J Component Report
On
CSE3021: Social Information Networks

Link Prediction in Social Networks using Machine Learning

Submitted by:
17BCE0851 Mihir Kumar Singh
17BCE2238 Pratima Yadav
18BCE2473 Kismat Khatri

Under the Supervision
Of:

PROF Vijayasherly V

VELLORE INSTITUTE OF TECHNOLOGY,
VELLORE

INDEX

ABSTRACT

Social networks mainly focus on building social relations among users who share common interests, background, etc. People may/may not want to maximize their social influence. Based on the current network, we want to be able to predict the upcoming changes in the same.

We are using link prediction in this project to predict future links. The dataset will be gathered from Twitch – an online video streaming platform

1. INTRODUCTION, PROBLEM STATEMENT

Link prediction in social networks is an interesting topic that has been studied extensively in recent times. The problem can be stated as follows - given a snapshot of a dynamic social network, represented as a directed graph of nodes and edges (where nodes represent users and edges represent follower - followee relations) is it possible to predict which new relations (links) are likely to be formed in the future. Such predictions have a wide variety of applications like recommending 'friend suggestions" on Facebook, or "new topics" on Twitter, "interesting movies" and other such personalized recommendations on Netflix etc. We attempted to solve the link prediction problem for a dataset gathered from Twitch - an online video streaming platform.

2. LITERATURE SURVEY

The base paper that we chose for our problem is **The Link-Prediction Problem for Social Networks** by *Liben-Nowell* and *Kleinberg* [1]. This paper was a seminal work in this domain as it formalized the link prediction problem and established a fundamental concept that it is possible to infer future changes in a network based solely on features intrinsic to the network itself. The authors of this paper analyzed the co-authorship network of arXiv.org to predict whether two scientists are likely to co-author a paper together in the future. This paper firmly established the idea that there is a notion of proximity that can be used to predict future links and the network contains certain latent information in the form of its topology which can be leveraged to infer future interactions. We have used this concept in our work and tried to extract features using basic proximity measures of the network.

Link Prediction in Social Networks: the State-of-the-art, by *Wang Peng et al* [2], is one of the most cited survey papers which speaks about the idea behind link prediction, its applications and different ways to tackle it. The authors have given a general framework for solving the link prediction problem which involves similarity based prediction or learning based prediction. In similarity based prediction, every possible future link is assigned a score based on similarity between the nodes and higher score edges are likely to appear in future. In learning based approach, link prediction is seen as a binary classification problem.

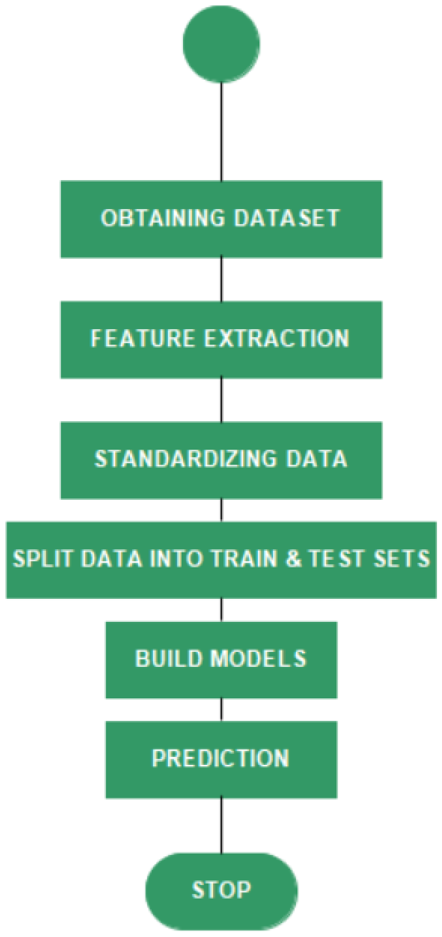
Furthermore, the paper also describes several metrics such as path based metrics or neighbour based metrics and their relevance in predicting links in a given graph. Overall the paper is the most comprehensive and gives direction to solve the link prediction problem. Based on the findings in this paper and the application of our research problem, we decided to use learning based prediction.

