Recommendation System on Yelp Dataset

Haoyang Ling, You Wu

University of Michigan, School of Information

Objectives

This project explores Yelp Dataset (over 9GB), which includes data about Yelp's business, users, tips, and reviews to:

- Explore data mining in the network.
- Recommend friends with similar tastes.
- Lead individuals to explore new businesses based on friend recommendation.

Existing Work

- Measure similarities of two different objects like user profiles in social networks.
- Implement alternating least squares matrix factorization in Spark.
- Convert recommendation problems to classification problems in the deep ranking model.
- Use the K-nearest neighbors algorithm to add extra layers of character breakdown and user behaviors.
- Research on graph learning approaches like message propagation and random walk.

Data Collection & Preprocessing

Data Source

- Scraped from https://www.yelp.com/dataset.
- Analyze two JSON files about business reviews and user profiles.
- Include Over 1.2 million business entities.

Algorithm BFS in PySpark
input : Adjacency List to represent the graph G
output: k-hops

d ← 0
V_0 = **BAG-CREATE**(v_0)

while V_d *is not empty* **do**
  V_{d+1} ← **MERGE-BAG**(**NEIGHBORHOOD**(V_d))
  d ← d + 1
end

Methodology

User similarity

- Use the **Jaccard Similarity** of businesses reviewed by users to define the similarity.
- Accelerate computation with **Min-hash**.
- Number of hash functions: 100
- Threshold of Jaccard Similarity: 0.3

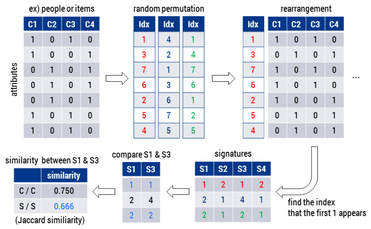

Figure: Min-hash Algorithm

Link Prediction

- **Negative Sampling** of edges for train and test.
- **Adamic–Adar Index** (baseline): calculate the sum of the inversed logarithmic degree centrality of the neighbors shared by the two nodes.
- Graph Convolution Neural Network (**GraphSage**): define the dot product as the similarity in two GCN layers with a hidden dimension of 32 and projection head.

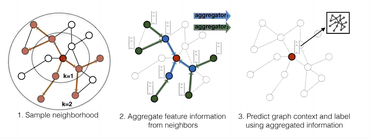

Figure: GraphSage

Popularity Prediction

- Define the number of fans as the popularity metric and explore underestimated users.
- Ridge Regression & Random Forest (max_depth=8) & (two-layer) MLP .
- Use all (18) columns in the dataset as attributes for prediction with **SVD** and **cross-validation**.
- The underestimation rate is defined as the ratio of the prediction to actual value.

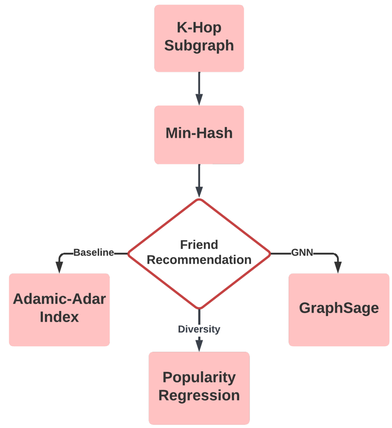

Figure: Project Workflow

Results

User similarity

- Limitations: Jaccard similarity is low. Many users with high similarity only review few businesses.

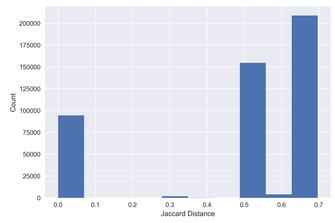

Figure: The Distribution of Jaccard Distance

Link Prediction

Methods	Train Accuracy	Test Accuracy
Adamic-Adar Index	**Baseline**: 0.311	
GraphSage (100 iter.)	0.603	0.600
GraphSage (2000 iter.)	0.692	0.673

Table: Model Performance in Link Prediction

Popularity Prediction

Methods	Train R-Squared	Test R-Squared
Ridge Regression	0.597	0.609
Random Forest	0.924	0.700
MLP	0.792	0.723

