In [ ]:

```
#Importing Libraries
# please do go through this python notebook:
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
# if numpy is not installed already : pip3 install numpy
import numpy as np#Do aritmetic operations on arrays
# matplotlib: used to plot graphs
import matplotlib
import matplotlib.pylab as plt
import seaborn as sns#Plots
from matplotlib import rcParams#Size of plots
from sklearn.cluster import MiniBatchKMeans, KMeans#Clustering
import math
import pickle
import os
# to install xgboost: pip3 install xgboost
import xgboost as xgb

import warnings
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
from sklearn.metrics import f1_score
```

In [ ]:

```
import zipfile
```

In [ ]:

```
# importing required modules
from zipfile import ZipFile

# specifying the zip file name
file_name = "train.zip"

# opening the zip file in READ mode
with ZipFile(file_name, 'r') as zip:
    # printing all the contents of the zip file
    zip.printdir()

    # extracting all the files
    print('Extracting all the files now...')
    zip.extractall()
    print('Done!')
```

In [ ]:

```
# importing required modules
from zipfile import ZipFile

# specifying the zip file name
file_name = "test.zip"

# opening the zip file in READ mode
with ZipFile(file_name, 'r') as zip:
    # printing all the contents of the zip file
    zip.printdir()

    # extracting all the files
    print('Extracting all the files now...')
    zip.extractall()
    print('Done!')
```

In [ ]:

```
train_df = pd.read_csv("train.csv")
pos=train_df[train_df.is_chat==1]
neg=train_df[train_df.is_chat==0]
neg=neg.sample(len(pos))
del train_df
train_df=pd.concat([pos, neg], sort=False)
train_df=train_df.sample(frac=1)
del pos, neg
train_df =train_df.reset_index(drop=True)
```

In [ ]:

In [ ]:

In [ ]:

```
train=pd.read_csv("train.csv")
test=pd.read_csv("test.csv")
```

In [ ]:

```
test.shape , train.shape
```

In [ ]:

```
for_graph=pd.concat([train[["node1_id", "node2_id"]], test[["node1_id", "node2_id"]]], sort=False)
```

In [ ]:

```
for_graph.to_csv('for_graph.csv',header=False,index=False)
```

In [ ]:

```
train_graph=nx.read_edgelist("for_graph.csv",delimiter=',',nodetype=int)
print(nx.info(train_graph))
```

In [ ]:

```
from pandas import HDFStore,DataFrame
from pandas import read_hdf
from scipy.sparse.linalg import svds, eigs
import gc
from tqdm import tqdm
```

In [ ]:

```
test_df=pd.read_csv("test.csv")
```

In [ ]:

```
#for followers
def jaccard_for_followers(a,b):
    try:
        if len(set(train_graph.neighbours(a))) == 0  | len(set(g.neighbours(b
))) == 0:
            return 0
        sim = (len(set(train_graph.neighbors(a)).intersection(set(train_graph.
neighbors(b)))))/\
                                 (len(set(train_graph.neighbors(a)).union(set(
train_graph.neighbors(b)))))
        return sim
    except:
        return 0
```

train_df['jaccard_common_contact'] = train_df.apply(lambda row: jaccard_for_followers(row['node1_id'],row['node2_id']),axis=1) test_df['jaccard_common_contact'] = test_df.apply(lambda row: jaccard_for_followers(row['node1_id'],row['node2_id']),axis=1)

In [ ]:

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

        if len(set(train_graph.neighbors(a))) == 0  | len(set(train_graph.neig
hbors(b))) == 0:
            return 0
        sim = (len(set(train_graph.neighbors(a)).intersection(set(train_graph.
neighbors(b)))))/\
                                     (math.sqrt(len(set(train_graph.neighbors(
a))))*(len(set(train_graph.neighbors(b)))))
        return sim
    except:
        return 0
```

In [ ]:

