Social network Graph Link Prediction - Facebook Challenge

```
import warnings
warnings.filterwarnings("ignore")
import csv
import pandas as pd #pandas to create small dataframes
import datetime #Convert to unix time
import time #Convert to unix time
import numpy as np#Do aritmetic operations on arrays
import matplotlib
import matplotlib.pylab as plt
import seaborn as sns
from matplotlib import rcParams #Size of plots
from sklearn.cluster import MiniBatchKMeans, KMeans
import math
import pickle
import os
import xgboost as xgb
import networkx as nx
import pdb
import pickle
from pandas import HDFStore,DataFrame
from pandas import read_hdf
from scipy.sparse.linalg import svds, eigs
import gc
from tqdm import tqdm
from sklearn.ensemble import RandomForestClassifier
```

Saved successfully!

1. Reading Data

```
startTime = datetime.datetime.now()
print("Current Time = ",startTime)

if os.path.isfile('data/after_eda/train_pos_after_eda.csv'):
    train_graph=nx.read_edgelist('data/after_eda/train_pos_after_eda.csv',delimiter=',',cr
    print(nx.info(train_graph))
else:
    print("please run the FB_EDA.ipynb or download the files from drive")

print("\nTime taken for creation of dataframe is {}".format(datetime.datetime.now() - star
```

```
Current Time =  2019-05-24 21:09:06.868196
Name:
Type: DiGraph
Number of nodes: 1780722
Number of edges: 7550015
Average in degree:   4.2399
Average out degree:   4.2399
```

```
Time taken for creation of dataframe is 0:04:00.803636
```

2. Similarity measures

2.1 Jaccard Distance:

http://www.statisticshowto.com/jaccard-index/

$$j = \frac{|X \cap Y|}{|X \cup Y|}$$

```
#for followees
def jaccard_for_followees(a,b):
    try:
        if len(set(train_graph.successors(a))) == 0  | len(set(train_graph.successors(b)))
            return 0
        sim = (len(set(train_graph.successors(a)).intersection(set(train_graph.successors(
                                    (len(set(train_graph.successors(a)).union(set(train_gr
    except:
        return 0
    return sim


#one test case
print(jaccard_for_followees(273084,1505602))
```

Saved successfully!

```
#node 1635354 not in graph
print(jaccard_for_followees(273084,1505602))
```

```
0.0
```

```
#for followers
def jaccard_for_followers(a,b):
    try:
        if len(set(train_graph.predecessors(a))) == 0  | len(set(g.predecessors(b))) == 0:
            return 0
        sim = (len(set(train_graph.predecessors(a)).intersection(set(train_graph.predecess
                                (len(set(train_graph.predecessors(a)).union(set(train_gra
        return sim
    except:
        return 0


print(jaccard_for_followers(273084,470294))
```

```
0
```

```
#node 1635354 not in graph
print(jaccard_for_followees(669354,1635354))
```

```
0
```

2.2 Cosine distance

$$CosineDistance = \frac{|X \cap Y|}{|X| \cdot |Y|}$$

```
#for followees
def cosine_for_followees(a,b):
    try:
        if len(set(train_graph.successors(a))) == 0  | len(set(train_graph.successors(b)))
            return 0
        sim = (len(set(train_graph.successors(a)).intersection(set(train_graph.successors(
                                    (math.sqrt(len(set(train_graph.successors(a)))*len((se
        return sim
    except:
        return 0


print(cosine_for_followees(273084,1505602))
```

```
0.0
```



```
35354))
```

```
0
```

```
def cosine_for_followers(a,b):
    try:

        if len(set(train_graph.predecessors(a))) == 0  | len(set(train_graph.predecessors(
            return 0
        sim = (len(set(train_graph.predecessors(a)).intersection(set(train_graph.predecess
                                     (math.sqrt(len(set(train_graph.predecessors(a))))*(le
        return sim
    except:
        return 0


print(cosine_for_followers(2,470294))
```

```
0.02886751345948129
```

```
print(cosine_for_followers(669354,1635354))
```

```
0
```

3. Ranking Measures

https://networkx.github.io/documentation/networkx-1.10/reference/generated/networkx.algorithms.link_analysis.pagerank_alg.pagerank.html

PageRank computes a ranking of the nodes in the graph G based on the structure of the incoming links.

Mathematical PageRanks for a simple network, expressed as percentages. (Google uses a logarithmic scale.) Page C has a higher PageRank than Page E, even though there are fewer links to C; the one link to C comes from an important page and hence is of high value. If web surfers who start on a random page have an 85% likelihood of choosing a random link from the page they are currently visiting, and a 15% likelihood of jumping to a page chosen at random from the entire web, they will reach Page E 8.1% of the time. **(The 15% likelihood of jumping to an arbitrary page corresponds to a damping factor of 85%.) Without damping, all web surfers would eventually end up on Pages A, B, or C, and all other pages would have PageRank zero. In the presence of damping, Page A effectively links to all pages in the web, even though it has no outgoing links of its own.**

3.1 Page Ranking

Saved successfully!

k

```
if not os.path.isfile('data/fea_sample/page_rank.p'):
    pr = nx.pagerank(train_graph, alpha=0.85)
    pickle.dump(pr,open('data/fea_sample/page_rank.p','wb'))
else:
    pr = pickle.load(open('data/fea_sample/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))
```

```
min 1.6556497245737814e-07
max 2.7098251341935827e-05
mean 5.615699699389075e-07
```

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

```
5.615699699389075e-07
```

4. Other Graph Features

4.1 Shortest path:

Getting Shortest path between twoo nodes, if nodes have direct path i.e directly connected then we are removing that edge and calculating path.

```
#if has direct edge then deleting that edge and calculating shortest path
def compute_shortest_path_length(a,b):
    p=-1
    try:
        if train_graph.has_edge(a,b):
            train_graph.remove_edge(a,b)
            p= nx.shortest_path_length(train_graph,source=a,target=b)
            train_graph.add_edge(a,b)
        else:
            p= nx.shortest_path_length(train_graph,source=a,target=b)
        return p
    except:
        return -1


#testing
compute_shortest_path_length(77697,826021)
```

Saved successfully!

```
#testing
compute_shortest_path_length(669354,1635354)
```

```
-1
```

4.2 Checking for same community

```
#getting weekly connected edges from graph
wcc=list(nx.weakly_connected_components(train_graph))
def belongs_to_same_wcc(a,b):
    index = []
    if train_graph.has_edge(b,a):
        return 1
    if train_graph.has_edge(a,b):
            for i in wcc:
                if a in i:
                    index= i
                    break
```

```
            if (b in index):
                train_graph.remove_edge(a,b)
                if compute_shortest_path_length(a,b)==-1:
                    train_graph.add_edge(a,b)
                    return 0
                else:
                    train_graph.add_edge(a,b)
                    return 1
            else:
                return 0
    else:
            for i in wcc:
                if a in i:
                    index= i
                    break
            if(b in index):
                return 1
            else:
                return 0
```

```
belongs_to_same_wcc(861, 1659750)
```

```
0
```

```
belongs_to_same_wcc(669354,1635354)
```

```
0
```

Saved successfully!

Adamic/Adar measures is defined as inverted sum of degrees of common neighbours for given two vertices.

$$A(x,y) = \sum_{u \in N(x) \cap N(y)} \frac{1}{log(|N(u)|)}$$

```
#adar index
def calc_adar_in(a,b):
    sum=0
    try:
        n=list(set(train_graph.successors(a)).intersection(set(train_graph.successors(b))))
        if len(n)!=0:
            for i in n:
                sum=sum+(1/np.log10(len(list(train_graph.predecessors(i)))))
            return sum
        else:
            return 0
    except:
        return 0
```

```
calc_adar_in(1,189226)
```

```
0
```

```
calc_adar_in(669354,1635354)
```

```
0
```

4.4 Is persion was following back:

```
def follows_back(a,b):
    if train_graph.has_edge(b,a):
        return 1
    else:
        return 0
```

```
follows_back(1,189226)
```

```
1
```

```
follows_back(669354,1635354)
```

```
0
```

4.5 Katz Centrality:

Saved successfully!

trality

https://www.geeksforgeeks.org/katz-centrality-centrality-measure/ Katz centrality computes the centrality for a node based on the centrality of its neighbors. It is a generalization of the eigenvector centrality. The Katz centrality for node `i` is

$$x_i = \alpha \sum_j A_{ij} x_j + \beta,$$

where `A` is the adjacency matrix of the graph G with eigenvalues

$$\lambda$$

.

