MIDDLE EAST TECHNICAL UNIVERSITY

Department Of Computer Engineering

CENG 300

Summer Practice Report

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

October 4, 2020

Student:
Onat ÖZDEMİR

Instructors:
Prof.Dr.Pınar KARAGÖZ

Prof.Dr.İsmail Hakkı TOROSLU

Student's Signature

Organization Approval

Contents

1 Introduction **2**

2 Project **2**
2.1 Analysis Phase 2
2.2 Design Phase 2
2.2.1 For Eigentrust Weighted Recommender 2
2.2.2 For Inverse Distance Weighted Recommender 3
2.3 Implementation Phase 3
2.3.1 Neo4j 3
2.3.2 Numpy 4
2.3.3 Scipy 5
2.4 Eigentrust Weighted Trust Based Recommender 6
2.4.1 About Eigentrust 6
2.4.2 Filterer Module 7
2.4.3 Recommender Module 7
2.5 Inverse Distance Weighted Trust Based Recommender 8
2.5.1 Graph Module 8
2.5.1.1 Constructing Adjacency Matrix and Distance Matrix 8
2.5.1.2 Trust Calculation 10
2.5.2 Filterer Module 10
2.5.3 Recommender Module 11
2.6 Testing Phase 11
2.6.1 Methods 11
2.6.1.1 Leave-one-out Cross Validation 11
2.6.1.2 K-Fold Cross Validation 11
2.6.2 Results 11
2.6.3 Libraries I Used 11
2.6.3.1 Surprise 11
2.6.3.2 Matplotlib 12

3 Organization **13**
3.1 METU Data Mining Research Group 13

4 Conclusion **13**

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 task I have worked on was implementing a Trust Based Recommender using the collaborative filtering method and testing it on provided dataset. The dataset contains information about customers and the products they have bought. In addition to dataset, I was able to use eigentrust calculation and community detection modules provided by the TACOREC.

2 Project

During the internship, I implemented two trust based recommenders with different weightening 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

There were two problems I need to solve:
1 - Dataset was very sparse
2 - There was no explicit trust information
In the both implementations I have made, first case handled by filtering the customers and products which purchased and were bought more than filtering threshold times. To gain better understanding on the second problem, I studied implicit trust calculation methods and looked into lots of research papers.

2.2 Design Phase

2.2.1 For Eigentrust Weighted Recommender

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

1. We don't need to recalculate trust values, it's already stored in the database and all we need to get it using neo4j driver.

2. Since eigentrust values are community dependent, rather than iterating over customer list and storing all the eigentrust values in a huge matrix, we can iterate over communities and create eigentrust matrix for each community.

Based on these conclusions, I designed two modules: Recommender and Filterer. The only task of the Recommender is getting recommendations from Filterer module and writing them to database. The remaining weight of the project is carried by the filter: getting transactions and eigentrust values via neo4j driver, calculating recommendation coefficients for each product, sorting these coefficients and composing recommended items list from k number of products with the highest recommendation coefficients. For the detailed information and recommender structure, please check section 2.4 and Figure 1.

2.2.2 For Inverse Distance Weighted Recommender

For the detailed information and recommender structure, please check section 2.5 and Figure 3.

2.3 Implementation Phase

Since there are two different implementations, I have divided the implementational details of the recommenders into two subsections: section 2.4 and 2.5. Under this subsection, libraries and technologies used in implementations are explained.

2.3.1 Neo4j

Driver Installation :

```
pip install neo4j

```

Configuration :

```python
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 :

```python
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

2.3.2 Numpy

Installation :

```
pip install numpy

```

Sample Usage :

```python
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

2.3.3 Scipy

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)

self._distance_matrix = dijkstra(
csgraph=self._adjacency_matrix,
directed=False,
return_predecessors=False,
unweighted=True,
```

```
limit=self._max_distance)

self._distance_matrix\
[~np.isfinite(self._distance_matrix)] = 0

```

Listing 3: Scipy example

2.4 Eigentrust Weighted Trust Based Recommender

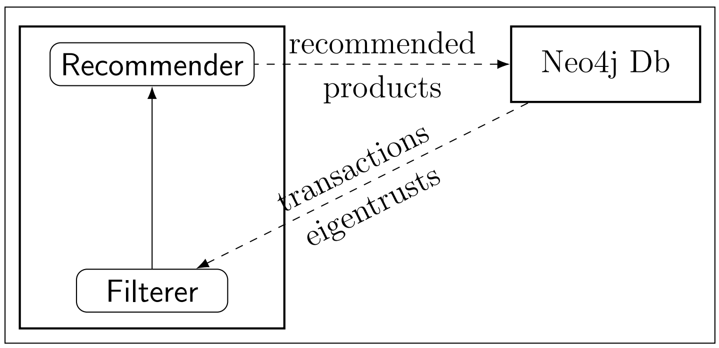

Figure 1: Recommender Structure

2.4.1 About Eigentrust

Eigentrust[4] is a reputation calculation algorithm mainly designed for peer-to-peer networks. In our case, Eigentrust represents how strongly connected the customers are to their communities. Eigentrust values calculated by the eigentrust module provided by TACoRec[1] and stored in Neo4j database as a property of the relationship between a customer and his/her community.

Problem encountered with Eigentrust: Especially for the customers connected to communities with small size and low densities, eigentrust values stored in the database are either very small or equal to zero (check Figure 2). Most of the customers with zero eigentrust values are eliminated after filtering the network from customers with a small number of products. Unfortunately, eigentrust values are still quite small.

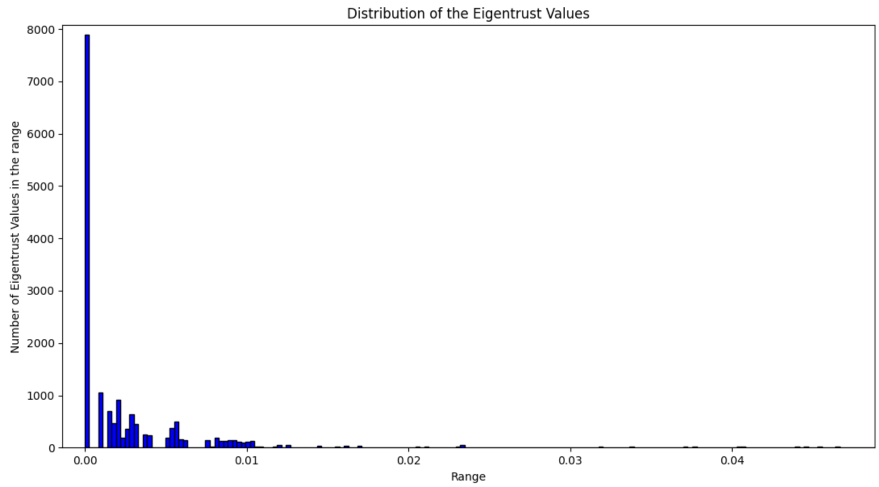

Figure 2: Distribution of the Eigentrust Values. As can be seen nearly 8000 of the customers have zero eigentrust value.

2.4.2 Filterer Module

2.4.3 Recommender Module

Recommender Module is basically responsible from 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 parameter.

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

2.5 Inverse Distance Weighted Trust Based Recommender

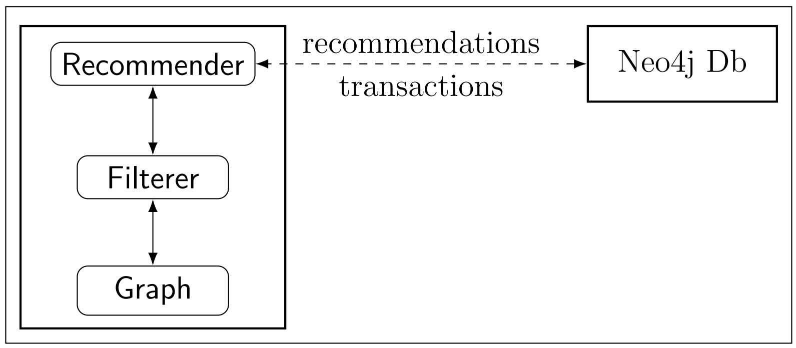

Figure 3: Recommender Structure

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 "customer versus product table" provided by Filterer module
2. Constructing "distance matrix" using "adjacency matrix"
3. Constructing "trust matrix" using "distance matrix"

