Problem Statement

The *"Link Prediction Problem"* of Social Networks. Here, we have considered the online messaging app Hike's social network for the problem of Link Prediction.

Introduction

Given a snapshot of a social network, can we infer which new interactions among its members are likely to occur in the near future?

Determining the *'proximity'* of nodes in a network has many applications as given below:

- To suggest interactions or collaborations that haven't yet been utilized within an organization
- To monitor terrorist networks - to deduce possible interaction between terrorists (without direct evidence)
- Used in Facebook and Linked In to suggest friends

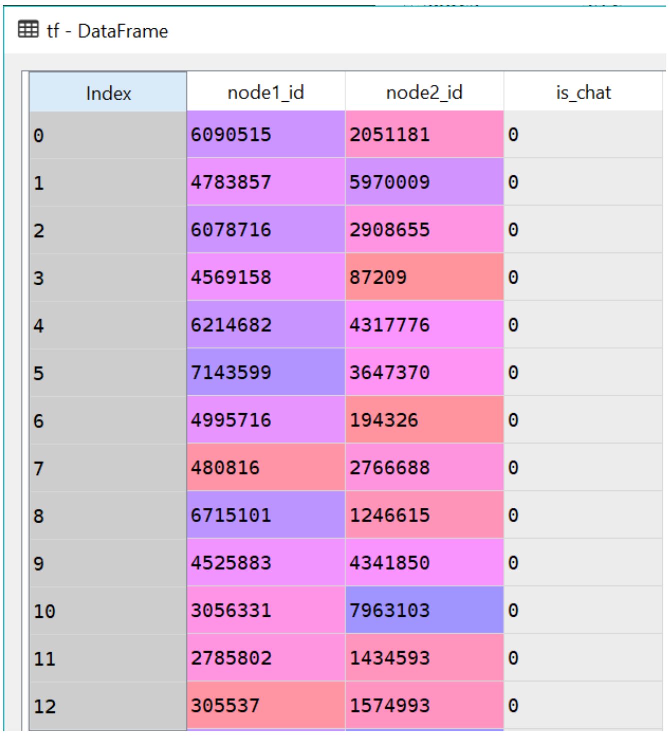
In the given problem statement, we are asked to find out whether if 2 individuals have chatted with each other, given their features.

Details of Datasets

As a part of *the Hike Social Network Prediction Hackathon: Hikeathon*, we were given a set 3 datasets:

1. Training data:

- The data consisted of pairs of nodes, representing people on the Hike social network.
- Each record contained the target value indicating whether the 2 actors have chatted with each other or not.
- The *is_chat* column is the target variable, indicating whether the two actors have communicated or not. ***1** meaning 'yes' and **0** meaning 'no'*

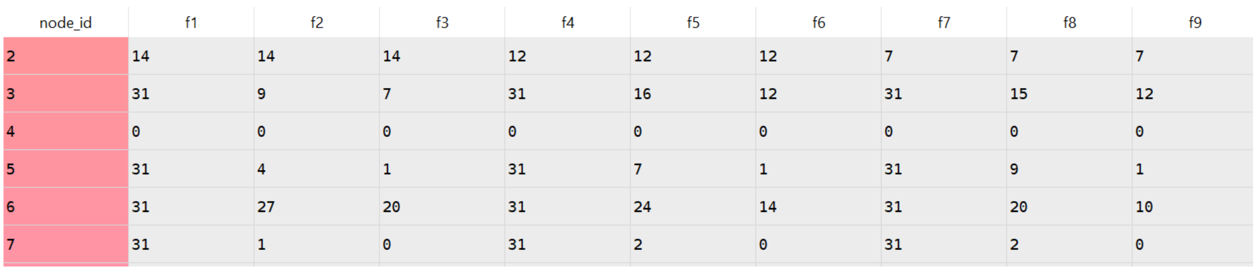


tf - DataFrame

Index	node1_id	node2_id	is_chat
0	6090515	2051181	0
1	4783857	5970009	0
2	6078716	2908655	0
3	4569158	87209	0
4	6214682	4317776	0
5	7143599	3647370	0
6	4995716	194326	0
7	480816	2766688	0
8	6715101	1246615	0
9	4525883	4341850	0
10	3056331	7963103	0
11	2785802	1434593	0
12	305537	1574993	0

2. User Features

- This dataset contains a numerical representation of all the actors in the network.
- There are exactly **13 features** representing each actor, with some of the features being of categorical nature and the rest as continuous values.
- Each record is a vector of features of a given node as shown below.