```
print(cosine_for_followers(2,4702))
```

train_df['cosine_common_contact'] = train_df.apply(lambda row: cosine_for_followers(row['node1_id'],row['node2_id']),axis=1) test_df['cosine_common_contact'] = test_df.apply(lambda row: cosine_for_followers(row['node1_id'],row['node2_id']),axis=1)

In [ ]:

In [ ]:

```
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

    inter_followers=[]

    for i,row in df_final.iterrows():
        try:
            s1=set(train_graph.neighbors(row['node1_id']))
        except:
            s1 = set()
        try:
            d1=set(train_graph.neighbors(row['node2_id']))
        except:
            d1 = set()

        inter_followers.append(len(s1.intersection(d1)))

    return inter_followers
```

train_df['common_contact']= compute_features_stage1(train_df)test_df['common_contact']= compute_features_stage1(test_df)

In [ ]:

train_df["n1_tot_contact"] = train_df["node1_id"].apply(lambda x:len(set(train_graph.neighbors(x))))
test_df["n1_tot_contact"] = test_df["node1_id"].apply(lambda x:len(set(train_graph.neighbors(x))))
train_df["n2_tot_contact"] = train_df["node2_id"].apply(lambda x:len(set(train_graph.neighbors(x))))
test_df["n2_tot_contact"] = test_df["node2_id"].apply(lambda x:len(set(train_graph.neighbors(x))))

In [ ]:

In [ ]:

```
#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
```

In [ ]:

```
#testing
compute_shortest_path_length(77697, 826021)
```

#mapping shortest path on train train_df['shortest_path'] = train_df.apply(lambda row: compute_shortest_path_length(row['node1_id'],row['node2_id']),axis=1) #mapping shortest path on test test_df['shortest_path'] = test_df.apply(lambda row: compute_shortest_path_length(row['node1_id'],row['node2_id']),axis=1)

In [ ]:

```

```

In [ ]:

```
#adar index
def calc_adar_in(a,b):
    sum=0
    try:
        n=list(set(train_graph.neighbors(a)).intersection(set(train_graph.neig
hbors(b))))
        if len(n)!=0:
            for i in n:
                sum=sum+(1/np.log1p(len(list(train_graph.neighbors(i)))))
            return sum
        else:
            return 0
    except:
        return 0
```

In [ ]:

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

train_df['calc_adar_in'] = train_df.apply(lambda row: calc_adar_in(row['node1_id'],row['node2_id']),axis=1) #mapping adar index on test test_df['calc_adar_in'] = test_df.apply(lambda row: calc_adar_in(row['node1_id'],row['node2_id']),axis=1)

In [ ]:

```

```

In [ ]:

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

    s2=set(train_graph.neighbors(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()))
```

#mapping to pandas train train_df['weight_n1'] = train_df.node1_id.apply(lambda x: Weight_in.get(x,mean_weight_in)) train_df['weight_n2'] = train_df.node2_id.apply(lambda x: Weight_out.get(x,mean_weight_out)) #mapping to pandas test test_df['weight_n1'] = test_df.node1_id.apply(lambda x: Weight_in.get(x,mean_weight_in)) test_df['weight_n2'] = test_df.node2_id.apply(lambda x: Weight_out.get(x,mean_weight_out)) hdf = HDFStore('train_test_undirect_graph.h5') hdf.put('train_df',train_df, format='table', data_columns=True) hdf.put('test_df',test_df, format='table', data_columns=True) hdf.close()

