MIDDLE EAST TECHNICAL UNIVERSITY

Department Of Computer Engineering

CENG 300

Summer Practice Report

METU Data Mining Research Group
Start Date: 06.07.2020 End Date: 05.08.2020
Total Working Dates: 20

October 11, 2020

Student:
Onat ÖZDEMİR

Supervisors:
Prof.Dr. Pınar KARAGÖZ

Prof.Dr. İsmail Hakkı TOROSLU

Contents

1 Introduction

I have done my summer internship at METU Data Mining Research Group under the supervision of Prof.Dr. Pınar KARAGÖZ and Prof.Dr. İsmail Hakkı TOROSLU. The main task I have worked on was describing trust between customers and integrating the trust scores calculated by TACoRec[1] to a collaborative filtering algorithm. Also, towards to end of my internship, I implemented an additional trust-based recommender system based on a new trust metric.

2 Project

During the internship, I implemented two trust-based recommenders with different weighting methods:

1. Eigentrust Weighted Recommender
2. Inverse Distance Weighted Recommender

The details of these two recommenders can be found in section 2.4 and 2.5, respectively.

2.1 Analysis Phase

The variety and number of products are increasing day by day, which creates the problem of recommending the most appropriate products for users. One of the main approaches used in the design of recommender systems is "collaborative filtering". The approach uses prior behaviours of customers such as rating profiles, product preferences, etc. to generate recommendations. Collaborative filtering methods can be classified according to which factor they prioritize while making suggestions. In this project, we focused on "trust-based collaborative filtering" which generates recommendations considering the trust between users.

The definition and calculation of trust may differ in many sources and researches. For instance, [2] approaches the issue from the probabilistic aspect and calculates trust through successful and unsuccessful transactions, the trust metric used in [7] is based on "Pearson Correlation Similarity" between users while [3] stress the value of providing ratings and argued that users

who give more rates are more trustworthy, even if they don't rate similarly. Such recommendation systems aim to calculate trust scores from behaviours of customers (e.g. ratings) to make good recommendations in the absence of an already existing trust network.

On the other hand, there are also approaches[5] that are developed to operate on datasets containing explicit trust scores (e.g. Epinions [8]) and make suggestions by using these scores directly or by combining them with additional features such as similarity, product, or user attributes (especially in hybrid recommenders), etc.

During the internship, due to lack of explicit trust information in most systems, we mainly focused on developing accurate trust-based recommender systems working on datasets with no explicit trust information such as Stockmount, Amazon Food Review, etc. Some of these datasets (e.g. Stockmount) contain implicit ratings while the rest of them have explicit ratings given by customers.

For the "Eigentrust Weighted Recommender", we were able to use "community detection" and "Eigentrust calculation" modules provided by TACoRec. The part I worked on was integrating the "Eigentrust scores" to the collaborative filtering algorithm.

For the "Inverse Distance Weighted Recommender", I experienced the difficulty of inferring trust from implicit customer behaviours which was I believe the most meaningful and challenging part of the internship.

2.2 Design Phase

2.2.1 For Eigentrust Weighted Recommender

As preliminary information, the Neo4j database we tested the recommender on initially contained customer, product, and transaction records. After using "community detection" and "Eigentrust" modules provided by TACoRec, disjoint customer communities and Eigentrust scores between the customers belong to the same community were added to the database.

Eigentrust represents how strongly connected the customers are to their communities (for the detailed information, please check section 2.4.1) and stored as a property of the relationship between the customer and his/her community in the Neo4j database. We can draw two conclusions from this information;

1. Since the Eigentrust scores are only calculated between the members of a community while making a recommendation to a customer, only the preferences of people in the same community as that customer should be taken into account.

2. Although the opinions of stereotype customers (i.e. customers with higher Eigentrust score) are important, we should also consider the similarity between the target user and the users who make suggestions to keep recommendations personalized.

Based on these conclusions, I designed two modules: Recommender and Filterer. The Recommender is responsible for writing the recommendations generated by the Filterer module to the database while the remaining weight of the project is carried by the Filterer: getting transaction and Eigentrust records from the database via Neo4j driver, measuring similarities between users based on transaction records, calculating recommendation coefficients for each product and generating recommendations based on these coefficients.

For the detailed information and the recommender structure, please check section 2.4 and Figure 1.

2.2.2 For Inverse Distance Weighted Recommender

"Inverse Distance Weighting" method is inspired by an article[4] that suggests calculating the trust scores between customer pairs by the reciprocal of the shortest distance between them on a trust network. Since I worked on datasets with no explicit trust information, I decided to use transaction records to build the trust network.