Another important paper that we considered was **Network Growth and Link Prediction Through an Empirical Lens** by *Liu, et al* [3]. This paper is relatively recent and less cited but it proves the importance of using machine learning models as opposed to purely mathematical and statistical distance metrics for link prediction. The authors of this paper implemented 18 link prediction algorithms each categorized as 'metric-based' (using a single similarity or proximity metric) or 'classification-based' (using machine learning classifiers with multiple metrics as input features). The authors evaluated these algorithms on large detailed network traces obtained from Facebook, Youtube and RenRen. They firmly established that SVM consistently outperforms all metric based methods across all three networks. Based on their findings, we chose to only implement machine learning classifiers instead of single similarity metrics.

The method proposed in **Transitive Node Similarity for Link Prediction in Social Networks with Positive and Negative Links** [4]**,** by *Panagiotis Symeonidis et al*, takes into account both local and global features of a graph. It states that the probability of existence of an edge directly depends on length of pathway between the nodes and similarity between neighbours of the two nodes. The authors define a proximity function based on Jaccard Distance using which node similarity matrix and transitive similarity matrix is constructed. The authors describe how link prediction can be done based on the transitive similarity and how this concept can be extended to signed networks as well.

3. PROPOSED SYSTEM DESIGN AND ANALYSIS

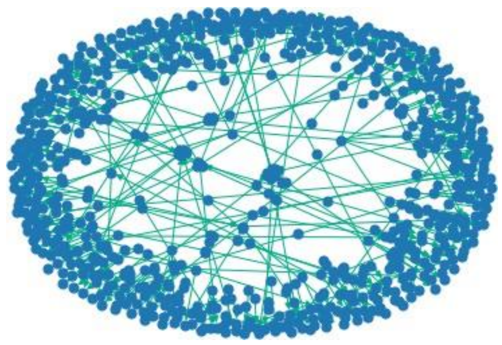
We are interested in solving the problem of personalized recommendations of streamers that a user can follow. This recommendation problem can be mapped to a binary classification problem wherein the two classes are - recommend or do not recommend with features extracted from the dataset. We propose solving this by generating link predictions and then using the predicted links to recommend streamers. Thus, we aim to train classification models using machine learning to predict whether a link exists between any given pair of nodes and use this prediction for recommendations.



4. METHODOLOGY

A] Dataset Description

Twitch is used widely by gamers to live-stream themselves while playing games. The nature of the platform is such that there are few popular gamers with many followers. We obtained the dataset from the Stanford Large Network Dataset Collection of more than 50 large network datasets from tens of thousands of nodes and edges to tens of millions of nodes and edges [5]. We chose the twitch dataset as there has not been much link prediction work done on this previously. The dataset was reasonably sized with 7126 nodes. Here is a visualization of a random sample of 500 edges from the dataset -



The total number of possible edges in the network is 50,772,750 from which 35,324 are present in the network. Using all the missing edges from the graph would highly skew the dataset so we randomly sampled 35,324 missing edges. While sampling these missing edges, we added a condition to only consider an edge as missing if the distance between the source and destination was more than 2 as closely connected users are likely to be mutual friends even if an edge does not already exist. This is to ensure the model is able to properly distinguish between present and absent edges, thus improving its performance. We labelled the edges that are present as 1 and the missing ones as 0. We used this presence or absence of an edge as the target class variable for prediction.

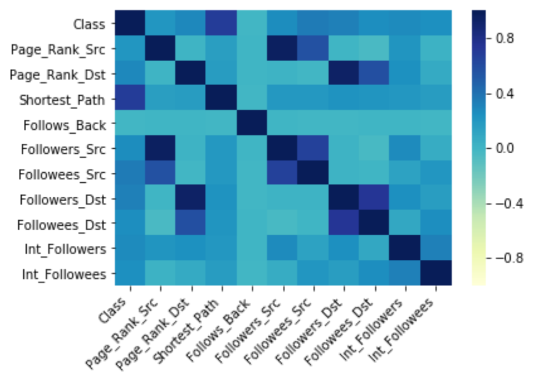
B] Feature Extraction

We then used the 70,648 edges to extract various features such as -

Page Rank : PageRank is the algorithm used by Google Search to rank websites in their search engine results. It works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more important websites are likely to receive more links from other websites [6]. Accordingly, we calculated the page ranks of both source and destination nodes of each edge and these formed two of the features we used.