In [ ]:

```
if not os.path.isfile("train_test_undirect_graph.h5"):

    train_df['jaccard_common_contact'] = train_df.apply(lambda row:
                                        jaccard_for_followers(row['node1_i
d'],row['node2_id']),axis=1)
    test_df['jaccard_common_contact'] = test_df.apply(lambda row:
                                        jaccard_for_followers(row['node1_i
d'],row['node2_id']),axis=1)

    train_df['cosine_common_contact'] = train_df.apply(lambda row:
                                        cosine_for_followers(row['node1_i
d'],row['node2_id']),axis=1)
    test_df['cosine_common_contact'] = test_df.apply(lambda row:
                                        cosine_for_followers(row['node1_i
d'],row['node2_id']),axis=1)


    train_df['common_contact']= compute_features_stage1(train_df)
    test_df['common_contact']= compute_features_stage1(test_df)

    train_df["n1_tot_contact"] = train_df["node1_id"].apply(lambda x:len(set(t
rain_graph.neighbors(x))))
    test_df["n1_tot_contact"] = test_df["node1_id"].apply(lambda x:len(set(tra
in_graph.neighbors(x))))

    train_df["n2_tot_contact"] = train_df["node2_id"].apply(lambda x:len(set(t
rain_graph.neighbors(x))))
    test_df["n2_tot_contact"] = test_df["node2_id"].apply(lambda x:len(set(tra
in_graph.neighbors(x))))


    #mapping shortest path on train
    train_df['shortest_path'] = train_df.apply(lambda row: compute_shortest_pa
th_length(row['node1_id'],row['node2_id']),axis=1)
    #mapping shortest path on test
    test_df['shortest_path'] = test_df.apply(lambda row: compute_shortest_path
_length(row['node1_id'],row['node2_id']),axis=1)

    train_df['calc_adar_in'] = train_df.apply(lambda row: calc_adar_in(row['no
de1_id'],row['node2_id']),axis=1)
    #mapping adar index on test
    test_df['calc_adar_in'] = test_df.apply(lambda row: calc_adar_in(row['node
1_id'],row['node2_id']),axis=1)

    #mapping to pandas train
    train_df['weight_n1'] = train_df.node1_id.apply(lambda x: Weight_in.get(x,
mean_weight_in))
    train_df['weight_n2'] = train_df.node2_id.apply(lambda x: Weight_out.get(x
,mean_weight_out))
    #mapping to pandas test
    test_df['weight_n1'] = test_df.node1_id.apply(lambda x: Weight_in.get(x,me
an_weight_in))
    test_df['weight_n2'] = test_df.node2_id.apply(lambda x: Weight_out.get(x,m
ean_weight_out))

    hdf = HDFStore('train_test_undirect_graph.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:
    train_df = read_hdf('train_test_undirect_graph.h5', 'train_df',mode='r')
    test_df = read_hdf('train_test_undirect_graph.h5', 'test_df',mode='r')
```

In [ ]:

```

```

In [2]:

```
#Importing Libraries
# please do go through this python notebook:
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
# if numpy is not installed already : pip3 install numpy
import numpy as np#Do aritmetic operations on arrays
# matplotlib: used to plot graphs
import matplotlib
import matplotlib.pylab as plt
import seaborn as sns#Plots
from matplotlib import rcParams#Size of plots
from sklearn.cluster import MiniBatchKMeans, KMeans#Clustering
import math
import pickle
import os
# to install xgboost: pip3 install xgboost
import xgboost as xgb

import warnings
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
from sklearn.metrics import f1_score
```

#reading from pandas import read_hdf train_df = read_hdf('train_test_undirect_graph.h5', 'train_df',mode='r') test_df = read_hdf('train_test_undirect_graph.h5', 'test_df',mode='r')

In [3]:

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

In [ ]:

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

In [5]:
```
if not os.path.isfile('train_test_undirect_graph_follows.h5'):
#mapping followback or not on train
    train_df['follows_back'] = train_df.apply(lambda row: follows_back(row['node1_id'],row['node2_id']),axis=1)

#mapping followback or not on test
    test_df['follows_back'] = test_df.apply(lambda row: follows_back(row['node1_id'],row['node2_id']),axis=1)

    hdf = HDFStore('train_test_undirect_graph_follows.h5')
    hdf.put('train_df',train_df, format='table', data_columns=True)
    hdf.put('test_df',test_df, format='table', data_columns=True)
    hdf.close()
else:
    train_df = read_hdf('train_test_undirect_graph_follows.h5', 'train_df',mode='r')
    test_df = read_hdf('train_test_undirect_graph_follows.h5', 'test_df',mode='r')
```