So with this approach, In addition to Filterer and Recommender, I implemented the Graph Module which is responsible for building the graph according to the preferred method (unweighted or euclidean distance weighted) from the transaction records and calculating trust scores by the reciprocal of the shortest distances between customers in the graph.

For the detailed information and the recommender structure, please check section 2.5 and Figure 2.

2.3 Implementation Phase

Since there are two different implementations, I have divided the implementation details of the recommenders into two subsections: section 2.4 and 2.5.

2.4 Eigentrust Weighted Trust Based Recommender

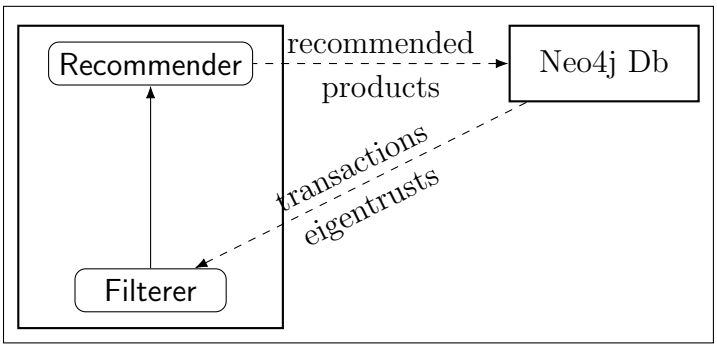

Figure 1: Recommender Structure

2.4.1 About Eigentrust

Eigentrust[2] is a reputation calculation algorithm based on the number of positive and negative transactions between customers and mainly designed for peer-to-peer networks. In our case, Eigentrust represents how strongly connected the customers are to their communities. Eigentrust scores are calculated by using the Eigentrust module provided by TACoRec and stored in the Neo4j database as a property of the relationship between a customer and his/her community.

Problem encountered with Eigentrust: Especially for the customers belong to communities with small size and low densities, Eigentrust scores stored in the database were either very small or equal to zero (check Figure 7). Most of the customers with zero Eigentrust were eliminated after filtering the network from customers with a small number of products. Unfortunately, even the filtering couldn't significantly improve the situation. Distribution of the Eigentrust scores before and after the filtering can be found in 5.1

2.4.2 Filterer Module

The Filterer Module is responsible for;

- Getting transaction/Eigentrust records for each community from the Neo4j database via Neo4j driver
- Calculating cosine similarities between customers
- Calculating weights
- If the dataset consists of implicit ratings calculating recommendation coefficients otherwise making predictions for products
- Selecting k products with the highest coefficients/predictions to recommend for each customer

Pseudocode for the Filterer

```
foreach community i do
    get transactions and eigentrusts of the customers belong to the i;
    construct customers versus products matrix using transactions;
    construct trust matrix using the eigentrusts;
    construct similarity matrix based on cosine similarity between
     customers belonged to the i;
    construct weight matrix using the trust and similarity matrices;
    for customer c belong to i do
        foreach product p purchased by neighbours do
            if c purchased p then
                continue;
            end
            else
                calculate recommendation coefficient using the weight
                 and customers versus products matrices for p;
            end
        end
        recommend k products with the highest recommendation
         coefficients;
    end
end
```

2.4.2.1 Calculating Weights

$$w_{c_{target}}(c_2) = \frac{2 * sim(c_{target}, c_2) * trust(c_2)}{sim(c_{target}, c_2) + trust(c_2)}$$

where $sim(c_{target}, c_2)$ represents "cosine similarity" between customers and $trust(c_2)$ represents eigentrust belonged to c_2.

2.4.2.2 Calculating Recommendation Coefficients

$$RC(i) = \frac{\sum_{c \in C} w_{c_{target}}(c) * b_c}{\sum_{c \in C} w_{c_{target}}(c)}$$

where $RC(i)$ represents the recommendation coefficient calculated for product i and b_c is a boolean value which indicates whether the product was purchased by the customer c or not.

2.4.2.3 Making Predictions

$$p(i) = \frac{\sum_{c \in C} w_{c_{target}}(c) * r_c}{\sum_{c \in C} w_{c_{target}}(c)}$$

where $p(i)$ represents the prediction for product i and r_c represents rating given by customer c for product i. C customer set only contains the customers who purchased product i.

Important remark: C customer set used in functions given above consists only of customers belonged to the same community with c_{target} while there is no such restriction in 2.5.

2.4.3 Recommender Module

Recommender Module is responsible for getting the recommendation list that contains ids of the customers and corresponding recommended products from the Filterer module and writing these recommendations to the Neo4j database as a relationship between the customer and the recommended product using Neo4j driver.

