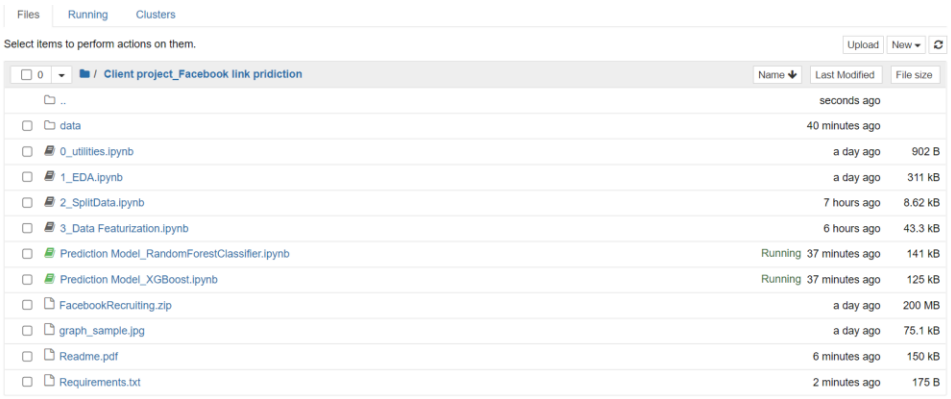
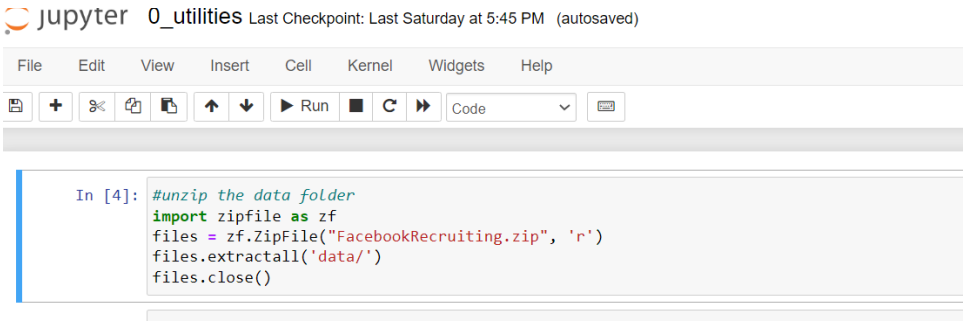
Facebook Recruiting Challenge: Working Project

1: Folder/files in the project



2: Unzip the data file in code



3: Exploratory data Analysis

Data overview

Explore data

```
In [5]: train_data = pd.read_csv('data/train.csv')
        print("----Original Data----")
        train_data.head(5)
```

```
----Original Data----
```

Out[5]:

	source_node	destination_node
0	1	690569
1	1	315892
2	1	189226
3	2	834328
4	2	1615927

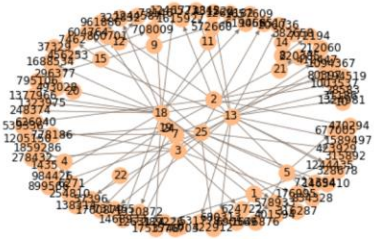
```
In [9]: #read graph data using built-in python network x module

        if os.path.isfile('data/new_train.csv'):
            graph = nx.read_edgelist('data/new_train.csv',delimiter=',',create_using=nx.DiGraph(),nodetype=int)
            print(nx.info(graph))
        else:
            train_data = pd.read_csv('data/train.csv')
```