The parameter

$$\beta$$

controls the initial centrality and

$$\alpha < \frac{1}{\lambda_{max}}.$$

```
if not os.path.isfile('data/fea_sample/katz.p'):
    katz = nx.katz.katz_centrality(train_graph,alpha=0.005,beta=1)
```

```
    pickle.dump(katz,open('data/fea_sample/katz.p','wb'))
else:
    katz = pickle.load(open('data/fea_sample/katz.p','rb'))


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

```
min 0.0007313532484065916
max 0.003394554981699122
mean 0.0007483800935562018
```

```
mean_katz = float(sum(katz.values())) / len(katz)
print(mean_katz)
```

```
0.0007483800935562018
```

4.6 Hits Score

The HITS algorithm computes two numbers for a node. Authorities estimates the node value based on the incoming links. Hubs estimates the node value based on outgoing links.

https://en.wikipedia.org/wiki/HITS_algorithm

[ ] *2 cells hidden*

Saved successfully!

5. 1 Reading a sample of Data from both train and test

```
import random
if os.path.isfile('data/after_eda/train_after_eda.csv'):
    filename = "data/after_eda/train_after_eda.csv"
    # you uncomment this line, if you dont know the lentgh of the file name
    # here we have hardcoded the number of lines as 15100030
    # n_train = sum(1 for line in open(filename)) #number of records in file (excludes hea
    n_train =  15100028
    s = 100000 #desired sample size
    skip_train = sorted(random.sample(range(1,n_train+1),n_train-s))
    #https://stackoverflow.com/a/22259008/4084039


if os.path.isfile('data/after_eda/train_after_eda.csv'):
    filename = "data/after_eda/test_after_eda.csv"
    # you uncomment this line, if you dont know the lentgh of the file name
    # here we have hardcoded the number of lines as 3775008
    # n_test = sum(1 for line in open(filename)) #number of records in file (excludes head
```

```
    n_test = 3775006
    s = 50000 #desired sample size
    skip_test = sorted(random.sample(range(1,n_test+1),n_test-s))
    #https://stackoverflow.com/a/22259008/4084039
```

```
print("Number of rows in the train data file:", n_train)
print("Number of rows we are going to elimiate in train data are",len(skip_train))
print("Number of rows in the test data file:", n_test)
print("Number of rows we are going to elimiate in test data are",len(skip_test))
```

```
    Number of rows in the train data file: 15100028
    Number of rows we are going to elimiate in train data are 15000028
    Number of rows in the test data file: 3775006
    Number of rows we are going to elimiate in test data are 3725006
```

```
df_final_train = pd.read_csv('data/after_eda/train_after_eda.csv', skiprows=skip_train, na
df_final_train['indicator_link'] = pd.read_csv('data/train_y.csv', skiprows=skip_train, na
print("Our train matrix size ",df_final_train.shape)
df_final_train.head(2)
```

```
    Our train matrix size  (100002, 3)
```

	source_node	destination_node	indicator_link
0	273084	1505602	1
1	832016	1543415	1

```
df_final_test = pd.read_csv('data/after_eda/test_after_eda.csv', skiprows=skip_test, names
                .read_csv('data/test_y.csv', skiprows=skip_test, names
                al_test.shape)
df_final_test.head(2)
```

Saved successfully!

```
    Our test matrix size  (50002, 3)
```

	source_node	destination_node	indicator_link
0	848424	784690	1
1	483294	1255532	1

5.2 Adding a set of features

we will create these each of these features for both train and test data points

1. jaccard_followers
2. jaccard_followees
3. cosine_followers
4. cosine_followees
5. num_followers_s
6. num_followees_s

7. num_followers_d
8. num_followees_d
9. inter_followers
10. inter_followees

```
if not os.path.isfile('data/fea_sample/storage_sample_stage1.h5'):
    #mapping jaccard followers to train and test data
    df_final_train['jaccard_followers'] = df_final_train.apply(lambda row:
                                            jaccard_for_followers(row['source_node'],row['
    df_final_test['jaccard_followers'] = df_final_test.apply(lambda row:
                                            jaccard_for_followers(row['source_node'],row['

    #mapping jaccard followees to train and test data
    df_final_train['jaccard_followees'] = df_final_train.apply(lambda row:
                                            jaccard_for_followees(row['source_node'],row['
    df_final_test['jaccard_followees'] = df_final_test.apply(lambda row:
                                            jaccard_for_followees(row['source_node'],row['


    #mapping jaccard followers to train and test data
    df_final_train['cosine_followers'] = df_final_train.apply(lambda row:
                                            cosine_for_followers(row['source_node'],row['d
    df_final_test['cosine_followers'] = df_final_test.apply(lambda row:
                                            cosine_for_followers(row['source_node'],row['d

    #mapping jaccard followees to train and test data
    df_final_train['cosine_followees'] = df_final_train.apply(lambda row:
                                            cosine_for_followees(row['source_node'],row['d
                                        ] = df_final_test.apply(lambda row:
                                            cosine_for_followees(row['source_node'],row['d
```

Saved successfully!

```
def compute_features_stage1(df_final):
    #calculating no of followers followees for source and destination
    #calculating intersection of followers and followees for source and destination
    num_followers_s=[]
    num_followees_s=[]
    num_followers_d=[]
    num_followees_d=[]
    inter_followers=[]
    inter_followees=[]
    for i,row in df_final.iterrows():
        try:
            s1=set(train_graph.predecessors(row['source_node']))
            s2=set(train_graph.successors(row['source_node']))
        except:
            s1 = set()
            s2 = set()
        try:
            d1=set(train_graph.predecessors(row['destination_node']))
            d2=set(train_graph.successors(row['destination_node']))
        except:
            d1 = set()
```

```
            d2 = set()
        num_followers_s.append(len(s1))
        num_followees_s.append(len(s2))

        num_followers_d.append(len(d1))
        num_followees_d.append(len(d2))

        inter_followers.append(len(s1.intersection(d1)))
        inter_followees.append(len(s2.intersection(d2)))

    return num_followers_s, num_followers_d, num_followees_s, num_followees_d, inter_follo


if not os.path.isfile('data/fea_sample/storage_sample_stage1.h5'):
    df_final_train['num_followers_s'], df_final_train['num_followers_d'], \
    df_final_train['num_followees_s'], df_final_train['num_followees_d'], \
    df_final_train['inter_followers'], df_final_train['inter_followees']= compute_features

    df_final_test['num_followers_s'], df_final_test['num_followers_d'], \
    df_final_test['num_followees_s'], df_final_test['num_followees_d'], \
    df_final_test['inter_followers'], df_final_test['inter_followees']= compute_features_s

    hdf = HDFStore('data/fea_sample/storage_sample_stage1.h5')
    hdf.put('train_df',df_final_train, format='table', data_columns=True)
    hdf.put('test_df',df_final_test, format='table', data_columns=True)
    hdf.close()
else:
    df_final_train = read_hdf('data/fea_sample/storage_sample_stage1.h5', 'train_df',mode=
    df_final_test = read_hdf('data/fea_sample/storage_sample_stage1.h5', 'test_df',mode='r
```

Saved successfully!

ures

we will create these each of these features for both train and test data points

1. adar index
2. is following back
3. belongs to same weakly connect components
4. shortest path between source and destination

```
if not os.path.isfile('data/fea_sample/storage_sample_stage2.h5'):
    #mapping adar index on train
    df_final_train['adar_index'] = df_final_train.apply(lambda row: calc_adar_in(row['sour
    #mapping adar index on test
    df_final_test['adar_index'] = df_final_test.apply(lambda row: calc_adar_in(row['source

    #------------------------------------------------------------------------------------
    #mapping followback or not on train
    df_final_train['follows_back'] = df_final_train.apply(lambda row: follows_back(row['so

    #mapping followback or not on test
    df_final_test['follows_back'] = df_final_test.apply(lambda row: follows_back(row['sour
```

```
    #-------------------------------------------------------------------------------
    #mapping same component of wcc or not on train
    df_final_train['same_comp'] = df_final_train.apply(lambda row: belongs_to_same_wcc(row

    ##mapping same component of wcc or not on train
    df_final_test['same_comp'] = df_final_test.apply(lambda row: belongs_to_same_wcc(row['

    #-------------------------------------------------------------------------------
    #mapping shortest path on train
    df_final_train['shortest_path'] = df_final_train.apply(lambda row: compute_shortest_pa
    #mapping shortest path on test
    df_final_test['shortest_path'] = df_final_test.apply(lambda row: compute_shortest_path

    hdf = HDFStore('data/fea_sample/storage_sample_stage2.h5')
    hdf.put('train_df',df_final_train, format='table', data_columns=True)
    hdf.put('test_df',df_final_test, format='table', data_columns=True)
    hdf.close()
else:
    df_final_train = read_hdf('data/fea_sample/storage_sample_stage2.h5', 'train_df',mode=
    df_final_test = read_hdf('data/fea_sample/storage_sample_stage2.h5', 'test_df',mode='r
```

5.4 Adding new set of features

we will create these each of these features for both train and test data points

1. Weight Features
    - weight of incoming edges

Saved successfully!

    - weight of outgoing edges
    - weight of incoming edges * weight of outgoing edges
    - 2*weight of incoming edges + weight of outgoing edges
    - weight of incoming edges + 2*weight of outgoing edges
2. Page Ranking of source
3. Page Ranking of dest
4. katz of source
5. katz of dest
6. hubs of source
7. hubs of dest
8. authorities_s of source
9. authorities_s of dest

Weight Features

In order to determine the similarity of nodes, an edge weight value was calculated between nodes. Edge weight decreases as the neighbor count goes up. Intuitively, consider one million people following a celebrity on a social network then chances are most of them never met each other or the celebrity. On the other hand, if a user has 30 contacts in his/her social network, the chances are higher that many of them know each other. `credit` - Graph-based Features for Supervised Link Prediction William Cukierski, Benjamin Hamner, Bo Yang

$$W = \frac{1}{\sqrt{1+|X|}}$$

it is directed graph so calculated Weighted in and Weighted out differently

```
#weight 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()))
```

Saved successfully!