Table: R-Squared Value of Different Regression Methods

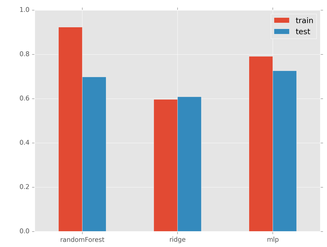

Figure: R-Squared Value of Different Regression Methods

Sample User Profile

- User: Gigi (4476724) who has 327 friends, 1184 funny comments, and 1567 useful comments.
- Jaccard Distance (0.686) has found one of Gigi's friends Mark (5495693) who has 465 friends, 3175 funny comments, and 5685 useful comments.
- Popularity regression (the underestimation ratio under 3) will recommend Nick (5236527), Tim (2568097), and Sue (4978186) with many useful votes and owned friends but fewer fans.

References

[1] Yelp dataset. accessed https://www.yelp.com/dataset/documentation/faq from 10.31.2022.

[2] Xiran Song, Jianxun Lian, Hong Huang, Mingqi Wu, Hai Jin, and Xing Xie. Friend recommendations with self-rescaling graph neural networks. In *KDD 2022*. ACM, August 2022.

[3] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation, 2019.

[4] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, New York, NY, USA, 2016.

[5] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. arXiv, 2017.

SCHOOL OF INFORMATION
UNIVERSITY OF MICHIGAN

SI671 Project: Friend Recommendations on Yelp Platform

Haoyang LING[1] You WU[1]

[1]University of Michigan, School of Information
hyfrankl@umich.edu, uvuuview@umich.edu

Abstract

With the rise of deep learning and machine learning, many retailers adopt recommendation systems to increase their competitive ability in the market.Yelp platform has published an extensive dataset about its user and business profiles (around 9 GB). Many researchers have explored the dataset, but few of them focus on friend recommendations with users. In this project, the k-hop sub-graph or the ego-net of one specific user will be analyzed to provide diverse recommendations.

First, the Min-Hash algorithm is implemented but few existing links are found with a low Jaccard distance. Next, we randomly drop the edges without destroying the connectivity of the network and randomly select negative edges from the network. We first measure each edge with the Adamic-Adar index and obtain the baseline accuracy of 0.311. Furthermore, we adapt GraphSage to this link prediction task and randomly choose different negative edges in each training epoch. The training and testing accuracy model reached around 68% after 2000 epochs compared with 60% in the first 100 epochs. Finally, we intend to provide diverse recommendations to the users, so we use the number of fans as the metric and try to predict it with other statistics with the help of the truncated single value decomposition (TruncatedSVD). With the criterion of R-Squared, ridge regression, random forest, and MLP respectively reached 0.60, 0.92, and 0.79 in the train and 0.61, 0.70, and 0.73 in the test. In sum, all the tasks are around the topic of friend recommendations. All the results meet the expectations.

1 Introduction

Recommendation system is one of the most popular topics recently and has various applications in daily life [11] that bring us great convenience. There are four mainstream methods for recommendation systems: content-based, collaborative, demographic, and hybrid filtering [11].

In this project, we intend to build a recommendation system from the Yelp Dataset (over 9GB), which includes data about Yelp's business, users, tips, and reviews [1]. This dataset has been explored before by some researchers mostly focusing on the analysis of reviews, while we emphasize the user data. Due to the data size, we will focus on the sub-graph of the data obtained from the breadth-first search in PySpark and the following tasks:

1) We will use the Jaccard similarity of the reviewed businesses to define the similarity of the users. It intends to find similar users through collaborative filtering. Since the data is large, we will implement the MinHash to approximate the Jaccard similarity.

2) The friend recommendation here is the link prediction task in the graph, so we will use the Adamic-Adar index as the baseline and GraphSage model to further improve it.

3) The previous recommendations are based on similarity, so we will also recommend the elite users to the users to increase the diversity of the recommendation. In this case, we use the number of fans as the metrics and other information to predict the popularity of the user with ridge regression, random forest, and MLP.

2 Objective

We intend to utilize the Yelp dataset and community detection techniques to help people make friends with those that have similar taste with them or follow them. Based on the results, we will implement a friend recommendation system and bring convenience for individuals to find people with common interests to explore new business.