In [ ]:

In [ ]:

In [6]:
```
train_df.shape
```

Out[6]:
```
(6755352, 13)
```

In [7]:
```
train_df.columns
```

Out[7]:
```
Index(['node1_id', 'node2_id', 'is_chat', 'jaccard_common_contact',
       'cosine_common_contact', 'total_common', 'n1_tot_contact',
       'n2_tot_contact', 'shortest_path', 'calc_adar_in', 'weight_n1',
       'weight_n2', 'follows_back'],
      dtype='object')
```

In [8]:
```
feature=pd.read_csv("user_features.csv")
```

In [9]:
```
train_df = pd.merge(train_df, feature, how='left', left_on='node2_id', right_o
n='node_id')
del train_df["node_id"]
train_df=train_df.rename(index=str, columns={"f1":"n1_f1", "f2":'n1_f2',"f3":
'n1_f3',"f4":'n1_f4',"f5":'n1_f5',"f6":'n1_f6',\
                                            "f7":'n1_f7',"f8":'n1_f8',"f9":'n1
_f9',"f10":'n1_f10',"f11":'n1_f11',\
                                            "f12":'n1_f12',"f13":'n1_f13'})

train_df = pd.merge(train_df, feature, how='left', left_on='node1_id', right_o
n='node_id')
del train_df["node_id"]
train_df=train_df.rename(index=str, columns={"f1":"n2_f1", "f2":'n2_f2',"f3":
'n2_f3',"f4":'n2_f4',"f5":'n2_f5',"f6":'n2_f6',\
                                            "f7":'n2_f7',"f8":'n2_f8',"f9":'n2
_f9',"f10":'n2_f10',"f11":'n2_f11',\
                                            "f12":'n2_f12',"f13":'n2_f13'})
```

In [10]:
```
test_df = pd.merge(test_df, feature, how='left', left_on='node2_id', right_on=
'node_id')
del test_df["node_id"]
test_df=test_df.rename(index=str, columns={"f1":"n1_f1", "f2":'n1_f2',"f3":'n1
_f3',"f4":'n1_f4',"f5":'n1_f5',"f6":'n1_f6',\
                                            "f7":'n1_f7',"f8":'n1_f8',"f9":'n1
_f9',"f10":'n1_f10',"f11":'n1_f11',\
                                            "f12":'n1_f12',"f13":'n1_f13'})

test_df = pd.merge(test_df, feature, how='left', left_on='node1_id', right_on=
'node_id')
del test_df["node_id"]
test_df=test_df.rename(index=str, columns={"f1":"n2_f1", "f2":'n2_f2',"f3":'n2
_f3',"f4":'n2_f4',"f5":'n2_f5',"f6":'n2_f6',\
                                            "f7":'n2_f7',"f8":'n2_f8',"f9":'n2
_f9',"f10":'n2_f10',"f11":'n2_f11',\
                                            "f12":'n2_f12',"f13":'n2_f13'})
```

In [11]:
```
train_df.head(2)
```

Out[11]:

| | node1_id | node2_id | is_chat | jaccard_common_contact | cosine_common_contact | total_commo |
|---|---|---|---|---|---|---|
| **0** | 6542909 | 5443649 | 0 | 0.036364 | 0 | |
| **1** | 2768271 | 3512596 | 0 | 0.040816 | 0 | |

2 rows × 39 columns

In [12]:
```
train_df=train_df.head(500000)
```

In [13]:
```
test_df.head(2)
```

Out[13]:

|   | id | node1_id | node2_id | jaccard_common_contact | cosine_common_contact | total_common | n1 |
|---|---|---|---|---|---|---|---|
| **0** | 1 | 7107094 | 8010772 | 0.027027 | 0 | 2 |  |
| **1** | 2 | 7995251 | 2805801 | 0.041237 | 0 | 4 |  |