```
100%|██████████| 1780722/17
```

```
if not os.path.isfile('data/fea_sample/storage_sample_stage3.h5'):
    #mapping to pandas train
    df_final_train['weight_in'] = df_final_train.destination_node.apply(lambda x: Weight_i
    df_final_train['weight_out'] = df_final_train.source_node.apply(lambda x: Weight_out.g

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


    #some features engineerings on the in and out weights
    df_final_train['weight_f1'] = df_final_train.weight_in + df_final_train.weight_out
    df_final_train['weight_f2'] = df_final_train.weight_in * df_final_train.weight_out
    df_final_train['weight_f3'] = (2*df_final_train.weight_in + 1*df_final_train.weight_ou
    df_final_train['weight_f4'] = (1*df_final_train.weight_in + 2*df_final_train.weight_ou

    #some features engineerings on the in and out weights
    df_final_test['weight_f1'] = df_final_test.weight_in + df_final_test.weight_out
    df_final_test['weight_f2'] = df_final_test.weight_in * df_final_test.weight_out
```

```
    df_final_test['weight_f3'] = (2*df_final_test.weight_in + 1*df_final_test.weight_out)
    df_final_test['weight_f4'] = (1*df_final_test.weight_in + 2*df_final_test.weight_out)
if not os.path.isfile('data/fea_sample/storage_sample_stage3.h5'):

    #page rank for source and destination in Train and Test
    #if anything not there in train graph then adding mean page rank
    df_final_train['page_rank_s'] = df_final_train.source_node.apply(lambda x:pr.get(x,mea
    df_final_train['page_rank_d'] = df_final_train.destination_node.apply(lambda x:pr.get(

    df_final_test['page_rank_s'] = df_final_test.source_node.apply(lambda x:pr.get(x,mean_
    df_final_test['page_rank_d'] = df_final_test.destination_node.apply(lambda x:pr.get(x,
    #========================================================================

    #Katz centrality score for source and destination in Train and test
    #if anything not there in train graph then adding mean katz score
    df_final_train['katz_s'] = df_final_train.source_node.apply(lambda x: katz.get(x,mean_
    df_final_train['katz_d'] = df_final_train.destination_node.apply(lambda x: katz.get(x,

    df_final_test['katz_s'] = df_final_test.source_node.apply(lambda x: katz.get(x,mean_ka
    df_final_test['katz_d'] = df_final_test.destination_node.apply(lambda x: katz.get(x,me
    #========================================================================

    #Hits algorithm score for source and destination in Train and test
    #if anything not there in train graph then adding 0
    df_final_train['hubs_s'] = df_final_train.source_node.apply(lambda x: hits[0].get(x,0)
    df_final_train['hubs_d'] = df_final_train.destination_node.apply(lambda x: hits[0].get

    df_final_test['hubs_s'] = df_final_test.source_node.apply(lambda x: hits[0].get(x,0))
    df_final_test['hubs_d'] = df_final_test.destination_node.apply(lambda x: hits[0].get(x
    #------------------------------========================================================

    and destination in Train and Test
    #if anything not there in train graph then adding 0
    df_final_train['authorities_s'] = df_final_train.source_node.apply(lambda x: hits[1].g
    df_final_train['authorities_d'] = df_final_train.destination_node.apply(lambda x: hits

    df_final_test['authorities_s'] = df_final_test.source_node.apply(lambda x: hits[1].get
    df_final_test['authorities_d'] = df_final_test.destination_node.apply(lambda x: hits[1
    #========================================================================

    hdf = HDFStore('data/fea_sample/storage_sample_stage3.h5')
    hdf.put('train_df',df_final_train, format='table', data_columns=True)
    hdf.put('test_df',df_final_test, format='table', data_columns=True)
    hdf.close()
else:
    df_final_train = read_hdf('data/fea_sample/storage_sample_stage3.h5', 'train_df',mode=
    df_final_test = read_hdf('data/fea_sample/storage_sample_stage3.h5', 'test_df',mode='r
```

Saved successfully!

5.5 Adding new set of features

we will create these each of these features for both train and test data points

1. SVD features for both source and destination

```
def svd(x, S):
    try:
        z = sadj_dict[x]
        return S[z]
    except:
        return [0,0,0,0,0,0]


#for svd features to get feature vector creating a dict node 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 (1780722, 1780722)
U Shape (1780722, 6)
V Shape (6, 1780722)
s Shape (6,)
```

```
if not os.path.isfile('data/fea_sample/storage_sample_stage4.h5'):
    #------------------------------============================================================
    df_final_train[['svd_u_s_1','svd_u_s_2','svd_u_s_3', 'svd_u_s_4', 'svd_u_s_5', 'svd_u
    df_final_train.source_node.apply(lambda x: svd(x, U)).apply(pd.Series)

    df_final_train[['svd_u_d_1', 'svd_u_d_2', 'svd_u_d_3', 'svd_u_d_4', 'svd_u_d_5','svd_u
    df_final_train.destination_node.apply(lambda x: svd(x, U)).apply(pd.Series)
    #============================================================================================

    df_final_train[['svd_v_s_1','svd_v_s_2', 'svd_v_s_3', 'svd_v_s_4', 'svd_v_s_5', 'svd_v
    df_final_train.source_node.apply(lambda x: svd(x, V.T)).apply(pd.Series)

    df_final_train[['svd_v_d_1', 'svd_v_d_2', 'svd_v_d_3', 'svd_v_d_4', 'svd_v_d_5','svd_v
    df_final_train.destination_node.apply(lambda x: svd(x, V.T)).apply(pd.Series)
    #============================================================================================

    df_final_test[['svd_u_s_1', 'svd_u_s_2','svd_u_s_3', 'svd_u_s_4', 'svd_u_s_5', 'svd_u_
    df_final_test.source_node.apply(lambda x: svd(x, U)).apply(pd.Series)

    df_final_test[['svd_u_d_1', 'svd_u_d_2', 'svd_u_d_3', 'svd_u_d_4', 'svd_u_d_5','svd_u_
    df_final_test.destination_node.apply(lambda x: svd(x, U)).apply(pd.Series)

    #============================================================================================

    df_final_test[['svd_v_s_1','svd_v_s_2', 'svd_v_s_3', 'svd_v_s_4', 'svd_v_s_5', 'svd_v_
```

```
    df_final_test.source_node.apply(lambda x: svd(x, V.T)).apply(pd.Series)

    df_final_test[['svd_v_d_1', 'svd_v_d_2', 'svd_v_d_3', 'svd_v_d_4', 'svd_v_d_5','svd_v_
    df_final_test.destination_node.apply(lambda x: svd(x, V.T)).apply(pd.Series)
    #================================================================================

    hdf = HDFStore('data/fea_sample/storage_sample_stage4.h5')
    hdf.put('train_df',df_final_train, format='table', data_columns=True)
    hdf.put('test_df',df_final_test, format='table', data_columns=True)
    hdf.close()

# prepared and stored the data from machine learning models
# pelase check the FB_Models.ipynb


#df_final_train
df_final_train.ix[:,'weight_f1':][:10]
```

	weight_f1	weight_f2	weight_f3	weight_f4	page_rank_s	page_rank_d	katz_s	
0	0.627964	0.094491	1.005929	0.877964	2.045290e-06	3.459963e-07	0.000773	0.(
1	0.229598	0.013030	0.332196	0.356598	2.353458e-07	6.427660e-07	0.000845	0.(
2	0.339999	0.028653	0.525694	0.494302	6.211019e-07	5.179801e-07	0.000885	0.(
3	0.696923	0.117851	0.985599	1.105172	2.998153e-07	1.704245e-06	0.000739	0.(
4	0.447214	0.000000	1.341511	1.603023	4.349180e-07	2.089590e-07	0.000751	0.(
5	0.617739	0.095346	0.933967	0.919250	5.942343e-07	1.143388e-06	0.000767	0.(
6	1.447214	0.447214	1.894427	2.447214	2.848986e-07	1.128758e-06	0.000735	0.(
7	0.853553	0.176777	1.353553	1.207107	6.694862e-07	5.254600e-07	0.000763	0.(
8	0.583489	0.084515	0.850750	0.899717	1.466870e-06	1.373409e-06	0.000757	0.(
9	0.930904	0.204124	1.508254	1.284457	6.630224e-07	2.618341e-07	0.000758	0.(

10 rows × 42 columns

Saved successfully!

Assignments:

1. Add another feature called Preferential Attachment with followers and followees data of vertex. you can check about Preferential Attachment in below link

http://be.amazd.com/link-prediction/

2. Add feature called svd_dot. you can calculate svd_dot as Dot product between sourse node svd and destination node svd features. you can read about this in below pdf https://storage.googleapis.com/kaggle-forum-message-attachments/2594/supervised_link_prediction.pdf
3. Tune hyperparameters for XG boost with all these features and check the error metric.

Preferential_attachment

```
def followee_preferential_attachment(user1,user2):
    try:
        user_1 = len(set(train_graph.successors(user1)))
        user_2 = len(set(train_graph.successors(user2)))
        return(user_1*user_2)
    except:
        return(0)

def follower_preferential_attachment(user1,user2):
    try:
        user_1 = len(set(train_graph.predecessors(user1)))
        user_2 = len(set(train_graph.predecessors(user2)))
        return(user_1*user_2)
    except:
        return(0)
```

Saved successfully!