Sample graph plot

Display a sample of data as graph

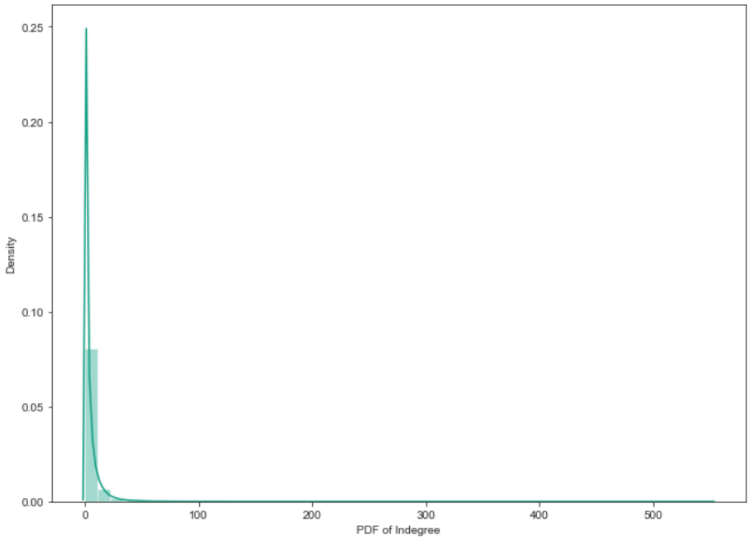
In [18]:
```
if not os.path.isfile('data/new_train_sample.csv'):
    pd.read_csv('data/train.csv', nrows=80).to_csv('data/new_train_sample.csv',header=False,index=False)

sub_graph=nx.read_edgelist('data/new_train_sample.csv',delimiter=',',create_using=nx.DiGraph(),nodetype=int)

pos=nx.spring_layout(sub_graph)
nx.draw(sub_graph,pos,node_color='#ffba85',edge_color='#857162',width=0.8,edge_cmap=plt.cm.Oranges,with_labels=True)
plt.savefig("graph_sample.jpg")
print(nx.info(sub_graph))
```

```
DiGraph with 102 nodes and 80 edges
```



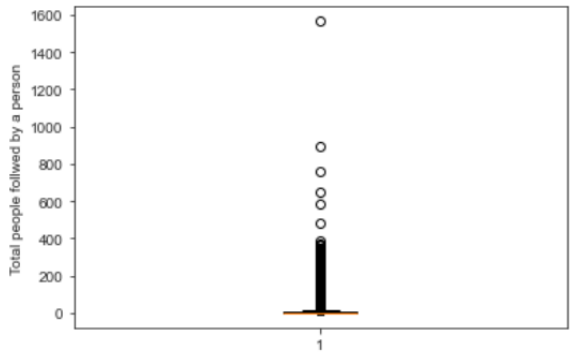
Indegree distribution

```
Out[45]: Text(0.5, 0, 'PDF of Indegree')
```



Boxplot of the distribution

```
plt.ylabel('Total people follwed by a person')
plt.show()
```



[48]:
```
### 90-100 percentile
for i in range(0,21):
    print(80+i,'percentile value is',np.percentile(person_outdegree,80+i))
```

```
80 percentile value is 7.0
81 percentile value is 7.0
82 percentile value is 8.0
83 percentile value is 8.0
84 percentile value is 9.0
85 percentile value is 9.0
86 percentile value is 10.0
```

```
99.3 percentile value is 93.0
99.4 percentile value is 99.0
99.5 percentile value is 108.0
99.6 percentile value is 120.0
99.7 percentile value is 138.0
99.8 percentile value is 168.0
99.9 percentile value is 221.0
100.0 percentile value is 1579.0
```

[72]:
```
#print('Minimium ', in_out_degree.min(), 'people and Maximum',in_out_degree.max(), 'people having followers + following')
print("No of people having following & followers:")
print("Minimum: ",np.sum(in_out_degree==in_out_degree.min()),"\t Maximum: ",np.sum(in_out_degree==in_out_degree.max()),'\n')

print(np.sum(in_out_degree<10),' have less than 10 followers & following')
print(len(list(nx.weakly_connected_components(graph))),' are weakly connected components',)

count=0
for i in list(nx.weakly_connected_components(graph)):
    if len(i)==2:
        count+=1
print('weakly connected components wit 2 nodes',count)
```

```
No of people having following & followers:
Minimum:  334291         Maximum:  1

1320326  have less than 10 followers & following
45558  are weakly connected components
weakly connected components wit 2 nodes 32195
```