2.5.1.1 Constructing Adjacency Matrix and Distance Matrix

Since the recommender is tested in 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 purchased a joint 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**.

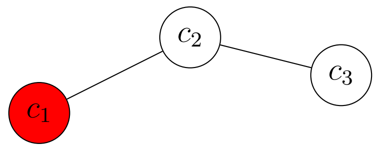

Figure 4: c_1 and c_2 have purchased at least 1 joint product, but c_1 and c_3 do not have a joint product

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**.

$$adj[c_1]][c_2] = \sqrt{\sum_{i \in I_1 \cap I2} (r1_i - r2_i)^2} \tag{1}$$

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 joint products are included in the calculation.

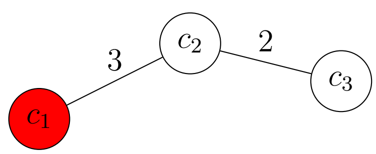

Figure 5: euclidean distance between c_1 and c_2 equals 3, and c_1 and c_3 do not have a joint product

Problem encountered with euclidean distance method: As the module is generally tested on sparse datasets,

Dijkstra's Algorithm

To construct the "distance matrix", the graph module uses "Dijkstra's Algorithm"[2] which takes the adjacency matrix as 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 similarity score for the 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 the customers.

2.5.2 Filterer Module

The module initially takes transaction list and filters it to create denser customer versus product matrix. Then creates a graph object by giving the customer versus product matrix as parameter and use the trust matrix created by the graph object to calculate recommendation coefficient for each product. Finally, for each user, the module sorts all products with respect to their recommendation coefficients in descending order and recommends top k of them.

$$w(c_1, c_2) = \frac{2 * sim(c_1, c_2) * trust(c_2)}{sim(c_1, c_2) + trust(c_2)} \tag{2}$$

Since the dataset contains implicit feedbacks, I didn't use a predictor algorithm for it. However, for the Movielens 100k dataset[3], (2) was used to predict the rating given by the c_{target} for item i.

$$p(i) = \frac{\sum_{c \in C} w(c_{target}, c) * r_c}{\sum_{c \in C} w(c_{target}, c)} \tag{3}$$

2.5.3 Recommender Module

Recommender Module is basically responsible from 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 parameter.

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

2.6 Testing Phase

2.6.1 Methods

2.6.1.1 Leave-one-out Cross Validation

2.6.1.2 K-Fold Cross Validation

2.6.2 Results

2.6.3 Libraries I Used

2.6.3.1 Surprise

Installation :

```
pip install scikit-surprise

```

Sample Usage :

```
from surprise import AlgoBase, PredictionImpossible,
 Dataset
from surprise.model_selection import cross_validate

class Inverse_distance_weighted_tbr(AlgoBase):
...
```

```python

reader = reader = Reader(line_format='user item rating
 timestamp', sep=';', rating_scale=(1, 5))

data = Dataset.load_from_file('./dataset.csv', reader=
 reader)
algo = Inverse_distance_weighted_tbr()

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

```

Listing 4: Surprise example

2.6.3.2 Matplotlib

Installation :

```
pip install matplotlib

```

Sample Usage :

```python
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

3 Organization

3.1 METU Data Mining Research Group

4 Conclusion

References

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

[2] Dijkstra, E. W. Dijkstra's algorithm. https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm.

[3] Harper, F. M., and Konstan, J. A. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst. 5*, 4 (Dec. 2015).

[4] 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.