3 Related Work

The topic of the proposal focuses on friend recommendation systems and social recommendations. One of the mainstream approaches is to measure similarities of two different objects in link prediction based on user profile [15]. The metrics for node-to-node similarities involve Jaccard similarity, Sim Rank [9], P-Rank [15], and Adamic Adar index [2].

However, there are other ways to approach recommendation systems. Xuelin Zeng et al. implemented alternating least squares matrix factorization in Spark [14]. Dianping. Lakshmi et al. used the user-item rating matrix to compute the user similarities [12]. Paul Covington et al. proposed a deep neural network in Youtube by converting recommendations to classification problems in the deep ranking model [5]. The k-nearest neighbors algorithm adds some extra layers of character breakdown and user behavior which introduced truer results [10].

Recent years have witnessed more applications of graph learning approaches in social recommendations [13]. Fan proposed to use message propagation and aggregate messages from neighborhoods [7]. Eksombatchai et al. adopt the random walk to implement recommendation systems [6]. Chen et al. consider bias offsets of users, treat the biases as vectors and fuse them into the process of learning user representations [4].

4 Methods and Approach

Based on the size of the dataset and survey of related work, we will employ big data analysis and neural networks with GPUs to perform our analysis. The main tools that we will use are MLlib in PySpark and PyTorch with the PyGeometric framework. To clarify the tasks in the project, we will break them down as follows:

1. Clean the dataset and extract useful information from the original data with a JSON file.
2. Prepare for the implementation of friend recommendations on the k-hop sub-graph (or egonet) extracted by breadth-first search (BFS).
3. Use MinHash to approximate the Jaccard Similarity of shops by users to define the user similarities.
4. Use the Adamic–Adar index to predict the user's relationship(or links) as a baseline.
5. Adapt the GraphSage to predict links between users.
6. Regress the number of fans for each user by linear regression, random forest, and MLP with the criterion of R-Squared.

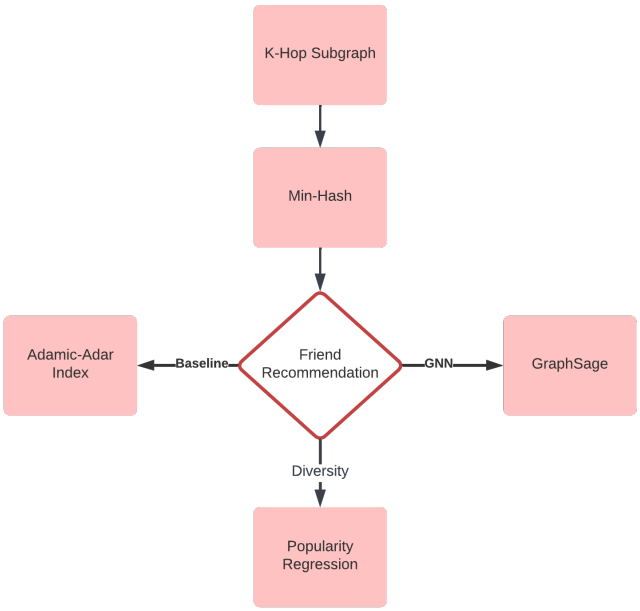

Figure 1: Project Workflow

4.1 Dataset

We mainly use the two JSON files related to business reviews and users' profiles. There are some important columns we use here. For business reviews,
1) user_id: the user id
2) business_id: the business id
are used to calculate the Jaccard similarity of the users.

For users' profiles,
1) user_id: the user id
2) review_count: count of the user's reviews
3) friends: list of user id of friends
4) useful: number of useful votes sent by the user
5) funny: number of funny votes sent by the user
6) cool: number of cool votes sent by the user
7) fans: number of fans the user has
8) average_stars: average rating of all reviews
and some other statistics. The "friends" field is used for the link prediction task. All columns are used in the regression task of the number of fans.

4.2 Data Pre-processing

We will use PySpark, a big data framework, to implement a breadth-first search algorithm shown below, which will be used to explore nodes in a graph and gain a sub-graph that is reasonable and feasible for us to deal with. The algorithm explores new nodes layer by layer with the most feasible representation of the graph information – adjacency list.

