Bio_emoji_C	#_tags_in_bi	Count_Uploa	isPrivate	Username_le	DigitsInUserr	FullName_le	has_ProfileP	Count_Mutu	isVerified	Class
461	1308	['Karma', 'be	26	7	0	264	1	13	0	17	1	4	0	0
370	487	['‡§≠‡§ö‡§∞	10	5	0	31	0	9	4	13	1	1	0	0
7730588	75	[]	0	0	0	103	0	8	0	12	1	32	1	0
106	217	[]	0	0	0	0	1	11	0	20	1	33	0	0
510392	201	['I', 'live', 'for	7	1	0	414	0	18	0	11	1	22	1	0
36	13	[]	0	0	0	2	1	8	0	0	1	35	0	0
740	502	[]	0	0	0	0	1	16	0	17	1	12	0	0
7025635	43	[]	11	0	0	188	0	15	0	10	1	7	1	0
356	500	['The', 'Magi	8	3	0	58	1	8	0	9	1	40	0	0
654	318	['Il', 'nest', 'ri	15	8	0	4	1	15	2	28	1	26	0	0
414	427	['Pune', 'Infu	14	1	0	8	1	10	0	18	1	20	0	0
659	868	[]	0	0	0	40	1	11	0	4	1	13	0	0
48	98	[]	0	0	0	0	0	15	2	13	1	20	0	0
3351839	16	[]	9	0	0	203	0	7	0	14	1	11	1	0
20196155	2	['oprahdaily'	6	7	0	1223	0	5	0	5	1	20	1	0
1914	1757	['Mrs', 'Mun	22	12	0	188	1	8	2	5	1	38	0	0
164	142	[]	1	2	0	177	1	10	0	11	1	11	0	0

Figure 1 : Fake Data

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
follower_cou	followed_cou	bio	bio_Length	Bio_emoji_C	#_tags_in_bi	Count_Uploa	isPrivate	Username_le	DigitsInUserr	FullName_le	has_ProfilePi	Count_Mutua	isVerified	Class
1231	728	['Ô¤óÔ¤ñÛΩ	14	23	8	30	0	15	4	7	1	0	0	1
75	1747	[]	2	0	0	22	0	11	2	3	1	0	0	1
1513	2661	['ýßÿµŸÑÿß',	13	1	0	3	0	17	0	4	1	0	0	1
97	3207	[]	0	0	0	22	0	14	1	0	1	0	0	1
133	7059	[]	0	0	0	4	0	12	0	0	1	0	0	1
1028	999	['ÿ™ŸÜŸáýßŧ	10	15	0	6	0	10	4	0	1	0	0	1
178	7502	[]	0	0	0	23	0	12	5	6	1	0	0	1
142	7501	[]	0	0	0	25	0	7	1	9	1	0	0	1
73	1994	[]	0	0	0	5	0	19	4	0	1	0	0	1
52	281	[]	4	0	0	3	0	10	3	8	1	0	0	1
70	3334	[]	0	0	0	4	0	6	0	0	1	0	0	1
125	7440	[]	0	0	0	10	0	11	6	0	1	0	0	1
77	429	[]	0	0	0	4	1	9	1	6	1	0	0	1
58	2925	[]	0	0	0	22	0	10	4	0	1	0	0	1
927	1725	[]	0	0	0	3	0	11	2	14	1	0	0	1
444	761	[]	0	0	0	8	0	11	0	19	1	0	0	1
62	937	[]	0	0	0	22	0	10	6	0	1	0	0	1
77	3139	['tehrantirra	3	3	0	22	1	10	4	5	1	0	0	1
48	885	[]	0	0	0	5	0	9	3	0	1	0	0	1

Figure 2: Real Data

2) Splitting:

As the total number of instances of each class is not in proper ratio it is important to sort proper number of training examples from the fake account dataset as a greater imbalance could cause the model to be biased towards that class. So we randomly select 130 instances out of 693 from that fake data set, and merge it with the real accounts dataset and then reshuffle it.