2.5 Inverse Distance Weighted Trust Based Recommender

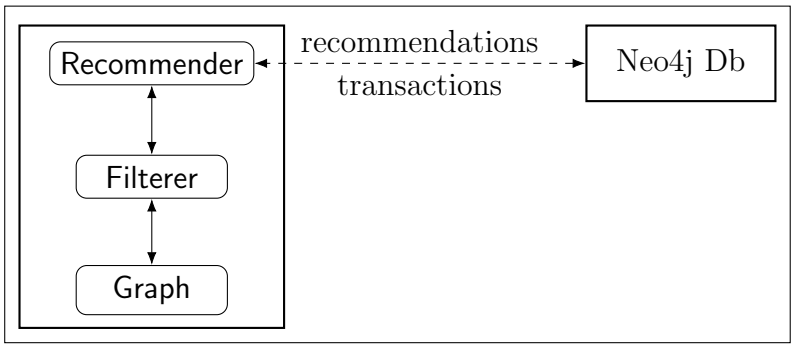

Figure 2: Recommender Structure

Inverse Distance Weighted Trust Based Recommender consists of three modules:

2.5.1 Graph Module

Graph Module is responsible for three tasks:

1. Constructing "adjacency matrix" from "customers versus products matrix" provided by the Filterer
2. Constructing "distance matrix" by performing Dijkstra's algorithm on the "adjacency matrix"
3. Constructing "trust matrix" by taking the reciprocal of the "distance matrix"

2.5.1.1 Constructing Adjacency Matrix and Distance Matrix

Since the recommender is tested on both the datasets with implicit ratings and explicit ratings, to construct the "adjacency matrix" from customer versus products table, I propose two methods:

Proposed Method 1: Unweighted Graph
In this method, the "adjacency matrix" is constructed based on whether customers have a common product or not. In other words, edge between two customers can exist if and only if the intersection of the set of products they purchased is not the empty set. This method is proposed for especially the datasets with **implicit ratings**.

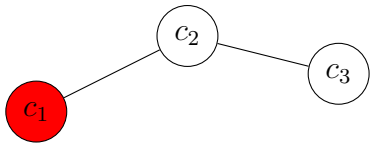

Figure 3: c_1 and c_2 have at least one common product while c_1 and c_3 do not have a common product

Problem encountered with Proposed Method 1: During the tests performed on the Movielens100k dataset, I observed that although the dataset is sparse, the maximum distance between two customers was calculated as 2. In other words, the graph was kind of a Small-world Network. Since the distances were distributed in such a small range, trust values calculated with this method were not so meaningful. With this observation, I decided to propose another method and use the "Unweighted Graph" method in extremely sparse datasets.

Proposed Method 2: Euclidean Distance Weighted Graph
In this method, the "adjacency matrix" is constructed based on the "euclidean distances"1 between customers. This method is proposed for especially the datasets with **explicit ratings**.

$$adj[c_1]][c_2] = \frac{\sqrt{\sum_{i \in I_1 \cap I_2} (r1_i - r2_i)^2}}{|I_1 \cap I_2|} \tag{1}$$

where $r1_i$ and $r2_i$ represents ratings given by c_1 and c_2 for product i. Unlike the commonly used "euclidean distance" calculation, in this method, only ratings given to common products are included in the calculation.

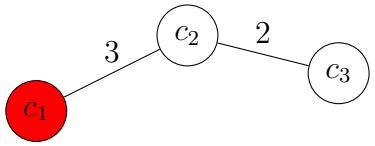

Figure 4: euclidean distance between c_1 and c_2 equals to 3, and c_1 and c_3 do not have a common product

Dijkstra's Algorithm
To construct the "distance matrix", the graph module uses Dijkstra's Algorithm which takes the adjacency matrix as a parameter and returns the distance matrix.

2.5.1.2 Trust Calculation

After calculating the shortest distance between each pair of customers using "Dijkstra's Algorithm", to calculate the trust scores between customers Graph module uses

$$T(c_1, c_2) = \begin{cases} \frac{1}{d(c_1,c_2)} & d(c_1, c_2) \neq np.inf \\ 0 & d(c_1, c_2) = np.inf \end{cases}$$

function where $d(c_1, c_2)$ represents the shortest distance between the $customer_1$ and $customer_2$. If $d(c_1, c_2)$ equals $np.inf$ that means either there is no path connecting the customers or the shortest distance between the customers exceeds the distance limit specified in the config file.

A benefit of the method: Especially for excessively sparse datasets, recommenders using euclidean distance-based similarity fails since they cannot calculate a similarity score for the customer pairs with no common products. Since the "Dijkstra's Algorithm" propagates weights even for the customer pairs with no common products, we are able to calculate trust scores between them.