```
print("Current Time = ",startTime)

if not os.path.isfile('data/fea_sample/storage_sample_stage5.h5'):
    #================================================================================

    df_final_train[['svd_u_s_1', 'svd_u_s_2','svd_u_s_3', 'svd_u_s_4', 'svd_u_s_5', 'svd_u
    df_final_train.source_node.apply(lambda x: svd(x, U)).apply(pd.Series)

    df_final_train[['svd_u_d_1', 'svd_u_d_2', 'svd_u_d_3', 'svd_u_d_4', 'svd_u_d_5','svd_u
    df_final_train.destination_node.apply(lambda x: svd(x, U)).apply(pd.Series)
    #================================================================================

    df_final_train[['svd_v_s_1','svd_v_s_2', 'svd_v_s_3', 'svd_v_s_4', 'svd_v_s_5', 'svd_v
    df_final_train.source_node.apply(lambda x: svd(x, V.T)).apply(pd.Series)

    df_final_train[['svd_v_d_1', 'svd_v_d_2', 'svd_v_d_3', 'svd_v_d_4', 'svd_v_d_5','svd_v
    df_final_train.destination_node.apply(lambda x: svd(x, V.T)).apply(pd.Series)
    #================================================================================

    df_final_test[['svd_u_s_1', 'svd_u_s_2','svd_u_s_3', 'svd_u_s_4', 'svd_u_s_5', 'svd_u_
    df_final_test.source_node.apply(lambda x: svd(x, U)).apply(pd.Series)
```

```
    df_final_test[['svd_u_d_1', 'svd_u_d_2', 'svd_u_d_3', 'svd_u_d_4', 'svd_u_d_5','svd_u_
    df_final_test.destination_node.apply(lambda x: svd(x, U)).apply(pd.Series)

    #========================================================================================

    df_final_test[['svd_v_s_1','svd_v_s_2', 'svd_v_s_3', 'svd_v_s_4', 'svd_v_s_5', 'svd_v_
    df_final_test.source_node.apply(lambda x: svd(x, V.T)).apply(pd.Series)

    df_final_test[['svd_v_d_1', 'svd_v_d_2', 'svd_v_d_3', 'svd_v_d_4', 'svd_v_d_5','svd_v_
    df_final_test.destination_node.apply(lambda x: svd(x, V.T)).apply(pd.Series)
    #========================================================================================

    df_final_train['followee_preferential_attachment'] = df_final_train.apply(lambda row:
    df_final_test['followee_preferential_attachment'] = df_final_test.apply(lambda row: fo

    df_final_train['follower_preferential_attachment'] = df_final_train.apply(lambda row:
    df_final_test['follower_preferential_attachment'] = df_final_test.apply(lambda row: fo

    #========================================================================================

    hdf = HDFStore('data/fea_sample/storage_sample_stage5.h5')
    hdf.put('train_df',df_final_train, format='table', data_columns=True)
    hdf.put('test_df',df_final_test, format='table', data_columns=True)
    hdf.close()
else:
    df_final_train = read_hdf('data/fea_sample/storage_sample_stage5.h5', 'train_df',mode=
    df_final_test = read_hdf('data/fea_sample/storage_sample_stage5.h5', 'test_df',mode='r

print("Time taken for creation of dataframe is {}".format(datetime.datetime.now() - startT
```

Saved successfully!

```
8:39.178276
frame is 0:00:02.927990
```

```
# for Train data

x1 = list(df_final_train['svd_u_s_1'])
x2 = list(df_final_train['svd_u_s_2'])
x3 = list(df_final_train['svd_u_s_3'])
x4 = list(df_final_train['svd_u_s_4'])
x5 = list(df_final_train['svd_u_s_5'])
x6 = list(df_final_train['svd_u_s_6'])

x7 = list(df_final_train['svd_u_d_1'])
x8 = list(df_final_train['svd_u_d_2'])
x9 = list(df_final_train['svd_u_d_3'])
x10 = list(df_final_train['svd_u_d_4'])
x11 = list(df_final_train['svd_u_d_5'])
x12 = list(df_final_train['svd_u_d_6'])

y1 = list(df_final_train['svd_v_s_1'])
y2 = list(df_final_train['svd_v_s_2'])
y3 = list(df_final_train['svd_v_s_3'])
y4 = list(df_final_train['svd_v_s_4'])
y5 = list(df_final_train['svd_v_s_5'])
y6 = list(df_final_train['svd_v_s_6'])
```

```
y7 = list(df_final_train['svd_v_d_1'])
y8 = list(df_final_train['svd_v_d_2'])
y9 = list(df_final_train['svd_v_d_3'])
y10 = list(df_final_train['svd_v_d_4'])
y11 = list(df_final_train['svd_v_d_5'])
y12 = list(df_final_train['svd_v_d_6'])

print(np.shape(x1))
print(np.shape(x2))
print(np.shape(x3))
print(np.shape(x4))
print(np.shape(x5))
print(np.shape(x6))
print(np.shape(x7))
print(np.shape(x8))
print(np.shape(x9))
print(np.shape(x10))
print(np.shape(x11))
print(np.shape(x12))

print(np.shape(y1))
print(np.shape(y2))
print(np.shape(y3))
print(np.shape(y4))
print(np.shape(y5))
print(np.shape(y6))
print(np.shape(y7))
print(np.shape(y8))
print(np.shape(y9))
```

Saved successfully!

```
print(np.shape(y12))

train_u_source = []
train_u_destination = []
train_v_source = []
train_v_destination = []
train_u_s_dot = []
train_u_d_dot = []

for loop1 in range(0,len(x1)):
    train_u_source.append(x1[loop1])
    train_u_source.append(x2[loop1])
    train_u_source.append(x3[loop1])
    train_u_source.append(x4[loop1])
    train_u_source.append(x5[loop1])
    train_u_source.append(x6[loop1])

    train_u_destination.append(x7[loop1])
    train_u_destination.append(x8[loop1])
    train_u_destination.append(x9[loop1])
    train_u_destination.append(x10[loop1])
    train_u_destination.append(x11[loop1])
    train_u_destination.append(x12[loop1])
```

```
        dot_product = np.dot(train_u_source[loop1],train_u_destination[loop1])
        train_u_s_dot.append(dot_product)


    for loop2 in range(0,len(y1)):
        train_v_source.append(y1[loop2])
        train_v_source.append(y2[loop2])
        train_v_source.append(y3[loop2])
        train_v_source.append(y4[loop2])
        train_v_source.append(y5[loop2])
        train_v_source.append(y6[loop2])

        train_v_destination.append(y7[loop2])
        train_v_destination.append(y8[loop2])
        train_v_destination.append(y9[loop2])
        train_v_destination.append(y10[loop2])
        train_v_destination.append(y11[loop2])
        train_v_destination.append(y12[loop2])

        dot_product = np.dot(train_v_source[loop2],train_v_destination[loop2])
        train_u_d_dot.append(dot_product)

    print("*******************************************")
    print(np.shape(train_u_s_dot))
    print(np.shape(train_u_d_dot))
```

```
(100002,)
(100002,)
```

Saved successfully!

```
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
(100002,)
*******************************************
(100002,)
(100002,)
```

```
# for Test data

x1 = list(df_final_test['svd_u_s_1'])
x2 = list(df_final_test['svd_u_s_2'])
x3 = list(df_final_test['svd_u_s_3'])
x4 = list(df_final_test['svd_u_s_4'])
x5 = list(df_final_test['svd_u_s_5'])
x6 = list(df_final_test['svd_u_s_6'])

x7 = list(df_final_test['svd_u_d_1'])
x8 = list(df_final_test['svd_u_d_2'])
x9 = list(df_final_test['svd_u_d_3'])
x10 = list(df_final_test['svd_u_d_4'])
x11 = list(df_final_test['svd_u_d_5'])
x12 = list(df_final_test['svd_u_d_6'])

y1 = list(df_final_test['svd_v_s_1'])
y2 = list(df_final_test['svd_v_s_2'])
y3 = list(df_final_test['svd_v_s_3'])
y4 = list(df_final_test['svd_v_s_4'])
y5 = list(df_final_test['svd_v_s_5'])
y6 = list(df_final_test['svd_v_s_6'])

y7 = list(df_final_test['svd_v_d_1'])
y8 = list(df_final_test['svd_v_d_2'])
y9 = list(df_final_test['svd_v_d_3'])
y10 = list(df_final_test['svd_v_d_4'])
y11 = list(df_final_test['svd_v_d_5'])
y12 = list(df_final_test['svd_v_d_6'])
```



```
print(np.shape(x3))
print(np.shape(x4))
print(np.shape(x5))
print(np.shape(x6))
print(np.shape(x7))
print(np.shape(x8))
print(np.shape(x9))
print(np.shape(x10))
print(np.shape(x11))
print(np.shape(x12))

print(np.shape(y1))
print(np.shape(y2))
print(np.shape(y3))
print(np.shape(y4))
print(np.shape(y5))
print(np.shape(y6))
print(np.shape(y7))
print(np.shape(y8))
print(np.shape(y9))
print(np.shape(y10))
print(np.shape(y11))
print(np.shape(y12))
```