3: Generate bad links and Split data for test and train

```
train_graph=nx.read_edgelist('data/train_orig.csv',delimiter=',',create_using=nx.DiGraph(),nodetype=int)
test_graph=nx.read_edgelist('data/test_orig.csv',delimiter=',',create_using=nx.DiGraph(),nodetype=int)
print(nx.info(train_graph))
print(nx.info(test_graph))

# get unique nodes in test/train graphs
train_nodes = set(train_graph.nodes(),'\n')
test_nodes = set(test_graph.nodes(),'\n')

com_people = len(train_nodes.intersection(test_nodes))
only_train_people = len(train_nodes - test_nodes)
only_test_people = len(test_nodes - train_nodes)

print(com_people,'\n people common in train & test -- ')
print(only_train_people,' people only present in train')

print(only_test_people,' people present only in test')
print('People in Test are {} %'.format(only_test_people/len(test_nodes)*100))
```

```
Number of nodes in the graph with edges 9437519
Number of nodes in the graph without edges 9437519
=====================================================
Number of nodes in the train data graph with edges 7550015 = 7550015
Number of nodes in the train data graph without edges 7550015 = 7550015
=====================================================
Number of nodes in the test data graph with edges 1887504 = 1887504
Number of nodes in the test data graph without edges 1887504 = 1887504
```

In [10]:
```
#final train and test data sets
if (not os.path.isfile('data/train_x.csv')) and (not os.path.isfile('data/test_x.csv')) and\
(not os.path.isfile('data/train_y.csv')) and (not os.path.isfile('data/test_y.csv')):

    X_train = x_train_orig.append(x_train_gener,ignore_index=True)
    y_train = np.concatenate((y_train_orig,y_train_gener))
    X_test = x_test_orig.append(x_test_gener,ignore_index=True)
    y_test = np.concatenate((y_test_orig,y_test_gener))

    print("Total data points in training data",X_train.shape)
    print("Total data points in testing data",X_test.shape)
    print("Shape of traget variable in train",y_train.shape)
    print("Shape of traget variable in test", y_test.shape)

    X_train.to_csv('data/train_x.csv',header=False,index=False)
    X_test.to_csv('data/test_x.csv',header=False,index=False)
    pd.DataFrame(y_train.astype(int)).to_csv('data/train_y.csv',header=False,index=False)
    pd.DataFrame(y_test.astype(int)).to_csv('data/test_y.csv',header=False,index=False)
```

```
Total data points in training data (15100030, 2)
Total data points in testing data (3775008, 2)
Shape of traget variable in train (15100030,)
Shape of traget variable in test (3775008,)
```

4: Data Featurization:

Calculate similarities using jaquard and cosine:

In [2]:
```
if os.path.isfile('data/train_orig.csv'):
    train_graph=nx.read_edgelist('data/train_orig.csv',delimiter=',',create_using=nx.DiGraph(),nodetype=int)
    print(nx.info(train_graph))
else:
    print("Inorder to run this project, please follow the bello steps")
    print('Step 1: Run EDA.ipynb, atleast the first 3 cells if not full')
    print('Step 2: Run full SplitData.ipynb or search the data folder for the required files')
```

```
DiGraph with 1780938 nodes and 7550015 edges
```

Similarty measure used: Jaqard and Cosine

$$Jaquard\,Distance = \frac{|X \cap Y|}{|X \cup Y|}$$

In [3]:
```
#helper function to calculate jaqard distance for followees
def followees_jaqScore(a,b):
    try:
        if len(set(train_graph.successors(a))) == 0  | len(set(train_graph.successors(b))) == 0:
            return 0
        sim = (len(set(train_graph.successors(a)).intersection(set(train_graph.successors(b)))))/\
                                    (len(set(train_graph.successors(a)).union(set(train_graph.successors(b)))))
    except:
        return 0
    return sim

#example test case
print(followees_jaqScore(273084,1505602))
```