1. **Shortest Path** : The shortest path between two nodes is simply the path with the least number of intermediate nodes. In our network, if a direct link already exists between two nodes, we first delete it and then calculate the shortest path between them. The intuition behind this feature is that nodes which are close to each other have shorter path lengths indicating that they are likely to be good recommendation candidates.
2. **Follows Back** : This feature simply indicates whether a reversely directed edge exists in the network for each existing edge, ie whether a user follows back one of his followees.
3. **Follower & Followee Counts** : These features are the number of followers and followees of source and destination nodes. The intuition here is that popular streamers have a large number of followers and are good choices for recommendation candidates.
4. **Inter Followers & Followee Counts** : These features are the number of common followers and followees between the source and destination nodes of an edge.

The following heatmap shows the feature correlation with the class variable -



C] Model Building

In this phase we execute the following steps:

1. **Standardisation of Data :** Standardisation is required to bring the data to the same scale. In case the data is not standardised, we get skewed results as one or more features might dominate the others. For example, features like number_of_followers have a wider distribution than page_rank because of which number_of_followers will dominate over page_rank. However, after standardisation, all values lie between 0 to 1 and will be given equal weightage while training a machine learning model.
2. **Split Data for Training and Testing :** In order to test the model on unseen data, it is necessary to split the data and this should be done randomly to avoid bias in the training or the testing phase. Random split ensures that the model is trained on edges which belong to both the classes (1 and 0). We have used 70% of the data for training and the remaining 30% for testing the model.
3. **Build Models :** For this classification problem, we have trained four models namely, Logistic Regression, Random Forest, Support Vector Machine and XGBoost. In this phase, we have used Grid Search on each model in order to estimate the best parameters. Using Grid Search with cross validation we train the model on several possible combinations of parameters and use the validation accuracy to determine the best parameters. We then use this best model to predict on the test data.

Machine Learning Models Used :

a. **Logistic Regression :** Logistic regression gives us the probability of a certain input features belonging to a certain class. It uses the logistic function to model the binary dependent variable.
b. **Random Forest :** Random Forest is a collection of decision trees. It operates on the principle that, a large number of uncorrelated trees will outperform a single model. Each tree predicts a class and the class with most predictions is chosen as the model's prediction.
c. **Support Vector Machine :** In SVM, the aim is to find a hyperplane that best divides the data points into 2 classes. Predictions are made based on the position of a certain data point with respect to the hyperplane.

   d. **XGBoost :** XGBoost makes the best use of available resources like memory and time. It implements gradient boosting decision trees which trains multiple models in a gradual, additive and sequential manner.

4. **Prediction :** Once the models are built, we test them using the 30% data. In this phase we feed the features as input and compare the model's prediction to the ground truth.