```
test_u_source = []
test_u_destination = []
test_v_source = []
test_v_destination = []
test_v_s_dot = []
test_v_d_dot = []

for loop3 in range(0,len(x1)):
    test_u_source.append(x1[loop3])
    test_u_source.append(x2[loop3])
    test_u_source.append(x3[loop3])
    test_u_source.append(x4[loop3])
    test_u_source.append(x5[loop3])
    test_u_source.append(x6[loop3])

    test_u_destination.append(x7[loop3])
    test_u_destination.append(x8[loop3])
    test_u_destination.append(x9[loop3])
    test_u_destination.append(x10[loop3])
    test_u_destination.append(x11[loop3])
    test_u_destination.append(x12[loop3])

    dot_product = np.dot(test_u_source[loop3],test_u_destination[loop3])
    test_v_s_dot.append(dot_product)

for loop4 in range(0,len(y1)):
    test_v_source.append(y1[loop4])
    test_v_source.append(y2[loop4])
    test_v_source.append(y5[loop4])
    test_v_source.append(y6[loop4])

    test_v_destination.append(y7[loop4])
    test_v_destination.append(y8[loop4])
    test_v_destination.append(y9[loop4])
    test_v_destination.append(y10[loop4])
    test_v_destination.append(y11[loop4])
    test_v_destination.append(y12[loop4])

    dot_product = np.dot(test_v_source[loop4],test_v_destination[loop4])
    test_v_d_dot.append(dot_product)

print("****************************************")
print(np.shape(test_v_s_dot))
print(np.shape(test_v_d_dot))
```

Saved successfully!

```
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
```

```
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
(50002,)
*******************************************
(50002,)
(50002,)
```

```
startTime = datetime.datetime.now()
print("Current Time = ",startTime)

if not os.path.isfile('data/fea_sample/storage_sample_stage6.h5'):
    #=================================================================================
    df_final_train['s_dot'] = np.array(train_u_s_dot)
    df_final_train['d_dot'] = np.array(train_u_d_dot)

    df_final_test['s_dot'] = np.array(test_v_s_dot)
                                    y(test_v_d_dot)

    hdf = HDFStore('data/fea_sample/storage_sample_stage6.h5')
    hdf.put('train_df',df_final_train, format='table', data_columns=True)
    hdf.put('test_df',df_final_test, format='table', data_columns=True)
    hdf.close()
else:
    df_final_train = read_hdf('data/fea_sample/storage_sample_stage6.h5', 'train_df',mode=
    df_final_test = read_hdf('data/fea_sample/storage_sample_stage6.h5', 'test_df',mode='r

print("Time taken for creation of dataframe is {}".format(datetime.datetime.now() - startT
```



```
Current Time =  2019-05-24 21:20:42.572386
Time taken for creation of dataframe is 0:00:21.119923
```

```
#reading
from pandas import read_hdf
df_final_train = read_hdf('data/fea_sample/storage_sample_stage6.h5', 'train_df',mode='r')
df_final_test = read_hdf('data/fea_sample/storage_sample_stage6.h5', 'test_df',mode='r')
```

```
df_final_test.ix[:,'adar_index':][:10]
```

	adar_index	follows_back	same_comp	shortest_path	weight_in	weight_out	weight_
0	0.000000	1	1	2	0.258199	0.377964	0.636
1	0.000000	1	1	7	0.235702	0.707107	0.942
2	0.000000	0	1	5	0.301511	0.242536	0.544
3	0.000000	0	1	3	0.162221	0.301511	0.463
4	6.136433	0	1	2	0.188982	0.250000	0.438
5	0.000000	0	0	-1	0.588969	0.301511	0.890
6	0.000000	1	1	-1	1.000000	0.353553	1.353
7	3.095903	1	1	2	0.250000	0.288675	0.538
8	0.000000	0	0	-1	0.588969	0.301511	0.890
9	0.000000	1	1	-1	0.377964	1.000000	1.377

10 rows × 48 columns

]

Saved successfully!

	weight_f4	page_rank_s	page_rank_d	katz_s	katz_d	hubs_s	hubs_d
0	1.014128	6.557971e-07	1.559547e-06	0.000754	0.000786	3.243237e-16	1.745627e-16
1	1.649916	2.172064e-07	1.148360e-06	0.000739	0.000801	1.702625e-19	2.706300e-15

```
df_final_test.ix[:,'svd_u_s_1':][:10]
```

	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
0	-9.987980e-12	2.283708e-13	1.439960e-10	6.136158e-13	4.188181e-13	5.985839e-15	-1.026186e-11
1	-2.079268e-15	7.011592e-16	6.970671e-10	9.562313e-15	1.729157e-15	2.944887e-18	-1.960465e-11
2	-1.782381e-11	2.389626e-11	1.443051e-06	2.358027e-12	2.689642e-12	1.739184e-12	-1.863949e-13
3	-4.491249e-11	9.917580e-13	7.891237e-06	9.458626e-12	2.715849e-11	1.822079e-12	-1.070734e-10
4	-9.778569e-13	5.724678e-13	4.951878e-06	1.396835e-12	2.153250e-11	2.872774e-14	-2.275782e-11
5	-8.977767e-15	4.541045e-15	9.100025e-14	2.419543e-13	4.586857e-14	4.601913e-15	0.000000e+00
6	-2.143381e-11	1.069559e-09	3.714264e-11	2.544021e-11	2.935397e-11	4.036789e-13	-5.328096e-14
7			37949e-13	4.402437e-15	1.083419e-14	3.136596e-16	-1.012196e-11
8	-9.936400e-14	1.079878e-12	5.993042e-12	1.324362e-13	2.225664e-08	1.171559e-15	0.000000e+00
9	-4.079425e-20	-1.681818e-20	-1.769057e-19	9.391413e-20	1.097935e-19	-1.897890e-19	-1.277149e-17

10 rows × 30 columns

Saved successfully!

```
df_final_test.ix[:,'svd_u_d_4':][:10]
```

	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_
0	1.166038e-13	2.253356e-11	3.222301e-15	-2.148853e-13	1.883259e-13	5.904813e-11	2.70153
1	1.907402e-12	3.797448e-11	4.993662e-14	-4.054500e-13	2.895773e-13	2.545371e-10	2.24860
2	1.693277e-13	2.712767e-13	7.596286e-15	-4.148335e-13	4.618813e-12	1.122147e-05	1.77893
3	7.229387e-12	3.563661e-11	1.917977e-12	-8.942576e-12	5.535692e-12	5.223671e-06	7.91721
4	1.394103e-10	2.961998e-11	2.022055e-13	-3.804983e-12	1.593137e-13	1.035013e-06	1.36158
5	0.000000e+00	0.000000e+00	0.000000e+00	-2.065320e-20	1.431037e-20	1.688691e-19	4.50255
6	3.072229e-13	2.571312e-13	1.059874e-14	-5.259869e-12	4.156213e-09	1.005038e-09	3.21443

```
df_final_test.ix[:,'svd_v_d_1':][:10]
```

	svd_v_d_1	svd_v_d_2	svd_v_d_3	svd_v_d_4	svd_v_d_5	svd_v_d_6
0	-9.994077e-10	5.791914e-10	3.512351e-07	2.486660e-09	2.771146e-09	1.727695e-12
1	-9.360663e-12	3.206811e-10	4.668669e-08	6.665782e-12	1.495980e-10	9.836744e-14
	4.253080e-		3.479854e-07	1.630555e-13	3.954762e-13	3.875775e-14
3	-2.162598e-11	6.939286e-12	1.879855e-05	4.384839e-12	1.239415e-11	6.483494e-13
4	-8.742939e-12	7.467395e-12	1.256876e-05	3.637116e-12	3.948477e-12	2.415871e-13
5	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
6	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
7	-9.799884e-10	8.288353e-13	4.597542e-11	5.675210e-13	4.877886e-13	3.437278e-14
8	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
9	-1.036071e-13	6.846028e-15	2.417007e-13	6.131609e-14	1.297121e-13	6.960836e-16

Saved successfully!

```
y_train = df_final_train.indicator_link
y_test = df_final_test.indicator_link
```

```
df_final_train.columns
```

```
Index(['jaccard_followers', 'jaccard_followees', 'cosine_followers',
       'cosine_followees', 'num_followers_s', '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',
       'followee_preferential_attachment', 'follower_preferential_attachment',
       'train_u_s_dot', 'train_u_d_dot', 's_dot', 'd_dot'],
      dtype='object')
```

```
df_final_train.drop(['source_node', 'destination_node','indicator_link','train_u_s_dot','t
df_final_test.drop(['source_node', 'destination_node','indicator_link','test_v_s_dot','tes
```

```
df_final_train.drop(['train_u_s_dot','train_u_d_dot'],axis=1,inplace=True)
df_final_test.drop(['test_v_s_dot','test_v_d_dot'],axis=1,inplace=True)
```

```
df_final_train.columns
```

```
Index(['jaccard_followers', 'jaccard_followees', 'cosine_followers',
       'cosine_followees', 'num_followers_s', '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',
       'followee_preferential_attachment', 'follower_preferential_attachment',
       's_dot', 'd_dot'],
      dtype='object')
```

Saved successfully!