After loading Real Account data and the Fake data It is merged and then split in a ratio of 4:1

Splitting the dataset in the ratio of 4:1:

```
final_data
```

	follower_count	followed_count	bio_Length	Bio_emoji_Count	#_tags_in_bio	Count_Uploads	isPrivate	Username_len	DigitsInUsername	FullName_len	ha
0	1914	1757	22	12	0	188	1	8	2	5	
1	100	1930	0	0	0	4	0	11	4	0	
2	370	487	10	5	0	31	0	9	4	13	
3	1153	1291	0	0	0	67	1	12	0	0	
4	522886	707	6	0	3	1382	0	8	0	8	
...	...	...	...	...	...	...	...	...	...	...	
320	184	2515	0	0	0	4	0	12	4	0	
321	118	7363	0	0	0	4	0	7	0	0	
322	49	1451	0	0	0	6	1	12	2	11	
323	76	7181	0	0	0	22	0	9	4	0	
324	136	7235	1	0	0	10	0	15	1	0	

325 rows × 14 columns

Figure 3: Final Dataset

3) Selecting Important Features:

So in order to select the important features out of the ones I have in my dataset , I run the same classifier in a loop on the same amount of features and reduce the amount of features by 1 on every iteration.

```
Thresh=0.000, n=13, Accuracy: 95.38%
columns = ['follower_count' 'followed_count' 'bio_Length' 'Bio_emoji_Count'
 '#_tags_in_bio' 'Count_Uploads' 'isPrivate' 'Username_len'
 'DigitsInUsername' 'FullName_len' 'has_ProfilePicture' 'Count_Mutual'
 'isVerified']


Thresh=0.004, n=10, Accuracy: 95.38%
columns = ['follower_count' 'followed_count' 'bio_Length' 'Bio_emoji_Count'
 'Count_Uploads' 'isPrivate' 'Username_len' 'DigitsInUsername'
 'FullName_len' 'Count_Mutual']


Thresh=0.006, n=9, Accuracy: 95.38%
columns = ['follower_count' 'followed_count' 'bio_Length' 'Bio_emoji_Count'
 'Count_Uploads' 'isPrivate' 'Username_len' 'DigitsInUsername'
 'Count_Mutual']


Thresh=0.008, n=8, Accuracy: 95.38%
columns = ['follower_count' 'followed_count' 'bio_Length' 'Bio_emoji_Count'
 'Count_Uploads' 'isPrivate' 'DigitsInUsername' 'Count_Mutual']


Thresh=0.010, n=7, Accuracy: 95.38%
columns = ['follower_count' 'followed_count' 'bio_Length' 'Bio_emoji_Count'
 'isPrivate' 'DigitsInUsername' 'Count_Mutual']


Thresh=0.012, n=6, Accuracy: 95.38%
columns = ['follower_count' 'followed_count' 'bio_Length' 'isPrivate'
 'DigitsInUsername' 'Count_Mutual']


Thresh=0.013, n=5, Accuracy: 92.31%
columns = ['follower_count' 'bio_Length' 'isPrivate' 'DigitsInUsername'
 'Count_Mutual']


Thresh=0.014, n=4, Accuracy: 92.31%
columns = ['follower_count' 'bio_Length' 'isPrivate' 'Count_Mutual']
```

Figure 4: Selecting Important Features

Based on these new features (*'follower_count' , 'followed_count', 'bio_Length', 'isPrivate', 'DigitsInUsername', 'Count_Mutual'*), I have created a new data frame which will be used to make the model.

	follower_count	followed_count	bio_Length	isPrivate	DigitsInUsername	Count_Mutual
0	1914	1757	22	1	2	38
1	100	1930	0	0	4	0
2	370	487	10	0	4	1
3	1153	1291	0	1	0	61
4	522886	707	6	0	0	13
...	...	...	...	...	...	...
320	184	2515	0	0	4	0
321	118	7363	0	0	0	0
322	49	1451	0	1	2	0
323	76	7181	0	0	4	0
324	136	7235	1	0	1	0

325 rows × 6 columns

Figure 5: Reduced Data

B) Model and Training:

This newly created dataset is then passed through a XGBoost classifier with evaluation metric as error to create a model. In Gradient boosting, each new tree is fitted on the somewhat changed version of the original dataset[6]. It is an efficient way to convert a poor learner to a very good learner. A weak learner is the one whose performance is a little better than that of a random classifier. So in Gradient Boosting a weak learner is used several times[5]. XGBoost is a more regularized form of Gradient Boosting. XGBoost uses advanced regularization, which improves model generalization capabilities. It is a decision-tree-based Machine Learning algorithm which uses a gradient boosting framework. XGBoost is basically designed to enhance the performance and speed of a Machine Learning model.