Algorithm 1 BFS in PySpark

input : Adjacency List to represent the graph G
output: k-hops

d ← 0
V_0 = **BAG-CREATE**(v_0)

while V_d *is not empty* **do**
  V_{d+1} ← **MERGE-BAG**(**NEIGHBORHOOD**(V_d))
  d ← d + 1
end

After sampling, we check the data obtained above and re-index the user id so that it is convenient for us to use the number instead of strings to index the user. It will speed up the query of the user and shrink the file size. Since we re-index the user id here, we remap the user id in other files and store the results in CSV files for later usage.

4.3 Jaccard Similarity with MinHash

In this section, we use the business reviewed by the user as the history data to compute the similarity between the users.

However, we face the problem that there are over 60000 users in the dataset. So, we need some fast approximation methods like the min-hash algorithm to speed it up. The expectation of the min-hash algorithm is exactly the Jaccard similarity because the shared item comes to be the min value of the hash function is the Jaccard similarity in expectation. The expected MinHash value is closely related to Jaccard similarity, while MinHash is faster to calculate. There are 67095 users in the dataset, and we use 100 hash functions to reduce the error of MinHash where the error is O($\frac{1}{\sqrt{k}}$) and k is the number of the hash function.

Our implementation is accomplished by HashingTF and MinHashLSH in pyspark.ml.feature. We use the threshold of Jaccard similarity as 0.3 and see the distribution of the Jaccard similarity.

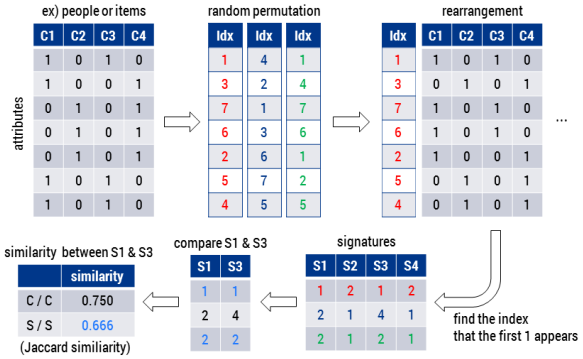

Figure 2: Principles of MinHash [3]

Listing 1: **minhash.py**

```
model = Pipeline(stages=[
        HashingTF(inputCol="bus_id",
            outputCol="vectors"),
        MinHashLSH(inputCol="vectors",
            outputCol="lsh", numHashTables=100)
    ]).fit(val)

hashed_val = model.transform(val)
```

4.4 Link Prediction

Negative Sampling: Link prediction needs to include the positive edges and negative edges in the testing set to avoid overfitting and even model collapse. By incorporating non-existent links as negative samples, the model is forced to learn the distinctions between positive and negative edges. Furthermore, using randomly selected negative samples can enhance the model's prediction of unseen data.

So, we first sample positive edges in the network and sample negative edges not in the network with the same size to be the goal of the prediction and randomly sample negative edges during the training process.

Adamic-Adar Index: it is a measure to predict links in the social network by calculating the sum of the inversed logarithmic degree centrality of the neighbors shared by the two nodes.

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

Criterion: for the Adamic-Adar-index-based link prediction, we see the accuracy of the edges with higher Adamic-Adar index and the threshold is to include the number of edges that are similar to the number of positive edges in this setting.

GraphSage [8]: it is a stochastic graph convolution neural network. It is very useful in massive and dynamic networks. It uses the idea of message passing to aggregate the information from randomly sampled neighbors. The sampling makes the model more powerful in extracting abstract information from the network shown in 3. And, to avoid the over-fitting, I insert the negative edges in the training process, so that the model will be degraded to output 1 only but instead it will try to distinguish the positive edges and negative edges. Furthermore, the negative edges are randomly selected from the network, which can improve the generality of the model.

Besides, I add the projection head (a two-layer MLP) after the graph convolution network because the output of the GraphSage will not immediately embed the knowledge of the link prediction. Rather, it includes more information about the node itself. So, I use MLP to project the node representation and do the dot product. However, as the dot product is less

meaningful to represent the edge existence, I use its sigmoid value to show the probability of the edge existence. For training, the loss is defined as the binary entropy loss.