Measure Page rank, adar Index and katz score

Ranking Measures: Page Rank

In [9]:
```
#Calculate page rank
if not os.path.isfile('data/page_rank.p'):
    pr = nx.pagerank(train_graph, alpha=0.85)
    pk.dump(pr,open('data/page_rank.p','wb'))
else:
    pr = pk.load(open('data/page_rank.p','rb'))

print('min',pr[min(pr, key=pr.get)])
print('max',pr[max(pr, key=pr.get)])
print('mean',float(sum(pr.values())) / len(pr))

#for inputing to nodes which are not there in Train data
mean_pr = float(sum(pr.values())) / len(pr)
print(mean_pr)
```

```
min 1.657890853286788e-07
max 2.799140908965827e-05
mean 5.61501860254627e-07
5.61501860254627e-07
```

In [10]:
```
# helper function to compute shortest path:
def compute_shortestPath(a,b):
    p=-1
    try:
        if train_graph.has_edge(a,b):
            train_graph.remove_edge(a,b)
```

Calculate weight features

In [23]:
```
# Weight Features for source and destination of each link
Weight_in = {}
Weight_out = {}
for i in  tqdm(train_graph.nodes()):
    s1=set(train_graph.predecessors(i))
    w_in = 1.0/(np.sqrt(1+len(s1)))
    Weight_in[i]=w_in

    s2=set(train_graph.successors(i))
    w_out = 1.0/(np.sqrt(1+len(s2)))
    Weight_out[i]=w_out

#for imputing with mean
mean_weight_in = np.mean(list(Weight_in.values()))
mean_weight_out = np.mean(list(Weight_out.values()))
```

```
100%|██████████| 1780938/1780938 [00:09<00:00, 191643.37it/s]
```

In [24]:
```
# Adding new set of features
if not os.path.isfile('data/storage_sample_stage3.h5'):
    #mapping to pandas train
    df_final_train['weight_in'] = df_final_train.destination_node.apply(lambda x: Weight_in.get(x,mean_weight_in))
    df_final_train['weight_out'] = df_final_train.source_node.apply(lambda x: Weight_out.get(x,mean_weight_out))

    #mapping to pandas test
    df_final_test['weight_in'] = df_final_test.destination_node.apply(lambda x: Weight_in.get(x,mean_weight_in))
    df_final_test['weight_out'] = df_final_test.source_node.apply(lambda x: Weight_out.get(x,mean_weight_out))
```

Calculate SVD

[25]:
```
# Adding new set of features
#for svd features to get feature vector creating a dict nde val and inedx in svd vector
sadj_col = sorted(train_graph.nodes())
sadj_dict = { val:idx for idx,val in enumerate(sadj_col)}

Adj = nx.adjacency_matrix(train_graph,nodelist=sorted(train_graph.nodes())).asfptype()

U, s, V = svds(Adj, k = 6)
print('Adjacency matrix Shape',Adj.shape)
print('U Shape',U.shape)
print('V Shape',V.shape)
print('s Shape',s.shape)
```

```
Adjacency matrix Shape (1780938, 1780938)
U Shape (1780938, 6)
V Shape (6, 1780938)
s Shape (6,)
```

Total features to train model on

```
from sklearn.metrics import f1_score
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import RandomForestClassifier as rfc
```

In [2]:
```
#read the stored features
df_final_train = pd.read_hdf('data/storage_sample_stage4.h5', 'train_df',mode='r')
df_final_test = pd.read_hdf('data/storage_sample_stage4.h5', 'test_df',mode='r')
df_final_train.columns
```

Out[2]:
```
Index(['source_node', 'destination_node', 'indicator_link',
       'jaccard_followers', 'jaccard_followees', 'cosine_followers',
       'cosine_followees', 'num_followers_s', 'num_followers_d',
       'num_followees_s', 'num_followees_d', 'inter_followers',
       'inter_followees', 'adar_index', 'follows_back', 'same_comp',
       'shortest_path', 'weight_in', 'weight_out', 'weight_f1', 'weight_f2',
       'weight_f3', 'weight_f4', 'page_rank_s', 'page_rank_d', 'katz_s',
       'katz_d', 'hubs_s', 'hubs_d', 'authorities_s', 'authorities_d',
       'svd_u_s_1', 'svd_u_s_2', 'svd_u_s_3', 'svd_u_s_4', 'svd_u_s_5',
       'svd_u_s_6', 'svd_u_d_1', 'svd_u_d_2', 'svd_u_d_3', 'svd_u_d_4',
       'svd_u_d_5', 'svd_u_d_6', 'svd_v_s_1', 'svd_v_s_2', 'svd_v_s_3',
       'svd_v_s_4', 'svd_v_s_5', 'svd_v_s_6', 'svd_v_d_1', 'svd_v_d_2',
       'svd_v_d_3', 'svd_v_d_4', 'svd_v_d_5', 'svd_v_d_6'],
      dtype='object')
```