```
df_final_test.columns
```

```
Index(['jaccard_followers', 'jaccard_followees', 'cosine_followers',
       'cosine_followees', 'num_followers_s', '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',
       'followee_preferential_attachment', 'follower_preferential_attachment',
```

```
       's_dot', 'd_dot'],
      dtype='object')
```

Random Forest

```
startTime = datetime.datetime.now()
print("Current Time = ",startTime)

estimators = [10,50,100,250,450]
train_scores = []
test_scores = []
for i in estimators:
    clf = RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=5, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=52, min_samples_split=120,
            min_weight_fraction_leaf=0.0, n_estimators=i,random_state=25,verbose=0,warm_st
    clf.fit(df_final_train,y_train)
    train_sc = f1_score(y_train,clf.predict(df_final_train))
    test_sc = f1_score(y_test,clf.predict(df_final_test))
    test_scores.append(test_sc)
    train_scores.append(train_sc)
    print('Estimators = ',i,'Train Score',train_sc,'test Score',test_sc)
plt.plot(estimators,train_scores,label='Train Score')
plt.plot(estimators,test_scores,label='Test Score')
plt.xlabel('Estimators')
```

Saved successfully!

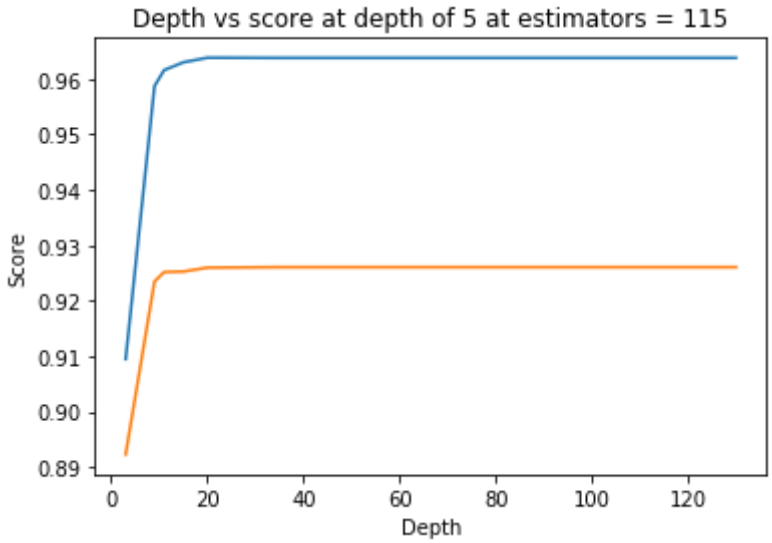
```
pth of 5')

print("Time taken for creation of dataframe is {}".format(datetime.datetime.now() - startT
```

```
Current Time =  2019-05-24 17:15:40.888059
Estimators =  10 Train Score 0.9168054647804179 test Score 0.8991258141926637
```

```
startTime = datetime.datetime.now()
print("Current Time = ",startTime)

depths = [3,9,11,15,20,35,50,70,130]
train_scores = []
test_scores = []
for i in depths:
    clf = RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=i, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=52, min_samples_split=120,
            min_weight_fraction_leaf=0.0, n_estimators=115, random_state=25,verbose=0,warm
    clf.fit(df_final_train,y_train)
    train_sc = f1_score(y_train,clf.predict(df_final_train))
    test_sc = f1_score(y_test,clf.predict(df_final_test))
    test_scores.append(test_sc)
    train_scores.append(train_sc)
    print('depth = ',i,'Train Score',train_sc,'test Score',test_sc)
plt.plot(depths,train_scores,label='Train Score')
plt.plot(depths,test_scores,label='Test Score')
plt.xlabel('Depth')
plt.ylabel('Score')
plt.title('Depth vs score at depth of 5 at estimators = 115')
plt.show()

print("Time taken for creation of dataframe is {}".format(datetime.datetime.now() - startT
```

Saved successfully!

```
Current Time =  2019-05-24 17:22:55.575829
depth =  3 Train Score 0.8929294708766 test Score 0.8923076923076922
depth =  9 Train Score 0.9237982787979 test Score 0.9234631450149783
depth =  11 Train Score 0.9615533269594579 test Score 0.92516748830742
depth =  15 Train Score 0.9629599545122248 test Score 0.9252643552260185
depth =  20 Train Score 0.963790564217155 test Score 0.9259657330720111
depth =  35 Train Score 0.963760461525961 test Score 0.9260593800703796
depth =  50 Train Score 0.963760461525961 test Score 0.9260593800703796
depth =  70 Train Score 0.963760461525961 test Score 0.9260593800703796
depth =  130 Train Score 0.963760461525961 test Score 0.9260593800703796
```




```
Time taken for creation of dataframe is 0:11:28.420432
```

```
from sklearn.metrics import f1_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint as sp_randint
from scipy.stats import uniform

startTime = datetime.datetime.now()
print("Current Time = ",startTime)

param_dist = {"n_estimators":sp_randint(105,125),
              "max_depth": sp_randint(10,15),
              "min_samples_split": sp_randint(110,190),
              "min_samples_leaf": sp_randint(25,65)}

clf = RandomForestClassifier(random_state=25)

rf_random = RandomizedSearchCV(clf, param_distributions=param_dist,
                                    n_iter=5,cv=10,scoring='f1',random_state=25)

rf_random.fit(df_final_train,y_train)
print('mean test scores',rf_random.cv_results_['mean_test_score'])
print('mean train scores',rf_random.cv_results_['mean_train_score'])

print("*****************************************************")
print(rf_random.best_estimator_)

clf = RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=14, max_features='auto', max_leaf_nodes=None,
                                    0, min_impurity_split=None,
                                    _samples_split=111,
            min_weight_fraction_lea=0.0, n_estimators=121, n_jobs=-1,
            oob_score=False, random_state=25, verbose=0, warm_start=False)

clf.fit(df_final_train,y_train)
y_train_pred = clf.predict(df_final_train)
y_test_pred = clf.predict(df_final_test)

from sklearn.metrics import f1_score
print('\nTrain f1 score',f1_score(y_train,y_train_pred))
print('Test f1 score',f1_score(y_test,y_test_pred))

print("Time taken for creation of dataframe is {}".format(datetime.datetime.now() - startT
```

```
Current Time =  2019-05-24 17:34:24.043261
mean test scores [0.96230736 0.96209145 0.96068497 0.96182116 0.96318534]
mean train scores [0.9633391  0.96287272 0.96126794 0.96251656 0.96423855]
*****************************************************
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=14, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=28, min_samples_split=111,
            min_weight_fraction_leaf=0.0, n_estimators=121, n_jobs=None,
            oob_score=False, random_state=25, verbose=0, warm_start=False)
```

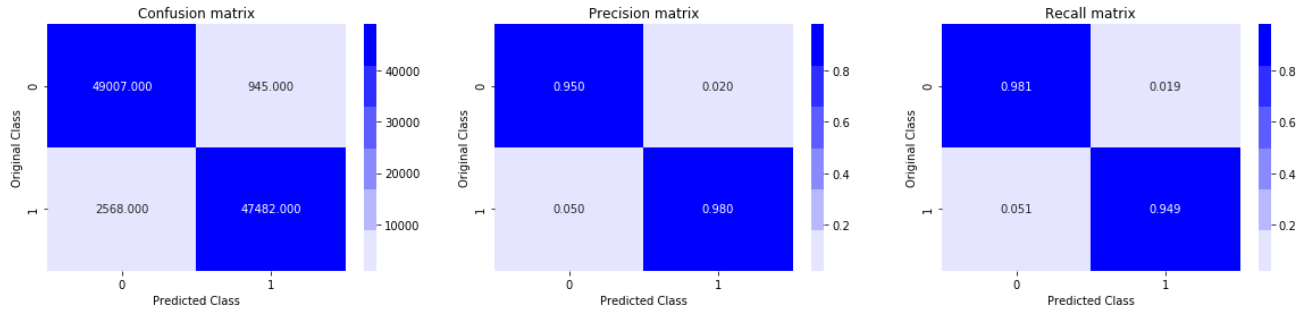
```
Train f1 score 0.9643266955735856
Test f1 score 0.9263264402706634
Time taken for creation of dataframe is 0:59:16.060665
```

```
from sklearn.metrics import confusion_matrix
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)

    A =(((C.T)/(C.sum(axis=1))).T)

    B =(C/C.sum(axis=0))
    plt.figure(figsize=(20,4))

    labels = [0,1]
    # representing A in heatmap format
    cmap=sns.light_palette("blue")
    plt.subplot(1, 3, 1)
    sns.heatmap(C, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=label
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Confusion matrix")

    plt.subplot(1, 3, 2)
    sns.heatmap(B, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=label
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Precision matrix")

    plt.subplot(1, 3, 3)
    # representing B in heatmap format
    sns.heatmap(A, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=label
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Recall matrix")

    plt.show()
```

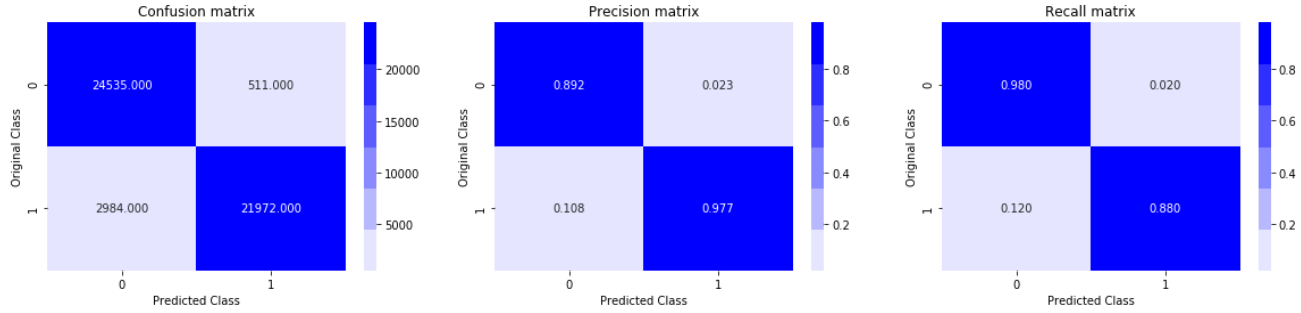
Saved successfully!