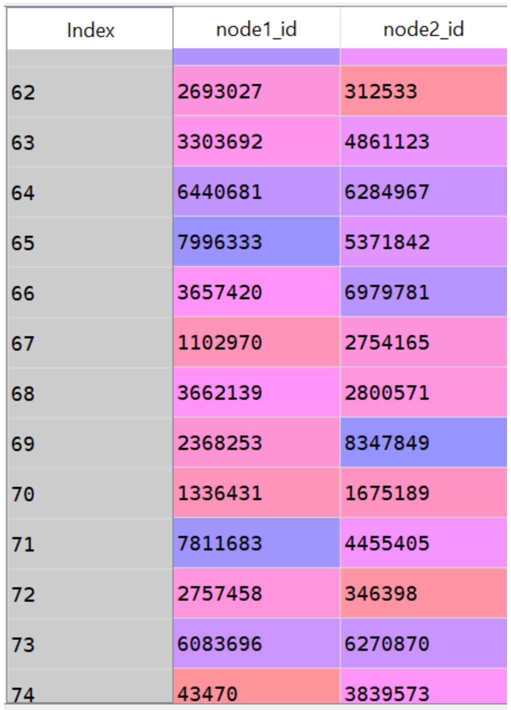


node_id	f1	f2	f3	f4	f5	f6	f7	f8	f9
2	14	14	14	12	12	12	7	7	7
3	31	9	7	31	16	12	31	15	12
4	0	0	0	0	0	0	0	0	0
5	31	4	1	31	7	1	31	9	1
6	31	27	20	31	24	14	31	20	10
7	31	1	0	31	2	0	31	2	0

3. Test Data

- The test data is similar to the training data, except for the absence of the *is_chat target variable.*

S



Index	node1_id	node2_id
62	2693027	312533
63	3303692	4861123
64	6440681	6284967
65	7996333	5371842
66	3657420	6979781
67	1102970	2754165
68	3662139	2800571
69	2368253	8347849
70	1336431	1675189
71	7811683	4455405
72	2757458	346398
73	6083696	6270870
74	43470	3839573

Social Network Analysis tools

The tools used along with their features are:

1. **Pandas**

   Used for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series and for storing the data in the form of dataframes.

2. **Scikit-learn**

   Framework for machine learning algorithms and data manipulation

3. **Lightgbm**

   LightGradientBoosting is a gradient boosting framework that uses tree based learning algorithm, where leaves in the tree as grown vertically instead of horizontally. This framework is used for prediction of the target variable.

4. **Networkx**

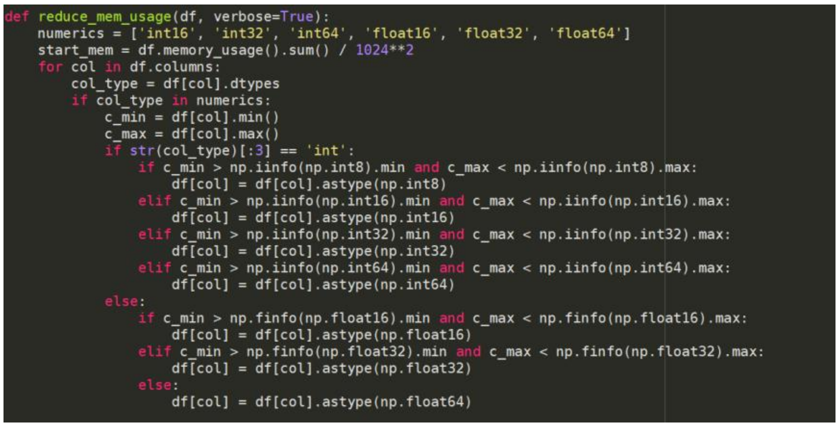
   NetworkX is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks. The social network graph is created from the nodes using NetworkX.

5. **Matplotlib**

   For plotting visualizations in the form of graphs.

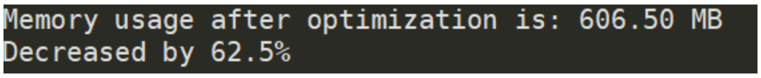
Implementation

- The dataset had sizes of *1.2GB and 600MB* respectively. In order to successfully load the dataframes using pandas, we had to utilize a memory reduction scheme. The code for which is given below:



```python
def reduce_mem_usage(df, verbose=True):
    numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
    start_mem = df.memory_usage().sum() / 1024**2
    for col in df.columns:
        col_type = df[col].dtypes
        if col_type in numerics:
            c_min = df[col].min()
            c_max = df[col].max()
            if str(col_type)[:3] == 'int':
                if c_min > np.iinfo(np.int8).min and c_max < np.iinfo(np.int8).max:
                    df[col] = df[col].astype(np.int8)
                elif c_min > np.iinfo(np.int16).min and c_max < np.iinfo(np.int16).max:
                    df[col] = df[col].astype(np.int16)
                elif c_min > np.iinfo(np.int32).min and c_max < np.iinfo(np.int32).max:
                    df[col] = df[col].astype(np.int32)
                elif c_min > np.iinfo(np.int64).min and c_max < np.iinfo(np.int64).max:
                    df[col] = df[col].astype(np.int64)
            else:
                if c_min > np.finfo(np.float16).min and c_max < np.finfo(np.float16).max:
                    df[col] = df[col].astype(np.float16)
                elif c_min > np.finfo(np.float32).min and c_max < np.finfo(np.float32).max:
                    df[col] = df[col].astype(np.float32)
                else:
                    df[col] = df[col].astype(np.float64)
```