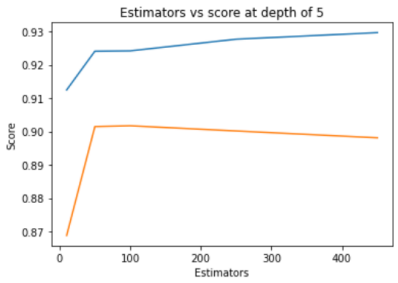
4: Random Forest Classifier Model

Estimators plot

```
plt.plot(estimators,train_scores,label='Train Score')
plt.plot(estimators,test_scores,label='Test Score')
plt.xlabel('Estimators')
plt.ylabel('Score')
plt.title('Estimators vs score at depth of 5')
```

```
Estimators =  10 Train Score 0.9124631578947369 test Score 0.868819176108455
Estimators =  50 Train Score 0.9240865404970575 test Score 0.9014697642343004
Estimators =  100 Train Score 0.9241865075180835 test Score 0.9017164708156996
Estimators =  250 Train Score 0.9276877357894071 test Score 0.9001573852489058
Estimators =  450 Train Score 0.9296561317722439 test Score 0.8981110846658157
```

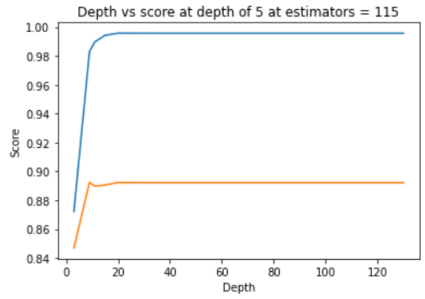
Out[4]: Text(0.5, 1.0, 'Estimators vs score at depth of 5')



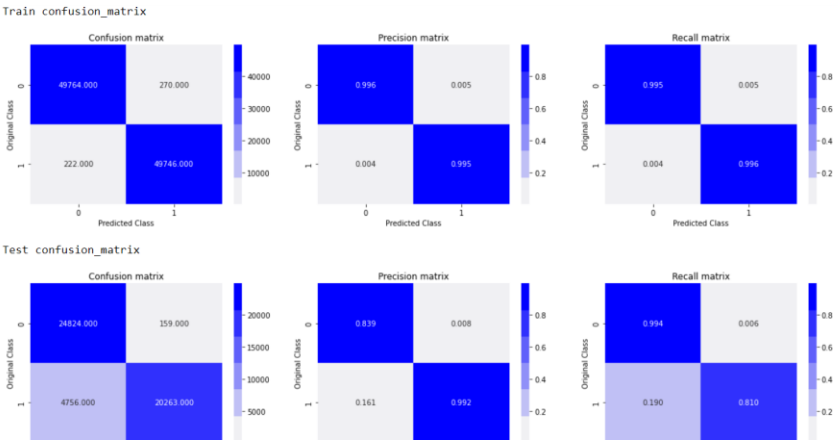
Depths plot

```
plt.ylabel('Score')
plt.title('Depth vs score at depth of 5 at estimators = 115')
plt.show()
```

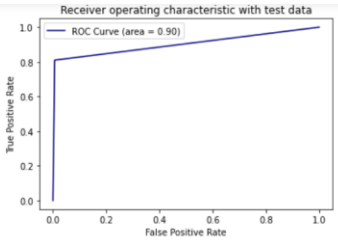
```
depth =  3 Train Score 0.8722415049063718 test Score 0.8468952855550455
depth =  9 Train Score 0.9829038175055991 test Score 0.8923429175289507
depth =  11 Train Score 0.9896575314902009 test Score 0.8897611749399079
depth =  15 Train Score 0.994314769588021 test Score 0.8905434279798224
depth =  20 Train Score 0.9957826990725935 test Score 0.892275886337644
depth =  35 Train Score 0.995732688406304 test Score 0.8921973961998592
depth =  50 Train Score 0.995732688406304 test Score 0.8921973961998592
depth =  70 Train Score 0.995732688406304 test Score 0.8921973961998592
depth =  130 Train Score 0.995732688406304 test Score 0.8921973961998592
```