2.5.2 Filterer Module

The Filterer Module is responsible for;

- Calculating cosine similarities between customers (for Proposed Method 1)

- Calculating weights
- If the dataset consists of implicit ratings calculating recommendation coefficients otherwise making predictions for products
- Selecting k products with the highest coefficients/predictions to recommend for each customer

Pseudocode for the Filterer

```
construct customers versus products matrix using transactions
 provided by the Recommender;
create a Graph object g by giving customers versus products matrix
 as a parameter;
if Proposed Method 1 has been chosen then
    construct similarity matrix based on cosine similarity between
     customers;
    construct weight matrix using the trust matrix of the g and the
     similarity matrix;
end
else
    weight matrix equals to the trust matrix of the g;
end
foreach customer c do
    choose n customers that c trusts most as neighbors;
    foreach product p purchased by the neighbours do
        if c purchased p then
            continue;
        end
        else
            calculate recommendation coefficient using the
             corresponding rows of the weight and customers versus
             products matrices for p;
        end
    end
    recommend k products with the highest recommendation
     coefficients;
end
```

2.5.2.1 Calculating Weights

If 2.5.1.1 being used,

$$w(c_1, c_2) = \alpha * sim(c_1, c_2) + (1 - \alpha) * trust(c_1, c_2)$$

where $sim(c_1, c_2)$ represents "cosine similarity" between customers, $trust(c_1, c_2)$ represents trust calculated by the Graph Module and α is a weight ratio that changes according to the dataset .

Otherwise, $w(c_1, c_2)$ directly equals to $trust(c_1, c_2)$ since 2.5.1.1 method is already based on similarity.

2.5.2.2 Calculating Recommendation Coefficients

$$RC(i) = \frac{\sum_{c \in C} w(c_{target}, c) * b_c}{\sum_{c \in C} w(c_{target}, c)}$$

where $RC(i)$ represents the recommendation coefficient calculated for product i and b_c is a boolean value which indicates whether the product was bought by person c or not.

2.5.2.3 Making Predictions

$$p(i) = \frac{\sum_{c \in C} w(c_{target}, c) * r_c}{\sum_{c \in C} w(c_{target}, c)}$$

where $p(i)$ represents the prediction for product i and r_c represents rating given by customer c for product i. C customer set only contains the customers who purchased product i.

2.5.3 Recommender Module

Recommender Module is responsible for reading/writing. The module has two tasks:

1. Getting transaction list which contains customer id -product id pairs from the Neo4j database using neo4j driver and sending the list to the Filterer module as a parameter.

2. Getting the recommendation list that contains ids of the customers and corresponding recommended products from the Filterer module and writing these recommendations to the Neo4j database as a relationship between the customer and the recommended product using neo4j driver.

2.6 Testing Phase

Although there are many factors such as diversity, coverage, serendipity that determine the efficiency of recommendation systems, in this project we focused on accuracy and decided to left other factors beyond the scope.

2.6.1 Methods

2.6.1.1 Leave-one-out Cross Validation

2.6.1.2 Shuffle Split Cross Validation

2.6.2 Results

2.6.2.1 Stockmount

About the dataset: Dataset origanally consists of 142501 customers and 73482 products with implicit ratings (i.e. purchased or not). However, to make the customer versus product matrix a little denser, customers with less than 2 products and products with less than 2 customers are eliminated. After filtering,

	filtering threshold = 2
Number of Customers	2182
Number of Products	1355
Number of Communities	175
Sparsity	99.808124 %

Testing method: Since the dataset contains implicit ratings, I preferred to use "Hit Rate" rather than RMSE or MAE as the test metric and "Leave-one-out CV" as the cross validation iterator. Basically, for each deleted transaction, test module checked whether the product in the deleted transaction was within the recommended products.

Tested Recommender: Eigentrust Weighted Trust Based Recommender 2.4

Results:

	5 Recommendations	10 Recommendations
Number of Tests	4160	4160
Number of Hits	1367	1983
Hit Rate	0.328605	0.476682

Results were analysed on the basis of community density, community size and number of products purchased by the customer, but no solid correlation with accuracy found.