5. IMPLEMENTATION

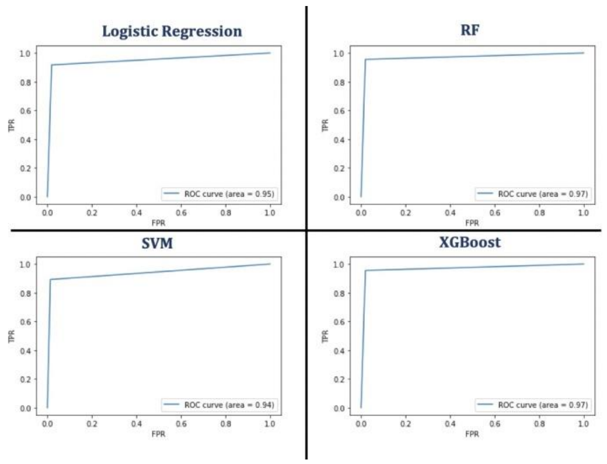
```
%matplotlib inline

import random

import pandas as pd
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
data = open('./data/musae_ENGB_edges.csv')
next(data)
g = nx.read_edgelist(data, create_using=nx.DiGraph(), delimiter=',', nodetype=int)
g = nx.convert_node_labels_to_integers(G=g, first_label=0)
GCOPY = g.copy()
print nx.info(g)
sampled_edges = random.sample(g.edges, 500)
sampled_g = nx.DiGraph()
sampled_g.add_edges_from(sampled_edges)
sampled_g = nx.convert_node_labels_to_integers(G=sampled_g, first_label=0)
subg = nx.spring_layout(sampled_g)
nx.draw(sampled_g, subg, node_color='#1F77B4', edge_color='#02AE87', width=1, edge_cmap='YlGnBu',
node_size=50)
import pickle
import random
import time
st = time.time()

edges = dict()
for edge in g.edges:
   edges[(edge[0], edge[1])] = 1

missing_edges = set([])
count = 0

while (len(missing_edges)<35324):
   if count % 50 == 0:
      print 'iteration: ', count, '\tlen of miss edges: ', len(missing_edges), '\tedges left: ', 1768149-
len(missing_edges)
   count += 1

   a=random.randint(0, 81305)
   b=random.randint(0, 81305)
   tmp = edges.get((a,b),-1)
   if tmp == -1 and a!=b:
        try:
            # adding points who less likely to be friends
            if nx.shortest_path_length(g,source=a,target=b) > 2:

                missing_edges.add((a,b))
            else:
                continue
        except:
                missing_edges.add((a,b))
    else:
        continue

end = time.time()
print '\n\ntime taken: ', end-st
pickle.dump(missing_edges,open('./data/twitch_missing_edges_final.p','wb'))
len(missing_edges)


import pickle
import copy
import warnings
warnings.filterwarnings('ignore')

import pandas as pd
import numpy as np
import networkx as nx
fp = open('./data/twitch_missing_edges_final.p', "rb")
missing_edges = pickle.load(fp)
type(missing_edges)
df_neg = pd.DataFrame(list(missing_edges), columns=['Source', 'Destination'])
print df_neg.shape
df_neg.head(1)
df_pos = pd.read_csv('./data/musae_ENGB_edges.csv')
df_pos = df_pos.rename(columns = {'from':'Source', 'to':'Destination'})
df_pos = df_pos.drop_duplicates()
print df_pos.shape
df_pos.head(1)
df_pos['Class'] = 1
df_pos.head(5)
df_neg['Class'] = 0
df_neg.head(5)
frames = [df_pos, df_neg]
df = pd.concat(frames)
print(df.shape)
df.head(5)
df.to_csv('./data/twitch_subset.csv')
df_copy = copy.deepcopy(df)
g=nx.from_pandas_edgelist(df[['Source','Destination']],source='Source',
target='Destination',create_using=nx.DiGraph())
print nx.info(g)
# Page Rank
pr = nx.pagerank(g)
df['Page_Rank_Src'] = df.Source.apply(lambda row: pr.get(row))
df['Page_Rank_Dst'] = df.Destination.apply(lambda row: pr.get(row))
# Shortest Path
def get_shortest_path(x, y):
    d = -1
    try:
        if g.has_edge(x, y):
            g.remove_edge(x, y)
            d = nx.shortest_path_length(g, source=x, target=y)
            g.add_edge(x, y)
        else:
            d = nx.shortest_path_length(g, source=x, target=y)
    except:
        d = -1
    return d

df['Shortest_Path'] = df.apply(lambda row: get_shortest_path(row['Source'], row['Destination']), axis = 1)
# Follows Back
def get_follows_back(x, y):
    return 1 if g.has_edge(y, x) else 0

df['Follows_Back'] = df.apply(lambda row: get_follows_back(row['Source'], row['Destination']), axis = 1)
# Follow Features
followers_src, followers_dst, followees_src, followees_dst, int_followers, int_followees = [], [], [], [], [], []

for i, r in df.iterrows():
    pre_src = set(g.predecessors(r['Source'])) if set(g.predecessors(r['Source'])) else set()
    suc_src = set(g.successors(r['Source'])) if set(g.successors(r['Source'])) else set()

    pre_dst = set(g.predecessors(r['Destination'])) if set(g.predecessors(r['Destination'])) else set()
    suc_dst = set(g.successors(r['Destination'])) if set(g.successors(r['Destination'])) else set()

    followers_src.append(len(pre_src))
    followees_src.append(len(suc_src))

    followers_dst.append(len(pre_dst))
    followees_dst.append(len(suc_dst))

    int_followers.append(len(pre_src.intersection(pre_dst)))
    int_followees.append(len(suc_src.intersection(suc_dst)))

df['Followers_Src'] = followers_src
df['Followees_Src'] = followees_src
df['Followers_Dst'] = followers_dst
df['Followees_Dst'] = followees_dst
df['Int_Followers'] = int_followers
df['Int_Followees'] = int_followees
df.head(5)
df.to_csv('./data/twitch_final_dataset.csv')
```