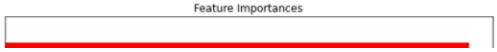
Confusion matrices for Random Forest performance



AUC curve and Feature importance



In [12]:
```
features = df_final_train.columns
importances = random_forest_cls.feature_importances_
indices = (np.argsort(importances))[-25:]
plt.figure(figsize=(10,12))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='r', align='center')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```



F1 scores of Random Forest classifier and saving the model for further use

```
In [18]: tab = PrettyTable()
         tab.field_names=["Model","Train_f1_Score","Test_f1_score"]
         tab.add_row(["RandomForest","0.9950792126740279","0.891837767654761"])
         print(tab)
```

```
+--------------+--------------------+-------------------+
|    Model     |   Train_f1_Score   |   Test_f1_score   |
+--------------+--------------------+-------------------+
| RandomForest | 0.9950792126740279 | 0.891837767654761 |
+--------------+--------------------+-------------------+
```

```
In [17]: #save the trained model
         pk.dump(random_forest_cls, open('data/random_forest_model.pkl', 'wb'))
```

```
In [ ]: # load the model from disk
        loaded_model = pk.load(open('data/random_forest_model.pkl', 'rb'))
        result = loaded_model.score(X_test, Y_test)
```

5: Model XGBoost

Training Process of XGBoost

```
In [18]: print('mean test scores',Rsearch2.cv_results_['mean_test_score'])
         #print('mean train scores',Rsearch2.cv_results_['mean_train_score'])
```

```
mean test scores [0.99988006 0.99427445 0.99416653 0.99964017 0.99678214 0.99433421
 0.99965015 0.99986006 0.98889135 0.99986007]
```

```
In [19]: print(Rsearch2.best_estimator_)
         print(Rsearch2.best_params_)
         print(Rsearch2.best_score_)
```

```
XGBRFClassifier(base_score=0.5, booster='gbtree', callbacks=None,
                colsample_bylevel=1, colsample_bytree=0.7,
                early_stopping_rounds=None, enable_categorical=False,
                eval_metric='auc', gamma=0, gpu_id=-1, grow_policy='depthwise',
                importance_type=None, interaction_constraints='', max_bin=256,
                max_cat_to_onehot=4, max_delta_step=0, max_depth=14,
                max_leaves=0, min_child_weight=1, missing=nan,
                monotone_constraints='()', n_estimators=105, n_jobs=0,
                num_parallel_tree=105, objective='binary:logistic',
                predictor='auto', random_state=123, reg_alpha=0,
                sampling_method='uniform', scale_pos_weight=1, ...)
{'subsample': 0.85, 'colsample_bytree': 0.7}
0.9998800599670196
```

```
In [20]: col=Rsearch2.best_params_["colsample_bytree"]
         sub=Rsearch2.best_params_['subsample']
```

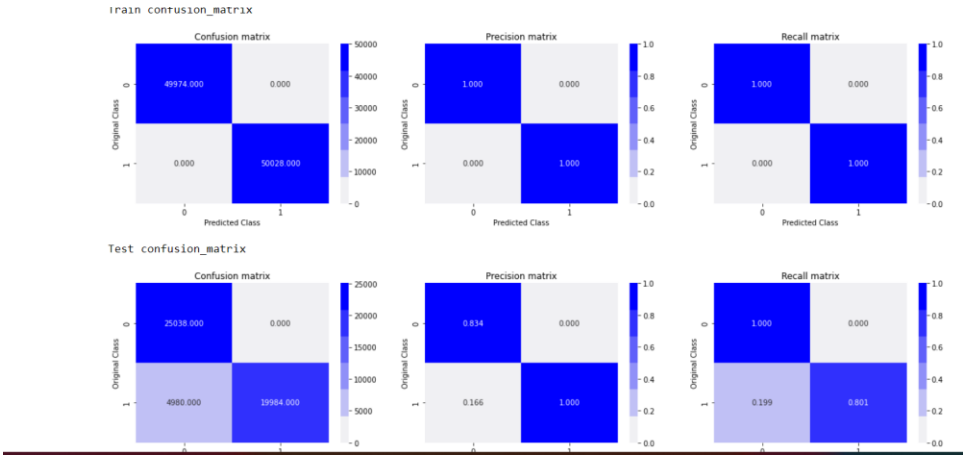
Trained XGBoost model