2.6.2.2 Amazon Food Reviews [6]

About the dataset:

	Amazon Food Reviews (threshold = 10)
Number of Users	4276
Number of Products	1140
Rating Range	1-5

Testing method: For this dataset, I wanted to see how efficiently the recommender works on extremely sparse datasets, as a result I preferred to use "Shuffle Split CV" as the cross validation iterator since it is easy to set up the test and train set ratios. The tests with the test set size greater than 0.5 were applied to see the performance of the implemented recommender in cold start and to understand how it improved the collaborative filtering method using the "mean squared difference similarity". As the test metrics, RMSE and MAE were used. Tests were performed using Surprise.
Tested Recommender: Inverse Distance Weighted Trust Based Recommender (with the Proposed Method 2)2.5

Benchmark:
Testset: 0.2, Trainset: 0.8, Sparsity: 0.9891

	Trust Based	MSD	SVD	Slope One	KNN	NMF	CoCluster
RMSE	0.6727	0.6443	0.6944	0.7167	0.6615	0.6661	0.7199
MAE	0.3321	0.3089	0.471	0.4049	0.395	0.4131	0.4441

Testset: 0.3, Trainset: 0.7, Sparsity: 0.9904

	Trust Based	MSD	SVD	Slope One	KNN	NMF	CoCluster
RMSE	0.672	0.6599	0.7252	0.7226	0.6906	0.6852	0.7133
MAE	0.3375	0.3239	0.5081	0.4074	0.4203	0.4308	0.415

Testset: 0.4, Trainset: 0.6, Sparsity: 0.9918

	Trust Based	MSD	SVD	Slope One	KNN	NMF	CoCluster
RMSE	0.7012	0.7211	0.7633	0.7396	0.7231	0.7086	0.7156
MAE	0.3636	0.3594	0.5548	0.4146	0.4458	0.4491	0.4232

Testset: 0.5, Trainset: 0.5, Sparsity: 0.9931

	Trust Based	MSD	SVD	Slope One	KNN	NMF	CoCluster
RMSE	0.7331	0.7784	0.8212	0.7648	0.7562	0.7447	0.7624
MAE	0.3975	0.3971	0.6144	0.429	0.4835	0.4848	0.463

Testset: 0.6, Trainset: 0.4, Sparsity: 0.9945

	Trust Based	MSD	SVD	Slope One	KNN	NMF	CoCluster
RMSE	0.7719	0.91	0.8787	0.7879	0.8241	0.7949	0.7545
MAE	0.4382	0.481	0.676	0.4437	0.5419	0.5217	0.4422

Testset: 0.7, Trainset: 0.3, Sparsity: 0.9958

	Trust Based	MSD	SVD	Slope One	KNN	NMF	CoCluster
RMSE	0.8671	1.1469	0.9456	0.8266	0.9157	0.863	0.8238
MAE	0.5186	0.6468	0.739	0.4611	0.6179	0.5848	0.5065

Testset: 0.8, Trainset: 0.2, Sparsity: 0.9970

	Trust Based	MSD	SVD	Slope One	KNN	NMF	CoCluster
RMSE	0.9893	1.5542	1.0201	0.9124	1.0562	0.9721	0.877
MAE	0.6396	0.9878	0.8088	0.5124	0.7207	0.7005	0.5391

Testset: 0.9, Trainset: 0.1, Sparsity: 0.9979

	Trust Based	MSD	SVD	Slope One	KNN	NMF	CoCluster
RMSE	1.2693	2.2067	1.106	1.0236	1.2687	1.1095	1.0273
MAE	0.8898	1.6609	0.8843	0.615	0.919	0.8506	0.6876

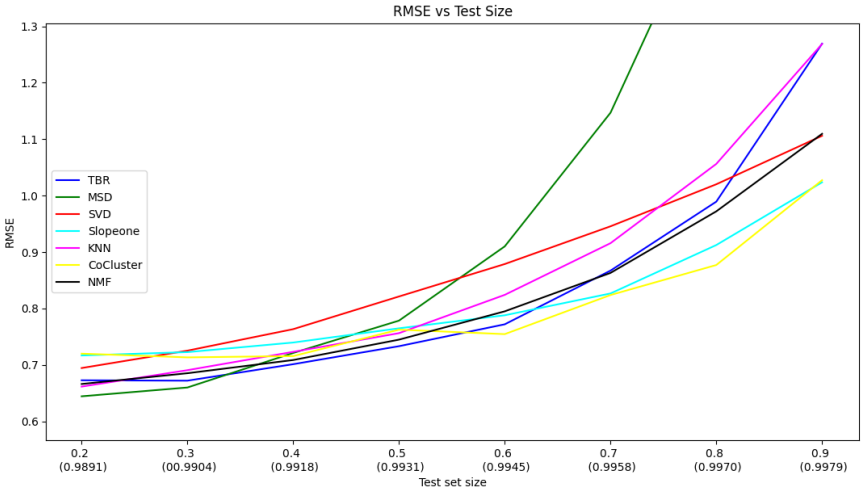

Figure 5: RMSE versus Testset Size. For instance, 0.2 means testset-trainset ratio is $20\% - 80\%$. Additionally, floats in parentheses represent the sparsity of trainset for corresponding test set size