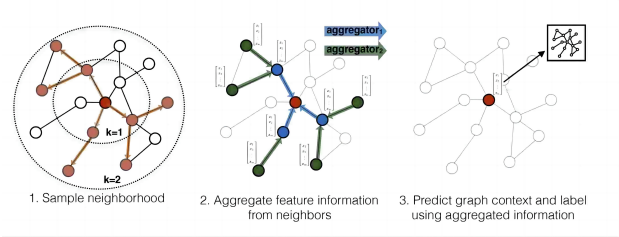

Figure 3: Principles of GraphSage [8]

Criterion: the feature of every user profile is fed into the model after standardization. We define similarity as the sigmoid value of the dot products of the node representation. Since it is supervised learning, we only need to calculate the accuracy of the testing edges with both positive edges and negative edges selected before the training.

Listing 2: **GraphSage.py**

```
from torch_geometric.nn import GraphSAGE
import torch.nn as nn


class MyModel(nn.Module):
    def __init__(self, in_dims, hidden_dims,
         num_layers, out_dims) -> None:
          super().__init__()
        self.gcn =  GraphSAGE(in_channels = in_dims,
            hidden_channels=hidden_dims, num_layers=
            num_layers,
                        out_channels=out_dims, project
                            =True, normalize=True)
        self.mlp = nn.Sequential(nn.Linear(out_dims,
            out_dims), nn.ReLU(), nn.Linear(out_dims,
             out_dims))
        self.init_weights(self.mlp)

    def init_weights(self, layers):
        for layer in layers:
            if  isinstance(layer, nn.Linear):
                nn.init.xavier_normal_(layer.weight.
                    data)

    def forward(self, data):
        embedding = self.mlp(self.gcn(x=data.x,
            edge_index=data.edge_index))
        return embedding


in_dims = data.x.shape[1]
model = MyModel(in_dims, 64, 2, 8).to(device)
```

4.5 Popularity Prediction

In this section, we first use the correlation matrix to find the most important columns. Then, we use the cross-validation with the metric R-Squared. We try ridge regression, random forest, and MLP.

Ridge Regression: we use the truncated SVD to reduce the dimensions as there are many zeros in the data. Then, we put the output to the ridge regression. Here we ignore the standardization due to the truncated SVD.

Random Forest: we use the columns selected from the correlation but not using SVD to reduce the dimensionality. Instead, we use the max depth of 8 to restrict it.

MLP: we use a two-layer multi-layer perception (32, 16) with the initial learning rate 1e-2. The data is first standardized and then sent to truncated SVD with the first 8 main components. In the end, the output is fed to the MLP. We use standardization here because the neural network is more sensitive to standardization. I will show the code of MLP here.

Unde-estimation Rate: the underestimation rate is defined as the ratio of the prediction to the actual value. The prediction value is based on the model we trained in the popularity prediction part. It intends to provide diverse recommendations to users and try to solve the cold start problem. We filter the user with an under-estimation rate below or about 3 and recommend it to other users. This method needs to be proved by A/B testing later (not mentioned in the project).

Listing 3: **regression.py**

```
regressor = MLPRegressor((32, 16), learning_rate_init
    =1e-2)
xScalar = StandardScaler()
svd = TruncatedSVD(n_components=8)
pipe = Pipeline([('scaler', xScalar), ('svd', svd), ('
    regression', regressor)])

scores_mlp = cross_validate(
    estimator=pipe,
    X=X, y=y, cv=5,
    scoring="r2",
    return_train_score=True,
    return_estimator=True,
)
print(pd.DataFrame(scores_mlp))
```

5 Experiments and Results

5.1 Jaccard Similarity with MinHash

In this case, we visualize the distribution of the **Jaccard Distance** which is one minus the Jaccard similarity. We only include the edge with the threshold 0.7 to the Jaccard Distance.

We can observe that the Jaccard distance of each two users is mostly larger than 0.5. Only a fewer of them are very similar to each other. And I have found that all of them review one business only, so there are some limitations to using this method. Therefore, it is not wise to use the Jaccard similarity to implement the KNN algorithm.