2 rows × 39 columns

In [14]:
```
ex=["is_chat", "id", "node1_id", "node2_id"]
target=train_df["is_chat"]
features=[col for col in train_df if col not in ex]
```

In [15]:
```
import lightgbm as lgb
from sklearn.metrics import roc_auc_score
```

In [16]:

```
from sklearn.model_selection import StratifiedKFold
%time
skf=StratifiedKFold(n_splits=3, shuffle=True, random_state=2019)
oof=np.zeros(len(train_df))
predictions=np.zeros(len(test_df))
feature_importance_df = pd.DataFrame()

start = time.time()
param = {"objective":"binary",
         "boost":"gbdt",
         "metric":"auc",
         "learning_rate":0.1,
         "num_leaves":12,
         "max_depth":-1,
         "tree_learner":"serial",
         #"feature_fraction":0.4,
         #"bagging_freq":5,
         #"bagging_fraction":0.4,
         "min_data_in_leaf":60,
         "min_sum_hessian_in_leaf":10,
         "n_jobs":-1,
         }

for fold_, (trn_idx, val_idx) in enumerate(skf.split(train_df.values , target.values )):
    #print("fold n{}".format(fold_))

    trn_data = lgb.Dataset(train_df[features].iloc[trn_idx], label = target.iloc[trn_idx])
    val_data = lgb.Dataset(train_df[features].iloc[val_idx], label = target.iloc[val_idx])

    num_round = 1000000
    clf = lgb.train(param, trn_data, num_round, valid_sets=[trn_data, val_data], verbose_eval = 1000, early_stopping_rounds=2000)
    oof[val_idx] = clf.predict(train_df[features].iloc[val_idx], num_iteration = clf.best_iteration)

    fold_importance_df = pd.DataFrame()
    fold_importance_df["feature"]=features
    fold_importance_df["importance"]=clf.feature_importance()
    fold_importance_df["fold"]=fold_+1
    feature_importance_df = pd.concat([feature_importance_df, fold_importance_df], axis=0)

    predictions+=clf.predict(test_df[features], num_iteration = clf.best_iteration)/skf.n_splits

feature_importance_df = feature_importance_df[["feature", 'importance']].groupby("feature").mean().sort_values(by = "importance", ascending=2000)
print("cv score: {:<8.5f}".format(roc_auc_score(target, oof)))
```

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 6.91 µs
Training until validation scores don't improve for 2000 rounds.
[1000]  training's auc: 0.883799        valid_1's auc: 0.872936
[2000]  training's auc: 0.894194        valid_1's auc: 0.873088
[3000]  training's auc: 0.902613        valid_1's auc: 0.872787
Early stopping, best iteration is:
[1714]  training's auc: 0.891469        valid_1's auc: 0.873226
Training until validation scores don't improve for 2000 rounds.
[1000]  training's auc: 0.884338        valid_1's auc: 0.872459
[2000]  training's auc: 0.894374        valid_1's auc: 0.872632
[3000]  training's auc: 0.902836        valid_1's auc: 0.872383
Early stopping, best iteration is:
[1640]  training's auc: 0.891016        valid_1's auc: 0.872693
Training until validation scores don't improve for 2000 rounds.
[1000]  training's auc: 0.884073        valid_1's auc: 0.87237
[2000]  training's auc: 0.894129        valid_1's auc: 0.872524
[3000]  training's auc: 0.902654        valid_1's auc: 0.872156
Early stopping, best iteration is:
[1767]  training's auc: 0.891883        valid_1's auc: 0.872699
cv score: 0.87287
```

In [17]:

```
result_2=pd.DataFrame({"id":test_df["id"], "is_chat":predictions})
result_2.to_csv("result_2.csv", index=False)
```

In [ ]:

In [ ]:

In [ ]: