



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: 04.10.2020

Total Working Dates: 62

October 28, 2020

Student:
Onat ÖZDEMİR

Supervisors:
Prof.Dr. Pınar
KARAGÖZ
Prof.Dr. İsmail Hakkı
TOROSLU

Contents

1	Introduction	3
2	Project	3
2.1	Analysis Phase	3
2.2	Design Phase	4
2.2.1	For Eigentrust Weighted Recommender	4
2.2.2	For Inverse Distance Weighted Recommender	5
2.3	Implementation Phase	5
2.4	Eigentrust Weighted Trust Based Recommender	6
2.4.1	About Eigentrust	6
2.4.2	Filterer Module	6
2.4.2.1	Calculating Weights	7
2.4.2.2	Calculating Recommendation Coefficients	8
2.4.2.3	Making Predictions	8
2.4.3	Recommender Module	8
2.5	Inverse Distance Weighted Trust Based Recommender	9
2.5.1	Graph Module	9
2.5.1.1	Constructing Adjacency Matrix and Distance Matrix	9
2.5.1.2	Trust Calculation	11
2.5.2	Filterer Module	11
2.5.2.1	Calculating Weights	13
2.5.2.2	Calculating Recommendation Coefficients	13
2.5.2.3	Making Predictions	13
2.5.3	Recommender Module	13
2.6	Testing Phase	14
2.6.1	Methods	14
2.6.1.1	Leave-one-out Cross Validation	14
2.6.1.2	Shuffle Split Cross Validation	14
2.6.2	Results	14
2.6.2.1	Stockmount	14
2.6.2.2	Amazon Food Reviews [6]	15
3	Organization	19
3.1	METU Data Mining Research Group	19

4	Conclusion	19
5	Appendix	21
5.1	Distributon of the Eigentrust Scores	21
5.2	Libraries Used in Implementation Phase	22
5.2.1	Neo4j	22
5.2.2	Numpy	23
5.2.3	Scipy	24
5.3	Libraries Used in Testing Phase	25
5.3.1	Surprise	25
5.3.2	Matplotlib	25

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 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 recommender systems is "collaborative filtering." The approach uses prior behaviors of customers, such as rating profiles, product preferences, etc., to generate recommendations. Collaborative filtering methods can be classified according to which factors 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 argues 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 behaviors of customers (e.g., ratings) to make good recommendations in the absence of a 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, and user attributes (especially in hybrid recommenders), etc.

During the internship, due to the 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" into the collaborative filtering algorithm.

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

2.2 Design Phase

2.2.1 For Eigentrust Weighted Recommender

Eigentrust represents how strongly connected the customers are to their communities (for the detailed information, please check section 2.4.1) and is 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 Eigentrust scores are calculated between the members of each 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 scores) 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 customers based on transaction records, calculating recommendation coefficients for each product and generating recommendations according to 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 taking 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, the implementation details of the recommenders are divided into two subsections: Section 2.4 and Section 2.5.

2.4 Eigentrust Weighted Trust Based Recommender

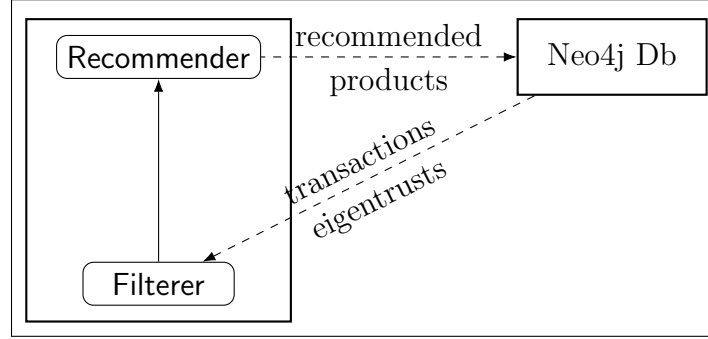


Figure 1: Recommender Structure

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

Pseudocode for the Filterer

```

for community i do
    get transactions and eigentrusts of the customers belonging to i;
    construct customers versus products matrix using transactions;
    construct trust matrix using the eigentrusts;
    construct similarity matrix based on cosine similarity between
        customers belonging to i;
    construct weight matrix using the trust and similarity matrices;
    for customer c belonging to i do
        for product p purchased by neighbours do
            if c purchased p then
                | continue;
            end
            else
                | calculate recommendation coefficient of p using the
                | weight and customers versus products matrices;
            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 the "cosine similarity" between customers and $trust(c_2)$ represents the Eigentrust of 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 rate prediction for product i and r_c represents the 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 who belong 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

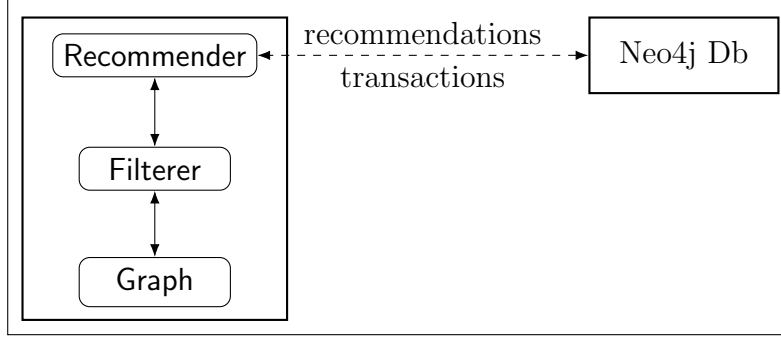


Figure 2: Recommender Structure

Inverse Distance Weighted Trust Based Recommender consists of three modules:

2.5.1 Graph Module

Graph Module is responsible for three tasks:

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

2.5.1.1 Constructing Adjacency Matrix and Distance Matrix

Since the recommender system is tested on both the datasets with implicit ratings and the datasets with explicit ratings, to construct the "adjacency matrix" from the 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**.

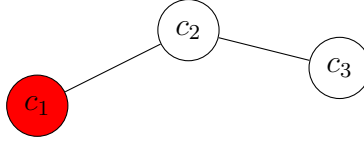


Figure 3: c_1 and c_2 have at least one common product while c_1 and c_3 don't have any common product

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 only 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**.

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

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.

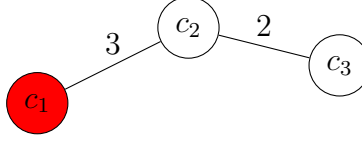


Figure 4: euclidean distance between c_1 and c_2 equals to 3, while c_1 and c_3 don't have any common product

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, recommender systems using euclidean distance-based similarity fails since they cannot calculate similarity scores for the customer pairs who do not have any common products. Since the "Dijkstra's Algorithm" propagates the weights even for the mentioned customer pairs, we are able to calculate the trust scores between them.

2.5.2 Filterer Module

The Filterer Module is responsible for;

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

- Calculating the weights
- If the dataset consists of implicit ratings, calculating the 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 from the transactions
provided by the Recommender;
create a Graph object g by providing 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 g and the
    | similarity matrix;
end
else
    | weight matrix equals to the trust matrix of g;
end
for customer c do
    | choose n customers that c trusts most as neighbors;
    | for product p purchased by the neighbours do
    | | if c purchased p then
    | | | continue;
    | | end
    | | else
    | | | calculate recommendation coefficient of p using the
    | | | corresponding rows of the weight and customers versus
    | | | products matrices;
    | | end
    | end
    | recommend k products with the highest recommendation
    | coefficients;
end

```

2.5.2.1 Calculating Weights

If 2.5.1.1 being used,

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

where $\text{sim}(c_1, c_2)$ represents the "cosine similarity" between customers, $\text{trust}(c_1, c_2)$ represents the 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 $\text{trust}(c_1, c_2)$ 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 rate prediction for product i and r_c represents the 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 the list of transactions which contains (customer id, product id) pairs from the Neo4j database using Neo4j driver and passing the list to the Filterer module as a parameter.

2. Getting the list of recommendations 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, and serendipity that determine the efficiency of recommendation systems, in this project, we focused on accuracy and decided to leave other factors outside the scope of this project.

2.6.1 Methods

2.6.1.1 Leave-one-out Cross Validation

In Leave-one-out Cross Validation, for the dataset with n data, $n-1$ samples are used as training set while the remaining singleton is used to test accuracy. This process is repeated for all n data, and the final result is calculated by taking the average of these n tests.

2.6.1.2 Shuffle Split Cross Validation

Shuffle Split Cross Validation randomly samples the dataset during each iteration to generate training and test sets at the given ratios. For the Shuffle Split CV, the number of splits was chosen as three, and the final result is calculated by taking the average of these three iterations.

2.6.2 Results

2.6.2.1 Stockmount

About the dataset: Dataset originally 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 were eliminated. After the filtering,

	filtering threshold = 2
Number of Customers	2182
Number of Products	1355
Number of Communities	175
Sparsity of the customers versus products matrix	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 transaction, test module used the remaining transactions as training set and checked whether the product in the tested transaction was within the recommended products or not.

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 analyzed on the basis of community density, community size, and the number of products purchased by the customer, but no significant correlation with accuracy was 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 training 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 that uses the "Mean Squared Difference Similarity". As the test metrics, RMSE and MAE were used. The tests were performed using Surprise.

Tested Recommender: Inverse Distance Weighted Trust Based Recommender (with the Proposed Method 2)2.5

Benchmark:

Test set: 0.2, Training set: 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

Test set: 0.3, Training set: 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

Test set: 0.4, Training set: 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

Test sets with ratios higher than 0.5 were applied for testing the cold start efficiency

Test set: 0.5, Training set: 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

Test set: 0.6, Training set: 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

Test set: 0.7, Training set: 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

Test set: 0.8, Training set: 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

Test set: 0.9, Training set: 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

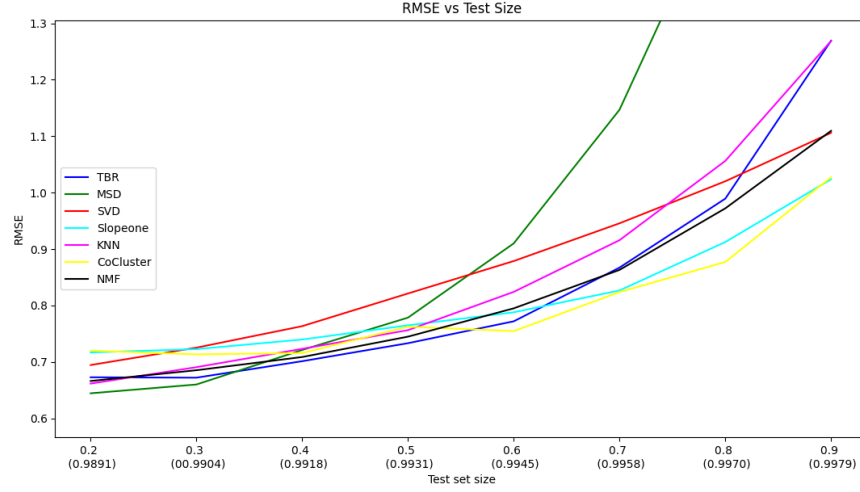


Figure 5: RMSE versus Test set Size. For instance, 0.2 means test set-training set ratio is 20% – 80% . Additionally, floats in parentheses represent the sparsity of the customers versus products matrix which is filled with training set for corresponding test set size

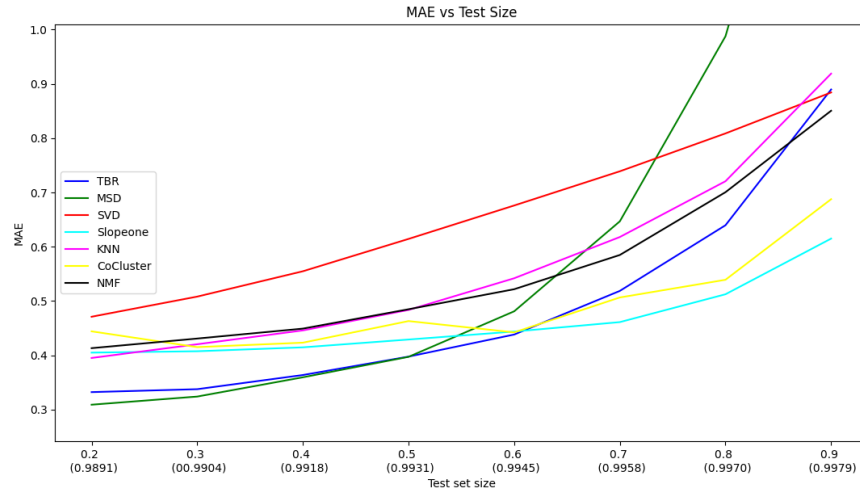


Figure 6: MAE versus Testset Size

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 firsthand 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 analyzed 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 performances of the recommender systems I implemented. 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 are 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 learned the basics, pros, and cons of graph databases. Especially the data visualization feature of Neo4j was really helpful to analyze the dataset. The first versions of the recommenders I implemented were running very slowly due to lots of unnecessary iterations; the solution was performing the 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 also be useful in future projects.

On the whole, the internship has provided me with new insights into recommender systems and, in general, data science.

5 Appendix

5.1 Distributon of the Eigentrust Scores

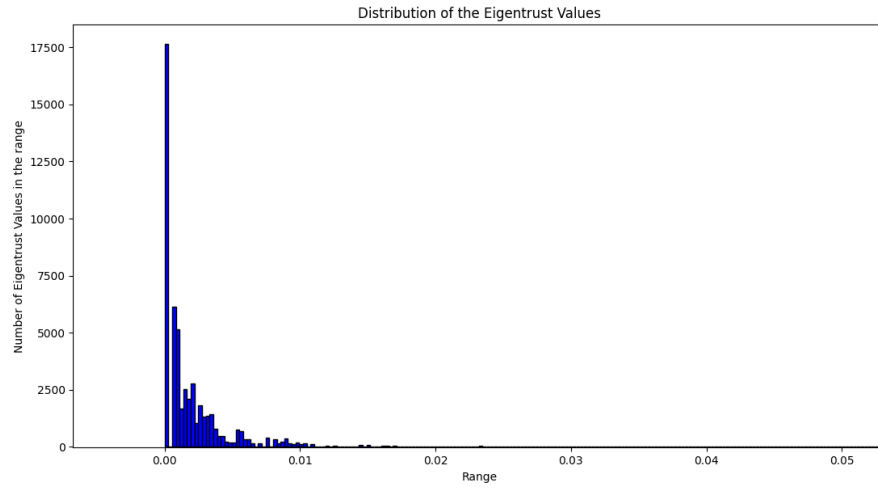


Figure 7: Distribution of the Eigentrust scores before filtering. As can be seen, nearly 17500 customers have zero Eigentrust.

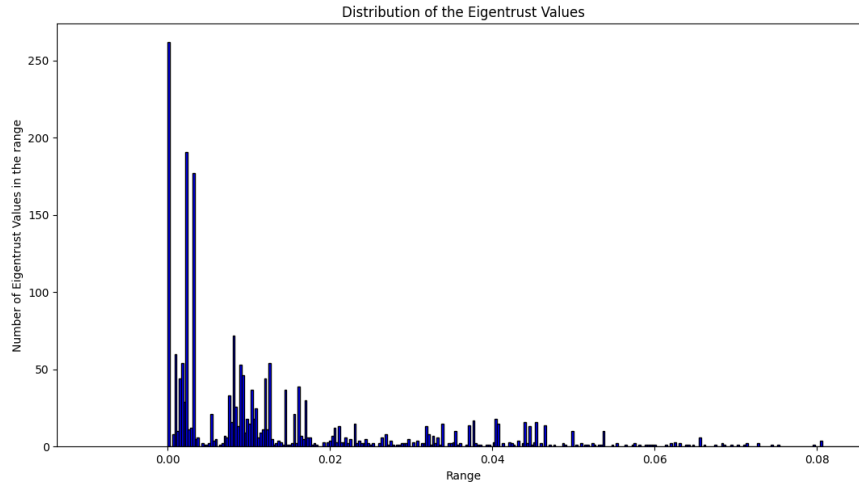


Figure 8: Distribution of the Eigentrust scores after filtering (threshold=2).

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 was maintained by Neo4j driver.

Driver Installation :

```
1 pip install neo4j
```

Configuration :

```
1 import neo4j
2 ...
3
4 uri = self._config["database"]["neo4j"]["uri"]
5 user = self._config["database"]["neo4j"]["user"]
6 password = self._config["database"]["neo4j"]["password"]
7
8 self._driver = neo4j.Driver(uri, auth=(user, password))
9
```

Sample Usage :

```
1 import neo4j
2 ...
3
4 def get_customer_trust(self, customer_id):
5
6     query = (
7         f"MATCH (u:Customer)-[r:BELONGS_IN]->(:Community) "
8         f"WHERE u.id = {repr(customer_id)} "
9         f"RETURN r.eigentrust"
10    )
11
12    with self._driver.session() as session:
13        return tuple(session.run(query).single())
14
```

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 the Numpy matrix provides serious time savings.

Installation :

```
1 pip install numpy
```

Sample Usage :

```
1 import numpy as np
2
3 class TrustBasedFilterer(object):
4     ...
5
6     def _create_customers_versus_products_table(self):
7
8         self._customers_versus_products_table = np.zeros(
9             (self._unique_customers.shape[0],
10              self._unique_products.shape[0]),
11             dtype=np.bool,
12         )
```



```

13
14 self._customers_versus_products_table[
15 self._sales[:, 0],
16 self._sales[:, 1],
17 ] = 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.

Installation :

```

1 pip install scipy

```

Sample Usage :

```

1 from scipy.sparse import csr_matrix
2 from scipy.sparse.csgraph import dijkstra
3
4 class Graph(object):
5     ...
6
7     def _create_distance_matrix(self):
8
9         self._create_adjacency_matrix()
10
11         self._adjacency_matrix = \
12             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

```

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 used 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
```

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()
11
12    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 used it to visualize the evaluation results of the recommenders and statistical properties of the dataset such as the distribution of Eigentrust.

Installation :

```
1 pip install matplotlib
```

Sample Usage :

```
1 import matplotlib.pyplot as plt
2 ...
3 plt.hist(eigentrust_list,
4 color = 'blue',
5 edgecolor = 'black',
6 bins = bins)
7 plt.title('Distribution of the Eigentrust Values')
8 plt.xlabel('Range')
9 plt.ylabel('Number of Eigentrust Values in the range')
10 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.