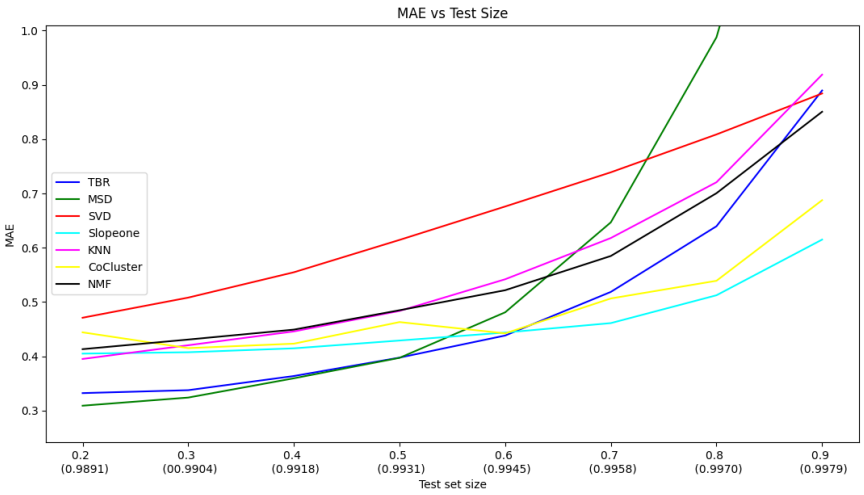

Figure 6: MAE versus Testset Size

3 Organization

3.1 METU Data Mining Research Group

Data mining is a process of extracting special patterns and useful information hidden in the raw data involving methods at the intersection of statistics, machine learning, and database systems. Data Mining Group is a research group that continues its research on data mining and big data within METU. The focus of the research group is especially on web mining and multi-relational data mining.

Some of the selected publications of the METU Data Mining Research Group:

- Y. Kavurucu, P. Senkul , I.H. Toroslu, "Concept Discovery on Relational Databases: New Techniques for Search Space Pruning and Rule Quality Improvement", Knowledge-Based Systems, vol:23, issue:8, 743-756pp, December 2010.(doi:10.1016/j.knosys.2010.04.011.)

- L. A. Guner, N. I. Karabacak, O. U. Akdemir, P. Senkul Karagoz, S.

A. Kocaman, A. Cengel, M. Unlu, "An open-source framework of neural networks for diagnosis of coronary artery disease from myocardial perfusion SPECT", Journal of Nuclear Cardiology, vol: 17, issue: 3, 405-413pp, June 2010, doi:10.1007/s12350-010-9207-5.

- A. Mutlu, M. A. Berk, P. Senkul , Improving the Time Efficiency of ILP-based Multi-Relational Concept Discovery with Dynamic Programming Approach, ISCIS 2010, London, UK, Sept 22-24, 2010.

The rest of the publications and additional information can be found at `http://ceng.metu.edu.tr/data-mining-group`

4 Conclusion

I started my internship without any knowledge about recommender systems. Throughout the internship, I experienced first hand the major challenges such as sparsity, scalability, cold start, etc. that recommender systems face and observed the mainstream approaches developed to overcome these issues. Moreover, I analysed the state-of-the-art algorithms designed for recommendation systems with their strengths and weaknesses.

As the next step, we needed to test the performances of the implemented recommenders. During the testing phase, I familiarized myself with different cross-validation types such as Leave-one-out, K-Fold, etc. and applied these methods to determine the performance of recommenders I built. Furthermore, I acquired profound knowledge about popular evaluation metrics and the situations in which they are preferred. For example, RMSE and MAE are widely-used metrics for datasets containing explicit ratings while hit rate and map@k which is especially useful if we care about the order of recommended products are preferred for datasets with implicit ratings.

From the technical aspect, as a result of working with Neo4j, I learnt the basics, pros, and cons of graph databases. Especially the data visualization feature of Neo4j was really helpful to analyse the dataset. The first versions of the recommenders I implemented were running very slowly due to lots of unnecessary iterations, the solution was performing operations with a matrix approach using Numpy. This experience showed me the efficiency of Numpy coming from being written based on C. Since one of the recommenders is graph-based, I got to know python libraries with graph utilities such as Scipy

and Scikit-learn which provided information that could be also useful in future projects.