```
In [33]: alpha = Rsearch4.best_params_["reg_alpha"]
         #Lamda = Rsearch4.best_params_['reg_lamda']
```

```
In [37]: final_clf = xgb.XGBRFClassifier(base_score=0.5, colsample_bylevel=1, colsample_bynode=0.8,
                colsample_bytree=0.7, eval_metric='auc', gamma=0.2,
                learning_rate=0.5, max_delta_step=0, max_depth=14,
                min_child_weight=1,missing=1, n_estimators=111, n_jobs=1, nthread=None,
                objective='binary:logistic', random_state=123, reg_alpha=0.001, scale_pos_weight=1, seed=None,
                silent=None, subsample=0.9, verbosity=1)

         final_clf.fit(df_final_train,y_train)
```

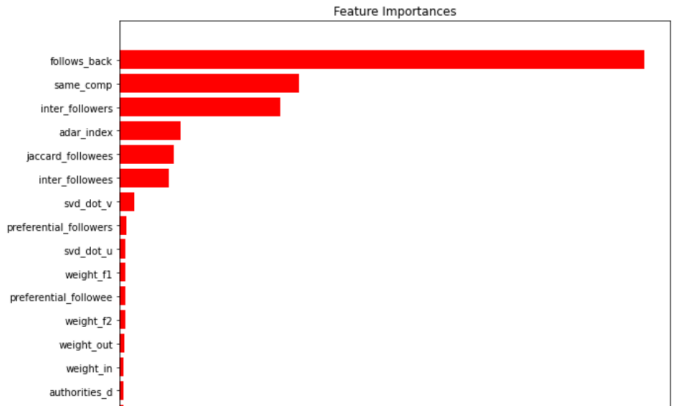
```
Out[37]: XGBRFClassifier
                colsample_bylevel=1, colsample_bytree=0.7,
                early_stopping_rounds=None, enable_categorical=False,
                eval_metric='auc', gamma=0.2, gpu_id=-1,
                grow_policy='depthwise', importance_type=None,
                interaction_constraints='', learning_rate=0.5, max_bin=256,
                max_cat_to_onehot=4, max_delta_step=0, max_depth=14,
                max_leaves=0, min_child_weight=1, missing=1,
                monotone_constraints='()', n_estimators=111, n_jobs=1,
                nthread=1, num_parallel_tree=111, objective='binary:logistic',
                predictor='auto', random_state=123, reg_alpha=0.001, ...)
```

Confusion matrices for XGBoost performance



Feature Importance rate as per XGBoost

```
plt.figure(figsize=(10,12))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='r', align='center')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```



F1 score obtained by XGBOOST and saving the classifier for further use

In [50]:
```
tab = PrettyTable()
tab.field_names=["Model","Train_f1_Score","Test_f1_score"]
tab.add_row(["XGBoost","1.0","0.8892053039067367"])
print(tab)
```

```
+---------+----------------+--------------------+
|  Model  | Train_f1_Score |   Test_f1_score    |
+---------+----------------+--------------------+
| XGBoost |      1.0       | 0.8892053039067367 |
+---------+----------------+--------------------+
```

In [49]:
```
#save the model for next use
# save to JSON
final_clf.save_model("data/xgb_model.json")
# save to text format
final_clf.save_model("data/xgb_model.txt")
```

In [ ]:
```
final_clf = xgb.Booster()
final_clf.load_model("data/xgb_model.json")
```