```
print('Train confusion_matrix')
plot_confusion_matrix(y_train,y_train_pred)
print('Test confusion_matrix')
plot_confusion_matrix(y_test,y_test_pred)
```

```
Train confusion_matrix
```

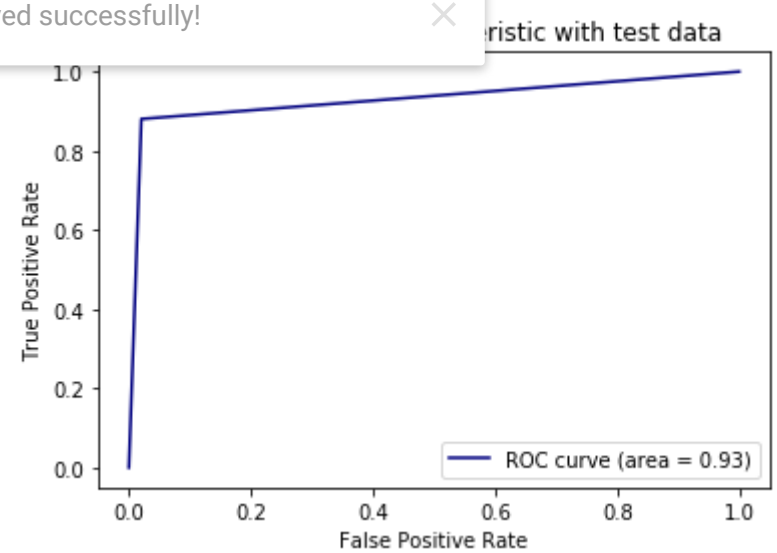



```
Test confusion_matrix
```



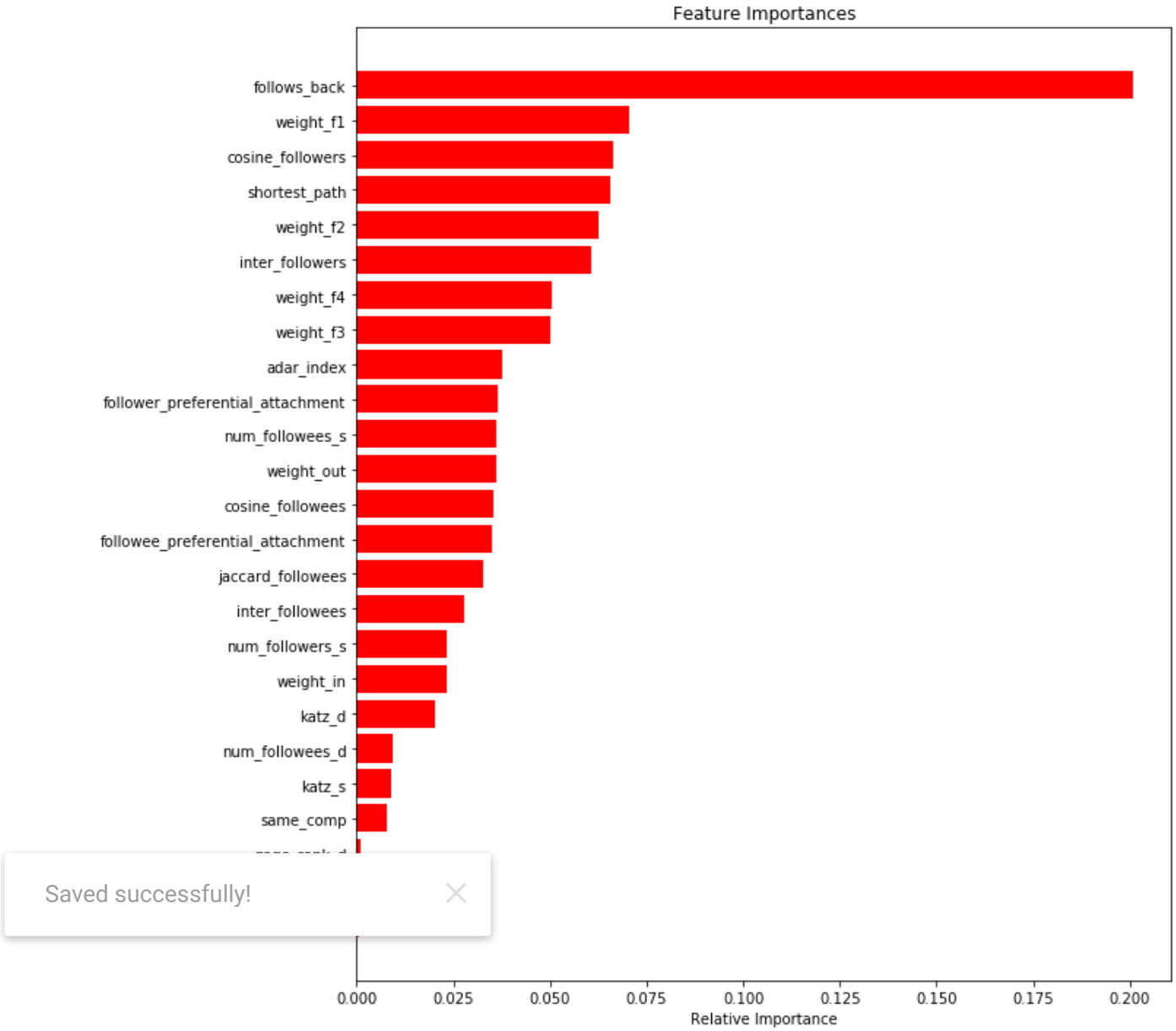

```
from sklearn.metrics import roc_curve, auc
fpr,tpr,ths = roc_curve(y_test,y_test_pred)
auc_sc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='navy',label='ROC curve (area = %0.2f)' % auc_sc)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic with test data')
plt.legend()
plt.show()
```

Saved successfully!




```
features = df_final_train.columns
importances = clf.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()
```



Saved successfully!

Hyperparameter tuning XGBoost

```
startTime = datetime.datetime.now()
print("Current Time = ",startTime)

import xgboost as xgb
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import f1_score,make_scorer

min_child_weight = [2,4,6]
max_depth = [2,4,6]
```

```
n_estimators =[100,200,300]
learning_rate = [0.1,0.2,0.3]

scorer = make_scorer(f1_score)
tuned_parameters = {
                    'min_child_weight':min_child_weight,
                    'max_depth':max_depth,
                    'n_estimators': n_estimators,
                    'learning_rate':learning_rate}

clf = xgb.XGBClassifier()
model_gbt = RandomizedSearchCV(clf,tuned_parameters,scoring =scorer,cv=3,pre_dispatch=2)
model_gbt.fit(df_final_train,y_train)
print(model_gbt.best_estimator_)

best_min_child_weight_xgb = model_gbt.best_estimator_.min_child_weight
best_max_depth_xgb = model_gbt.best_params_["max_depth"]
best_n_estimators_xgb = model_gbt.best_estimator_.n_estimators
best_learning_rate_xgb = model_gbt.best_estimator_.learning_rate

print("\nbest_min_child_weight_xgb = ", best_min_child_weight_xgb)
print("best_max_depth_xgb = ",best_max_depth_xgb)
print("best_n_estimators_xgb = ", best_n_estimators_xgb)
print("best_learning_rate_xgb = ",best_learning_rate_xgb)

print("Time taken for creation of dataframe is {}".format(datetime.datetime.now() - startT
```