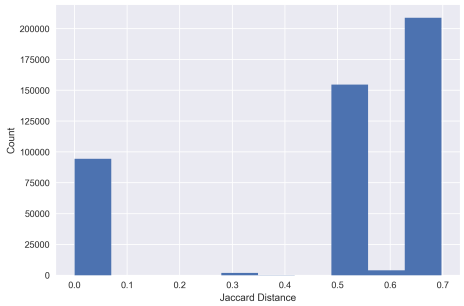

Figure 4: The Distribution of Jaccard Distance

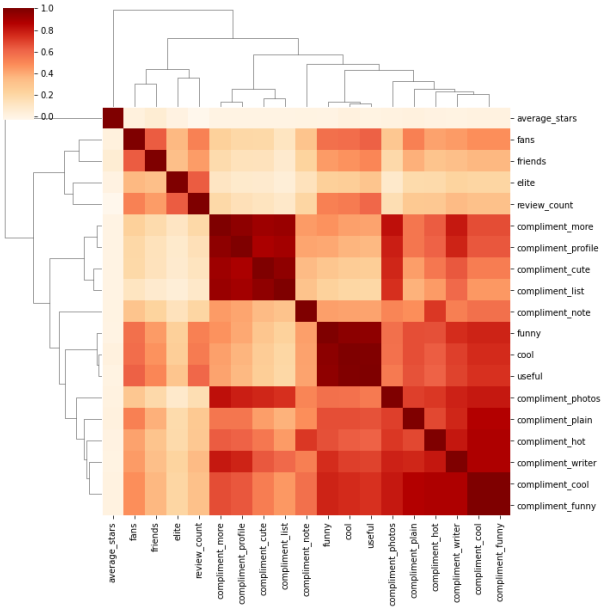

Figure 5: Clustered Heatmap of Correlation

5.2 Link Prediction

Here we use two methods to do link predictions. We first sample positive and negative edges from the network. Next, we use unsupervised learning to predict with the Adamic-Adar index and use supervised learning with GraphSage where we define the similarity as the sigmoid value of the dot products of the node representation. The results are shown in 1.

Methods	Train Accuracy	Test Accuracy
Adamic-Adar Index	**Baseline**: 0.311	
GraphSage (100 iter.)	0.603	0.600
GraphSage (2000 iter.)	0.692	0.673

Table 1: Model Performance in Link Prediction

We can find that the baseline provides pretty acceptable results from the Adamic-Adar index. Moreover, when we use GraphSage to explore the relationship in the network, the model reached 0.692 in the train and 0.673 in the test compared with the first 100 iterations and the baseline. Therefore, we can conclude that the model is working as we see noticeable improvements. It will be very promising if we tune the parameters with more trials.

5.3 Regression

To select features for ridge regression, we first draw a heatmap to show the correlation between columns, except for the number of fans features. We could see that compliment-related columns tend to correlate with each other. So we could choose only one or two of them for further analysis.

According to training and testing results in the following table and chart, we could see that ridge regression has the poorest performance; random forest performs extremely well on training, but not so well on testing; MLP has the most stable performance for both training and testing. To improve these models, for ridge regression, we could do feature engineering and create our own features instead of using original columns. As for random forests, it is obvious that this model overfits, so in future analysis, we could reduce the depth of trees to reduce the complexity. As for MLP, this is the most stable version currently, so we may use it for production.

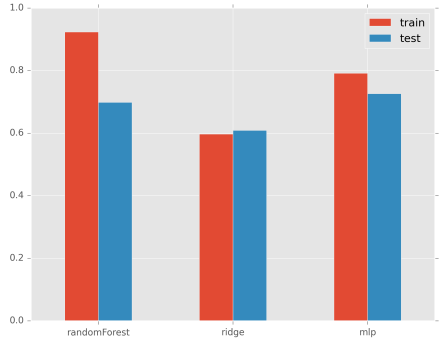

Figure 6: Regression Performance of Different Models

Methods	Train	Test
Ridge	0.597	0.609
RandomForest	0.924	0.699
MLP	0.792	0.727

Table 2: R-Squared Value of Different Regression Methods

5.4 Sample User Profile

To help reader gain a clearer impression for our work, we use a sample user and shows his statistics. His name is Gigi, whose user ID is 4476724 and has 327 friends on Yelp. 1184 of his comments are marked as funny comments, and 1567 of them are useful.

According to our methodology, he has a Jaccard similarity of 0.314 with one of his friends Mark, whose user ID is 5495693 and has 465 friends. 3175 of his comments are marked as funny comments, and 5685 of them are useful.