On the whole, the internship has provided me new insights into the recommender systems and in general data science. Not only the internship developed my technical skills and perspective but also it allowed me to see the internal dynamics of the technologies around me. For instance, now I have an idea of how the shopping platform I visit makes recommendations or why sites like Linkedin and Twitter ask new members to follow someone as soon as they register.

5 Appendix

5.1 Distributon of the Eigentrust Scores

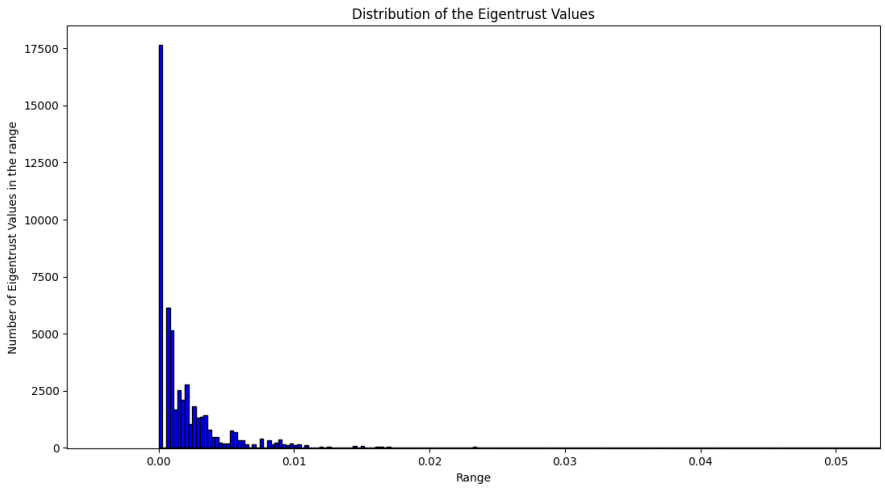

Figure 7: Distribution of the Eigentrust scores before filtering. As can be seen, nearly 17500 of the customers have zero Eigentrust.

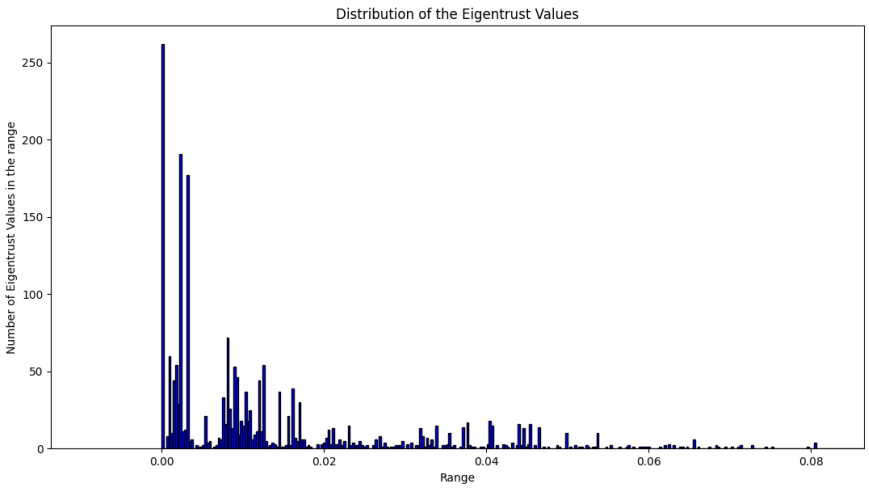

Figure 8: Distribution of the Eigentrust scores after filtering.

5.2 Libraries Used in Implementation Phase

5.2.1 Neo4j

Neo4j is a graph database management system where data is organized as nodes, relationships, and properties. The communication between recommender and the database is maintained by Neo4j driver.

Driver Installation :

```
pip install neo4j

```

Configuration :

```
import neo4j
...

uri = self._config["database"]["neo4j"]["uri"]
user = self._config["database"]["neo4j"]["user"]
password = self._config["database"]["neo4j"]["password"]

```

```
self._driver = neo4j.Driver(uri, auth=(user, password))


```

Sample Usage :

```
import neo4j
...

def get_customer_trust(self, customer_id):

  query = (
    f"MATCH (u:Customer)-[r:BELONGS_IN]->(:Community) "
    f"WHERE u.id = {repr(customer_id)} "
    f"RETURN r.eigentrust"
  )

  with self._driver.session() as session:
    return tuple(session.run(query).single())


```

Listing 1: Neo4j driver example

5.2.2 Numpy

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices. Since the core of Numpy is optimized C code, performing the calculations in the recommendation process using Numpy matrix provides serious time savings.

Installation :

```
pip install numpy

```

Sample Usage :

```
import numpy as np

class TrustBasedFilterer(object):
...