- On loading the dataset using the above code we were able to obtain memory reduction of about **62.5%** on the train data as well as user features data.

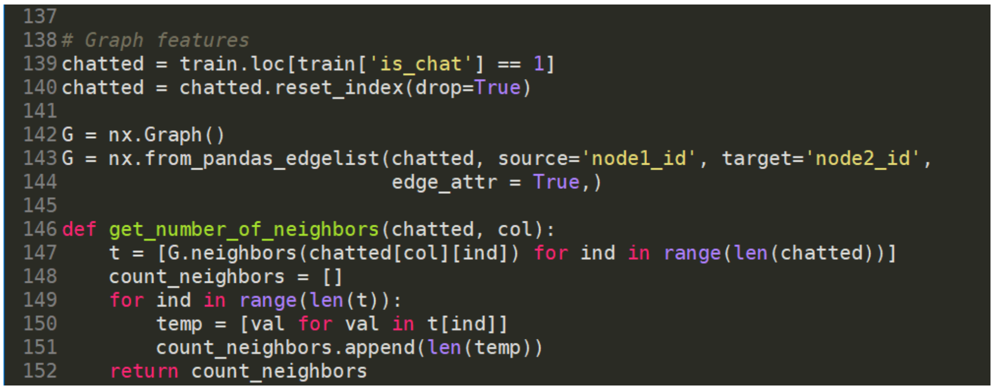


```
Memory usage after optimization is: 606.50 MB
Decreased by 62.5%
```

- Training data was **merged** with the User Features dataset, to get a dataset combining the features of both actors participating in the relation.
- Each of the 13 features were converted into **categorical features**.

- Further features were developed, namely:
  - **Total sum of chats** associated with each pair of nodes
  - **Mean of the chats** associated with both nodes.
  - **Count of chats** given by the division of sum by mean for each node.

- Using **NetworkX,** a undirected graph was generated using data from the train dataset. Each pair of nodes which having a **is_chat** relation between them, had an edge. Doing so we were further able to generate *2 more graph based features:*
    - **Count of neighbours** for each nodes in the network
    - **Common neighbours** between each pairs of nodes in the network



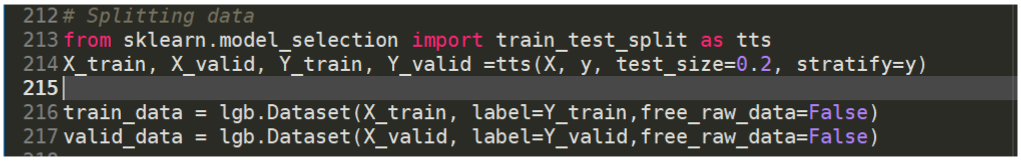
```python

# Graph features
chatted = train.loc[train['is_chat'] == 1]
chatted = chatted.reset_index(drop=True)

G = nx.Graph()
G = nx.from_pandas_edgelist(chatted, source='node1_id', target='node2_id',
                            edge_attr = True,)

def get_number_of_neighbors(chatted, col):
    t = [G.neighbors(chatted[col][ind]) for ind in range(len(chatted))]
    count_neighbors = []
    for ind in range(len(t)):
        temp = [val for val in t[ind]]
        count_neighbors.append(len(temp))
    return count_neighbors
```

- These set of features were selected and the merged dataset was now divided into **train and validation datasets** to estimate the accuracy of the predictions.