Then, using our link prediction and regression methods, we would recommend Nick (5236527), Tim (2568097), and Sue (4978186) who also contribute lots of useful votes and owned friends but have fewer fans.

6 Conclusion

In this project, we explore Yelp Dataset (over 9GB), which includes data about Yelp's business, users, tips, and reviews, to explore data mining in the network, recommend friends with similar tastes, and lead individuals to explore new businesses based on friend recommendations. After studying existing work on this topic, which includes similarity measurement, classification problems, clustering, and graph learning approaches, we decide to use a subgraph as our research object due to the size of the original user network. To obtain the subgraph, we use BFS in PySpark.

Then, we computed user similarity using Jaccard similarity and accelerated the computation by min-hash. Results show that most linked users have Jaccard similarity smaller than 0.5, while a few of them show high similarity. After that, we use Adamic-Adar Index and GraphSage to predict links, which means friendship between users. The former method serves as a baseline and the latter one serves as our main model. We found that the baseline has a baseline accuracy of 31.1%, while GraphSage, no matter 100 iterations or 2000 iterations, has a stable performance of more than 60%, and the performance slightly increases as iteration increases. Finally, as for popularity regression, we used three different models, ridge regression, random forest, and MLP, and compare their R-squared values. Ridge regression has the poorest performance; random forest performs well only on training; MLP has the most stable performance for both datasets.

7 Future Work

Due to time limits, our work still has room for improvement, which will be completed if time allows. In future work, we could try other techniques for friend recommendation, for instance, clustering user communities based on KNN and using the more modern graph representation learning model.

Also, for production, we need to do A/B testing before releasing the recommendation system. To do this, we could record users' active time on Yelp every day for two user groups, one group use our recommendation system while another group serves as a control group. If the former group's average user active time is higher than the latter, our product could be proven to be useful. However, it should be kept in mind that all other factors, like age, gender, etc, in these two user groups should be as similar as possible.

References

[1] Yelp dataset. accessed https://www.yelp.com/dataset/documentation/faq from 10.31.2022.

[2] Lada A Adamic and Eytan Adar. Friends and neighbors on the web. *Social networks*, 25(3):211–230, 2003.

[3] Park ChangUk. Minhash. Sep 2013.

[4] Xin Xin Xianfeng Liang Xiangnan He Chen, Jiajia and Jun Liu. Gdsrec: Graph-based decentralizedcollaborative filtering for socialrecommendation. *IEEE Transactions on Knowledge and Data Engineering*, 2022.

[5] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, New York, NY, USA, 2016.

[6] Chantat Eksombatchai, Pranav Jindal, Zitao Liu, Yuchen Liu, Rahul Sharma, Charles Sugnet, Mark Ulrich, and Jure Leskovec. Pixie: A system for recommending 3+ billion items to 200+ million users in real-time. *Proceedings of the 2018 World Wide Web Conference*, 2018.

[7] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation, 2019.

[8] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. arXiv, 2017.

[9] Glen Jeh and Jennifer Widom. Simrank: a measure of structural-context similarity. *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2002.

[10] Salony Mewara and Sneha Goyal. A friend recommendation system using semantic based knn algorithm. arXiv preprint, September 2021.

[11] Marwa Mohamed, Mohamed Khafagy, and Mohamed Ibrahim. Recommender systems challenges and solutions survey. 02 2019.

[12] Lakshmi Tharun Ponnam, Sreenivasa Punyasamudram, Siva Nallagulla, and Srikanth Yellamati. Movie recommender system using item based collaborative filtering technique. pages 1–5, 02 2016.

[13] Shoujin Wang, Liang Hu, Yan Wang, Xiangnan He, Quan Z. Sheng, Mehmet Orgun, Longbing Cao, Nan Wang, Francesco Ricci, and Philip S. Yu. Graph learning approaches to recommender systems: A review, 2020.

[14] Xuelin Zeng, Bin Wu, Jing Shi, Chang Liu, and Qian Guo. Parallelization of latent group model for group recommendation algorithm. pages 80–89, 06 2016.

[15] Peixiang Zhao, Jiawei Han, and Yizhou Sun. P-rank: A comprehensive structural similarity measure over information networks. pages 553–562, 01 2009.