```
     Current Time =  2019-05-24 21:25:09.249187
     XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
            colsample_bytree=1, gamma=0, learning_rate=0.3, max_delta_step=0,
                                                    eight=4, missing=None, n_estimators=300,
                                                    objective='binary:logistic', random_state=0,
                                                    1, scale_pos_weight=1, seed=None,
            silent=True, subsample=1)

     best_min_child_weight_xgb =  4
     best_max_depth_xgb =  4
     best_n_estimators_xgb =  300
     best_learning_rate_xgb =  0.3
     Time taken for creation of dataframe is 1:55:39.179742
```



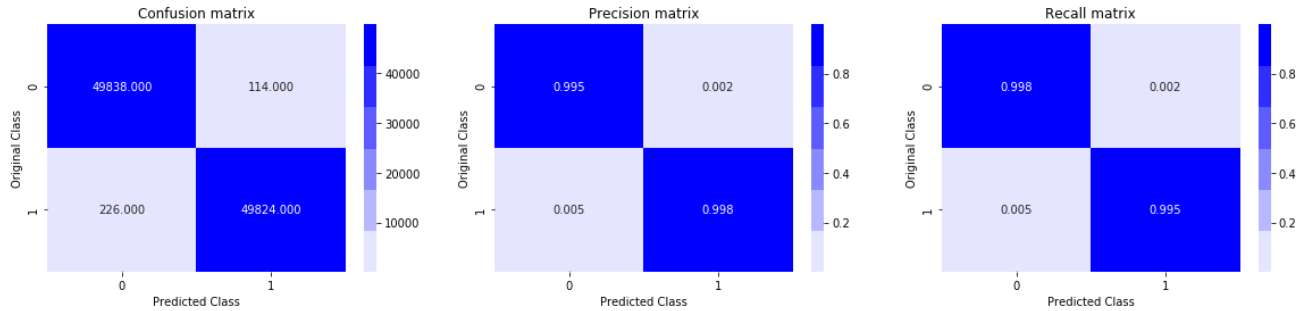
```
startTime = datetime.datetime.now()
print("Current Time = ",startTime)

xgb_best = xgb.XGBClassifier(objective='binary:logistic',learning_rate = best_learning_rat
                             min_child_weight = best_min_child_weight_xgb,n_estimators = b
                             max_depth = best_max_depth_xgb)
xgb_best.fit(df_final_train,y_train)
pred_train = xgb_best.predict(df_final_train)
pred_test = xgb_best.predict(df_final_test)

train_score = f1_score(y_train,pred_train)
test_score = f1_score(y_test,pred_test)
print('\nTrain Score: ',train_score)
print('Test Score: ',test_score)
```

```
print("Time taken for creation of dataframe is {}".format(datetime.datetime.now() - startT
```

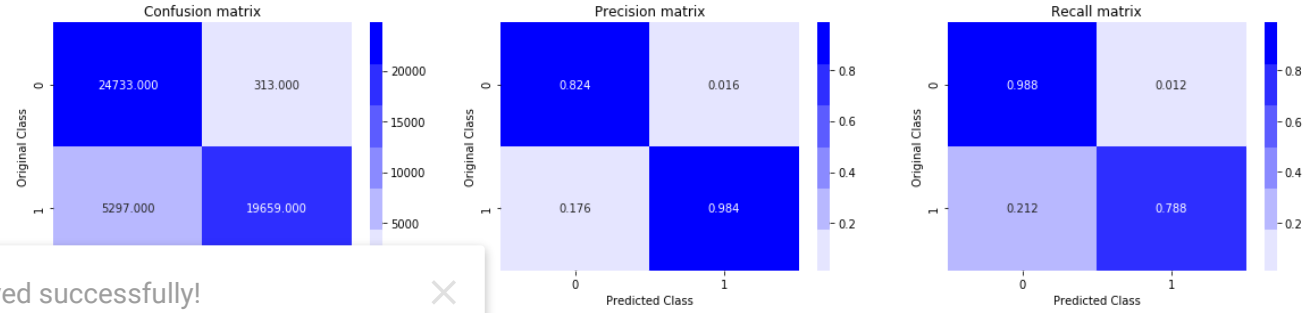
```
Current Time =  2019-05-24 23:20:48.456948

Train Score:  0.9965995919510341
Test Score:  0.875133547008547
Time taken for creation of dataframe is 0:06:15.302619
```

```
print('Train confusion_matrix')
plot_confusion_matrix(y_train,pred_train)
print('Test confusion_matrix')
plot_confusion_matrix(y_test,pred_test)
```

```
Train confusion_matrix
```

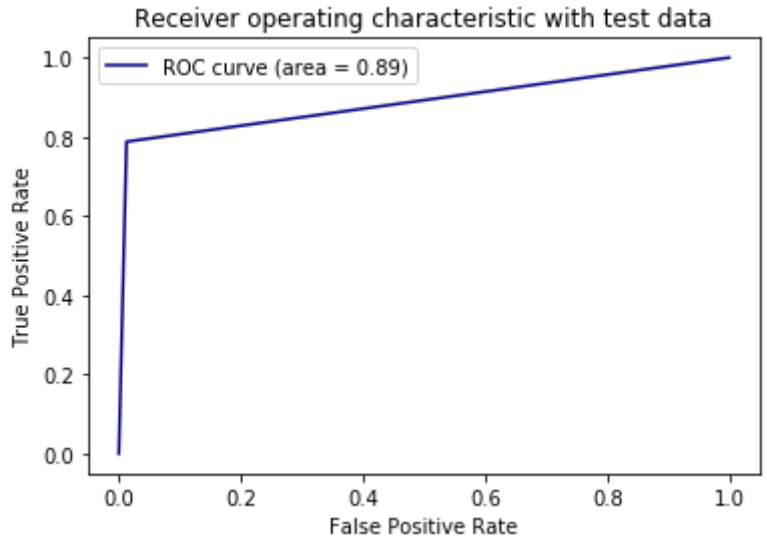



```
Test confusion_matrix
```



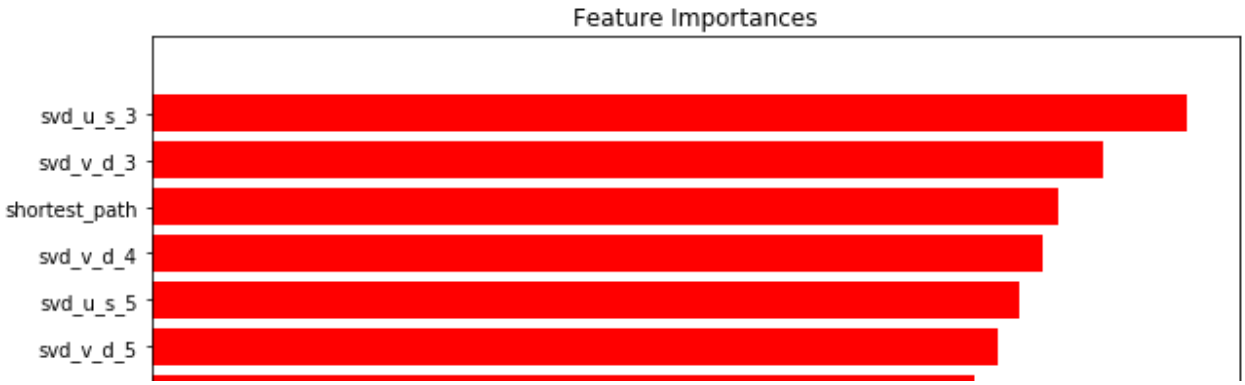
Saved successfully!

```
from sklearn.metrics import roc_curve, auc
fpr,tpr,ths = roc_curve(y_test,pred_test)
auc_sc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='navy',label='ROC curve (area = %0.2f)' % auc_sc)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic with test data')
plt.legend()
plt.show()
```




```
features = df_final_train.columns
importances = xgb_best.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()
```

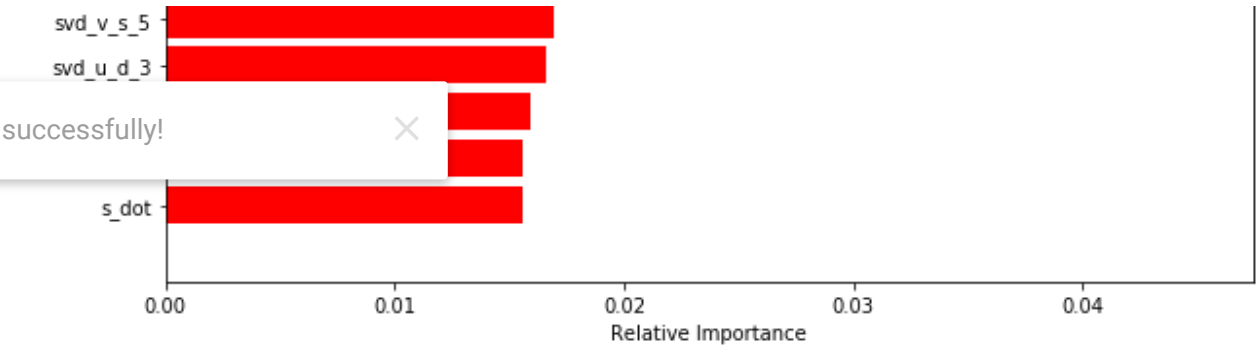
Saved successfully!




```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Model \ Parameters","Train f1_score","Test f1_score"]
x.add_row(["RandomForest: ",0.9643266955735856,0.9263264402706634])
x.add_row(["XGBClassifier: ",0.9965995919510341,0.875133547008547])
print(x)
```

```
+--------------------+--------------------+--------------------+
| Model \ Parameters |   Train f1_score   |   Test f1_score    |
+--------------------+--------------------+--------------------+
|   RandomForest:    | 0.9643266955735856 | 0.9263264402706634 |
|  XGBClassifier:    | 0.9965995919510341 | 0.875133547008547  |
+--------------------+--------------------+--------------------+
```

Thanks Applied AI, it was a great assignment.



Saved successfully!