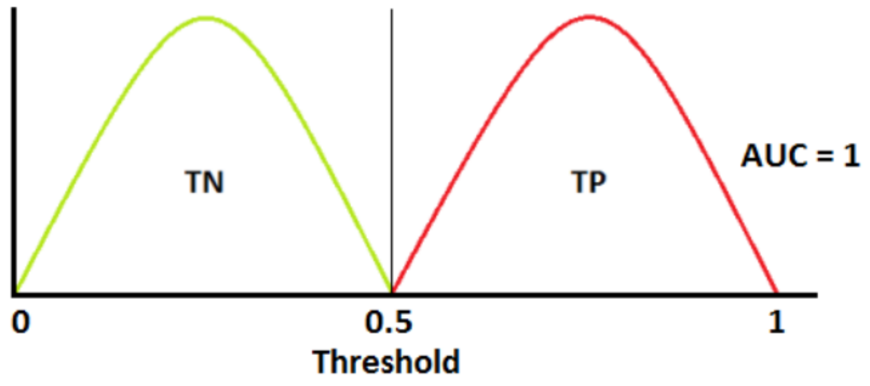
```python
# Splitting data
from sklearn.model_selection import train_test_split as tts
X_train, X_valid, Y_train, Y_valid =tts(X, y, test_size=0.2, stratify=y)

train_data = lgb.Dataset(X_train, label=Y_train,free_raw_data=False)
valid_data = lgb.Dataset(X_valid, label=Y_valid,free_raw_data=False)
```

AUC-ROC score metric for verifying predictions:

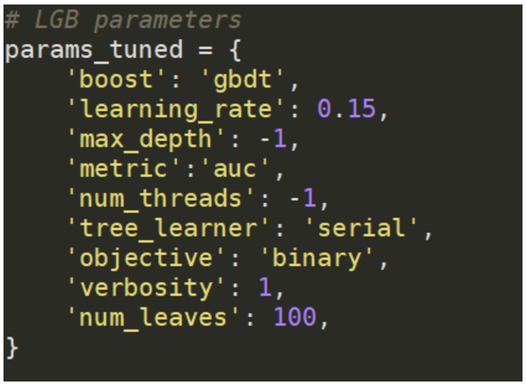
- The method for verifying predictions was selected as ***Area under the Curve, Receiver Operating Characteristics*** score.
- ROC is a probability curve and AUC represents degree or measure of separability. It tells how much model is capable of distinguishing between classes.
- The reason for selecting this metric is that almost 90% of the actors in the network have not chatted with the other. This results in a high direct accuracy score of 95%+ by predicting all the outputs as 0 *(not chatted).*
- Therefor by using ***AUC ROC score***, we take into consideration the Sensitivity and Specificity rates.

$$\text{TPR /Recall / Sensitivity} = \frac{TP}{TP + FN} \qquad \text{Specificity} = \frac{TN}{TN + FP}$$



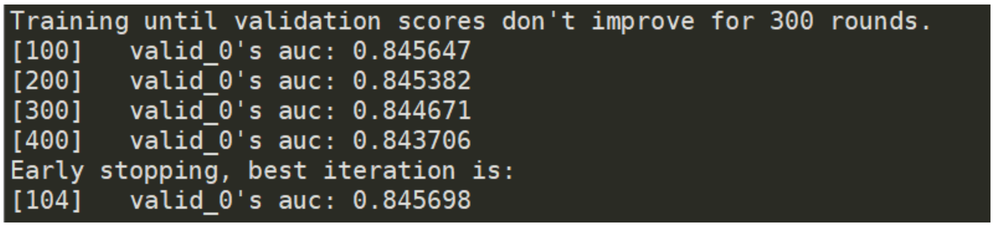
Light Gradient Boosting

- LightGradientBoosting was used to make the regression predictions of the target variable ***is_chat.***
- Parameters using for training are as follows:



```
# LGB parameters
params_tuned = {
    'boost': 'gbdt',
    'learning_rate': 0.15,
    'max_depth': -1,
    'metric':'auc',
    'num_threads': -1,
    'tree_learner': 'serial',
    'objective': 'binary',
    'verbosity': 1,
    'num_leaves': 100,
}
```

- The **AUC-ROC** score achieved on the validation dataset by setting *stopping rounds to 300* is as follows:



```
Training until validation scores don't improve for 300 rounds.
[100]	valid_0's auc: 0.845647
[200]	valid_0's auc: 0.845382
[300]	valid_0's auc: 0.844671
[400]	valid_0's auc: 0.843706
Early stopping, best iteration is:
[104]	valid_0's auc: 0.845698
```

- **AUC ROC score of 0.845** was obtained out a best of *0 to 1.*

References

- https://pdfs.semanticscholar.org/73d3/dd3cbbc6607bbc246f8bcc8ae101ac8489ce.pdf
- https://datahack.analyticsvidhya.com/contest/hikeathon/
- https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5
- https://medium.com/@pushkarmandot/https-medium-com-pushkarmandot-what-is-lightgbm-how-to-implement-it-how-to-fine-tune-the-parameters-60347819b7fc
- https://www.kaggle.com/gemartin/load-data-reduce-memory-usage
- https://networkx.github.io/
- https://www.python-course.eu/networkx.php
- https://scikit-learn.org/stable/