6. RESULTS AND EVALUATION

We observed very good accuracy for all the models, with Random Forest and XGBoost performing the best. The following table shows various classification metrics for each of the machine learning models.

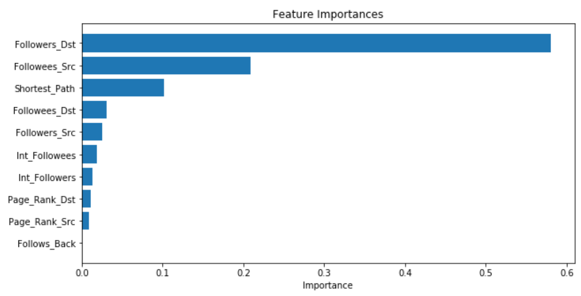
	Logistic Regression	Random Forest	Support Vector Machine	XGBoost
Precision	97.98%	98.05%	98.54%	98.03%
Recall	91.67%	95.53%	89.24%	95.51%
F1 Score	94.72%	96.77%	93.66%	96.75%
Accuracy	94.855	96.79%	93.91%	96.77%

Given below are the ROC-AUC curves for all the modes. The most ideal AUC is 1 and we observed AUC close to this in all the models.



7. CONCLUSION AND SCOPE FOR FUTURE WORK

We have presented a way of detecting whether a link will be formed in the future in a social network graph. Such a prediction has several applications like prediction of a disease outbreak, suggesting alternate route or recommendations on websites like Netflix/Amazon and so on.



It is evident from the above visualization that not all features have high contribution in the prediction. The contribution can depend on the application of link prediction and it is necessary to explore more features and focus on the most relevant ones.

8. REFERENCES

[1] D. Liben-Nowell and J. Kleinberg, “The link-prediction problem for social networks,” vol. 58, no. 7, pp. 1019–1031, May 2007 [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/asi.20591

[2] W. P. X. B. W. Y. Z. XiaoYu, “Link prediction in social networks： the state-of-the-art,” vol. 58, no. 1, pp. 1–38, 2015 [Online]. Available: http://lib.cqvip.com/qk/84009A/201501/663405989.html

[3] Q. Liu, S. Tang, X. Zhang, X. Zhao, B. Zhao, and H. Zheng, “Network Growth and Link Prediction Through an Empirical Lens,” 2016, pp. 1–15 [Online]. Available: http://dl.acm.org/citation.cfm?id=2987452

[4] P. Symeonidis, E. Tiakas, and Y. Manolopoulos, “Transitive node similarity for link prediction in social networks with positive and negative links,” 2010, pp. 183–190 [Online]. Available: http://dl.acm.org/citation.cfm?id=1864744

[5] J. Leskovec, “Stanford Large Network Dataset Collection.” [Online]. Available: http://snap.stanford.edu/data/index.html. [Accessed: 05-Dec-2019]

[6] “Page Rank Algorithm and Implementation.” 2017 [Online]. Available: https://www.geeksforgeeks.org/page-rank-algorithm-implementation/. [Accessed: 06-Dec-2019]