The model trained using the XGBClassifier provides an accuracy of 98.46%.

```
In [31]: X_train1, X_test1, y_train1, y_test1 = train_test_split(New_Data, class_data,test_size = 0.2, random_state=42, shuff

In [32]: model1 = XGBClassifier(use_label_encoder=False,eval_metric = 'error')
         model1.fit(X_train1, y_train1.values.ravel())

Out[32]: XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
                       colsample_bynode=1, colsample_bytree=1, eval_metric='error',
                       gamma=0, gpu_id=-1, importance_type='gain',
                       interaction_constraints='', learning_rate=0.300000012,
                       max_delta_step=0, max_depth=6, min_child_weight=1, missing=nan,
                       monotone_constraints='()', n_estimators=100, n_jobs=8,
                       num_parallel_tree=1, random_state=0, reg_alpha=0, reg_lambda=1,
                       scale_pos_weight=1, subsample=1, tree_method='exact',
                       use_label_encoder=False, validate_parameters=1, verbosity=None)

In [33]: model1.save_model("model_small.json")

In [34]: y_pred1 = model1.predict(X_test1)
         predictions1 = [round(value) for value in y_pred1]

In [35]: accuracy = accuracy_score(y_test1, predictions1)
         print("Accuracy: %.2f%%" % (accuracy * 100.0))

         Accuracy: 98.46%
```

Figure 6: Accuracy

Results:

The correlation matrix, gives the correlation coefficient of each and every attribute with another. In the Figure a feature with the value nearer to 1 is the most correlated, which means that change in one feature increase/decreases the other in the particular proportion while the ones where the value is negative, it means that, change in one attribute will have negative effect on other value.

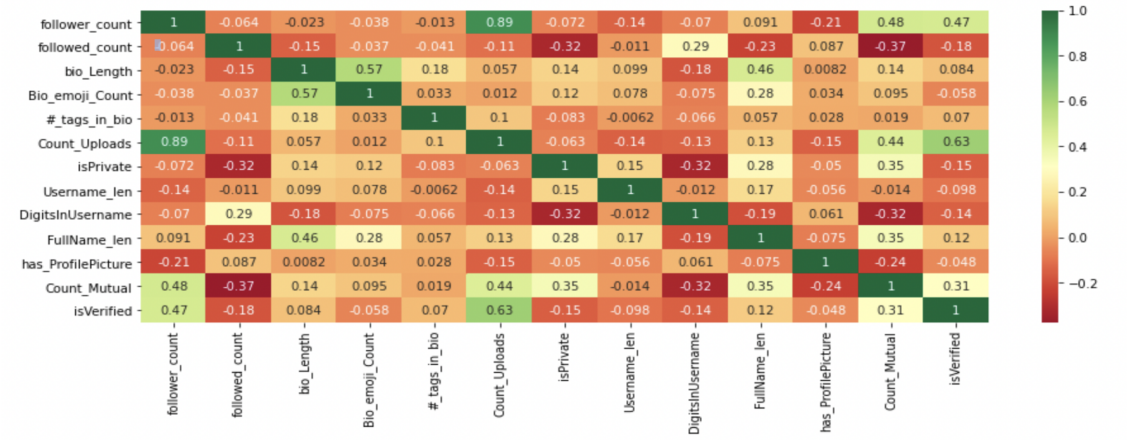


Figure 7: Correlation matrix of Original Dataset

After computing the important features out of the original total 13, I have computed and plotted the F1-score of the features and we find out that “followed_count” is the most important feature while “Count_Mutual” is the least important.

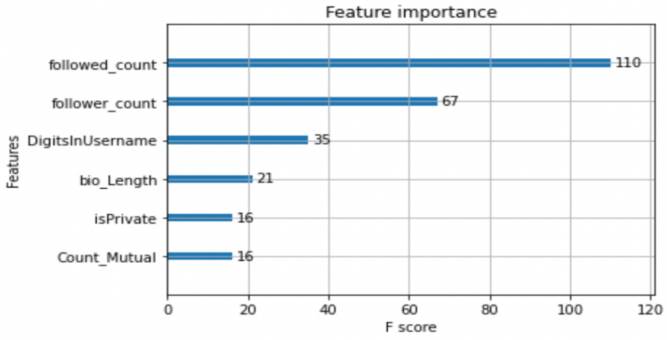


Figure 8: F-score vs Features - Feature Importance

The Figure shows the ROC curve of the model and which is 0.98.