```

```
  def _create_customers_versus_products_table(self):

    self._customers_versus_products_table = np.zeros(
      (self._unique_customers.shape[0],
      self._unique_products.shape[0]),
      dtype=np.bool,
      )

    self._customers_versus_products_table[
      self._sales[:, 0],
      self._sales[:, 1],
    ] = True

```

Listing 2: Numpy example

5.2.3 Scipy

SciPy is a Python library used for scientific computing. Similar to Numpy, many of the Scipy functions are written in C which provides a solution to the slowness caused by interpretation. For this reason, we prefer to use the "Dijkstra's Algorithm" provided by Scipy rather than implementing it by ourselves.

Installation :

```
pip install scipy

```

Sample Usage :

```
from scipy.sparse import csr_matrix
from scipy.sparse.csgraph import dijkstra

class Graph(object):
...

  def _create_distance_matrix(self):

    self._create_adjacency_matrix()

    self._adjacency_matrix = \
      csr_matrix(self._adjacency_matrix)
```

```
13
14      self._distance_matrix = dijkstra(
15        csgraph=self._adjacency_matrix,
16        directed=False,
17        return_predecessors=False,
18        unweighted=True,
19        limit=self._max_distance)
20
21      self._distance_matrix\
22        [~np.isfinite(self._distance_matrix)] = 0
23
```

Listing 3: Scipy example

5.3 Libraries Used in Testing Phase

5.3.1 Surprise

Surprise is a Python library for building and analyzing recommender systems. We use it for evaluating the performance of the trust based recommenders on various datasets and comparing them with the provided built-in recommender systems.

Installation :

```
1  pip install scikit-surprise
2
```

Sample Usage :

```
1  from surprise import AlgoBase, PredictionImpossible,
    Dataset
2  from surprise.model_selection import cross_validate
3
4  class Inverse_distance_weighted_tbr(AlgoBase):
5  ...
6
7  reader =  Reader(line_format='user item rating', sep='\t',
    rating_scale=(1, 5))
8
9  data = Dataset.load_from_file('./dataset.csv', reader=
    reader)
10 algo = Inverse_distance_weighted_tbr()
```

```

cross_validate(algo, data, cv=5, verbose=True)

```

Listing 4: Surprise example

5.3.2 Matplotlib

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. We use it to visualize the evaluation results of the recommenders and statistical properties of the dataset such as the distribution of Eigentrust.

Installation :

```
pip install matplotlib

```

Sample Usage :

```
import matplotlib.pyplot as plt
...
plt.hist(eigentrust_list,
color = 'blue',
edgecolor = 'black',
bins = bins)
plt.title('Distribution of the Eigentrust Values')
plt.xlabel('Range')
plt.ylabel('Number of Eigentrust Values in the range')
plt.show()

```

Listing 5: Matplotlib example

References

[1] Aksoy, K., Odabas, M., Bozdogan, I., and Temur, A. Tacorec. https://senior.ceng.metu.edu.tr/2020/tacorec/, 2020.

[2] Kamvar, S. D., Schlosser, M. T., and Garcia-Molina, H. The eigentrust algorithm for reputation management in p2p networks. In

Proceedings of the 12th International Conference on World Wide Web (New York, NY, USA, 2003), WWW '03, Association for Computing Machinery, p. 640–651.

[3] Lathia, N., Hailes, S., and Capra, L. Trust-based collaborative filtering. vol. 263.

[4] Li, Y., Liu, J., Ren, J., and Chang, Y. A novel implicit trust recommendation approach for rating prediction. *IEEE Access 8* (2020), 98305–98315.

[5] Massa, P., and Avesani, P. Trust-aware recommender systems. In *Proceedings of the 2007 ACM Conference on Recommender Systems* (New York, NY, USA, 2007), RecSys '07, Association for Computing Machinery, p. 17–24.

[6] McAuley, J., and Leskovec, J. From amateurs to connoisseurs: Modeling the evolution of user expertise through online reviews, 2013.

[7] Papagelis, M., Plexousakis, D., and Kutsuras, T. Alleviating the sparsity problem of collaborative filtering using trust inferences. In *Trust Management* (Berlin, Heidelberg, 2005), P. Herrmann, V. Issarny, and S. Shiu, Eds., Springer Berlin Heidelberg, pp. 224–239.

[8] Richardson, M., Agrawal, R., and Domingos, P. Trust management for the semantic web. In *The Semantic Web - ISWC 2003* (Berlin, Heidelberg, 2003), D. Fensel, K. Sycara, and J. Mylopoulos, Eds., Springer Berlin Heidelberg, pp. 351–368.