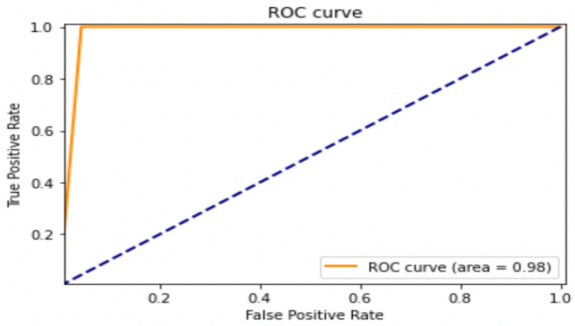
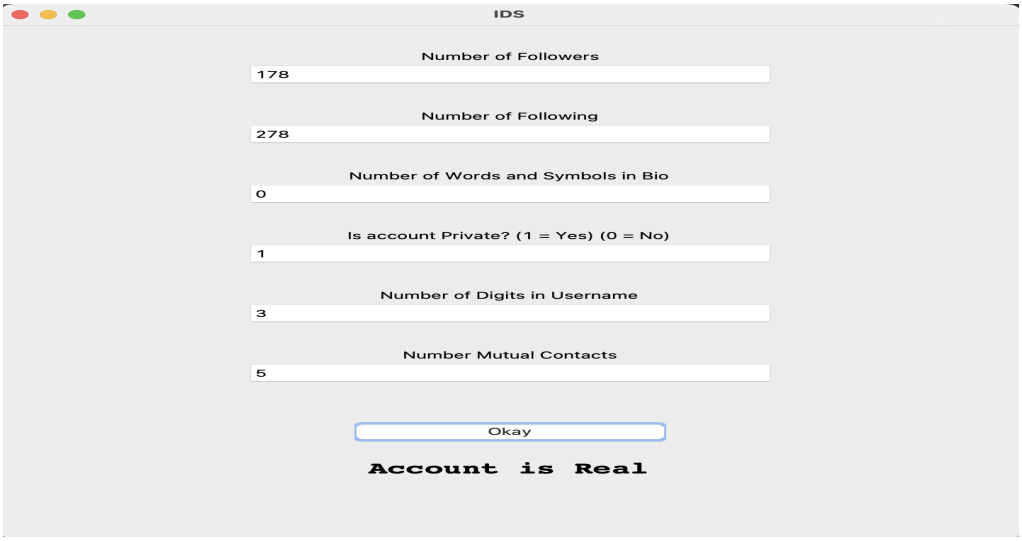


Figure 9: ROC Curve

Working of the application:

Figure 10 a and Figure 10 b represent the classification for 2 accounts, using the python application. One of which is a real account and the other is a fake account [3].



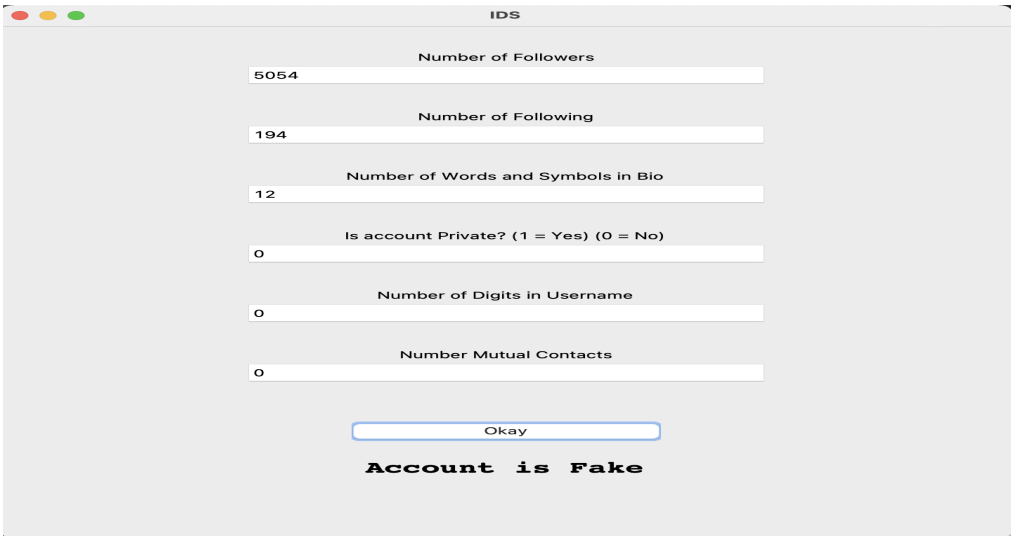


Figure (10 a) and (10 b) : Account Detection

References:

[1] F. C. Akyon and M. Esat Kalfaoglu, "Instagram Fake and Automated Account Detection," *2019 Innovations in Intelligent Systems and Applications Conference (ASYU)*, 2019, pp. 1-7, doi: 10.1109/ASYU48272.2019.8946437.

[2]A. D. Frunze and A. A. Frolov, "Methods for Detecting Fake Accounts on the Social Network VK," *2021 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (ElConRus)*, 2021, pp. 342-346, doi: 10.1109/ElConRus51938.2021.9396670.

[3]https://www.cnbc.com/amp/2021/12/14/instagram-accounts-created-with-stolen-pics-push-bogus-crypto-schemes.html

[4]https://towardsdatascience.com/understanding-gradient-boosting-machines-9be756fe76ab

[5] https://www.mygreatlearning.com/blog/gradient-boosting/

[6]https://machinelearningmastery.com/gentle-introduction-gradient-boosting-algorithm-machine-learning/