ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ
ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΤΜΗΜΑΤΟΣ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ

Διαδραστικό σύστημα συστάσεων

Διπλωματική Εργασία

της

Γκιουλέκας Φωτεινής

Θεσσαλονίκη, Νοέμβριος 2019

UNIVERSITY OF MACEDONIA
GRADUATE MASTER PROGRAM
DEPARTMENT APPLIED INFORMATICS

Interactive Recommendation System

Diploma thesis
of

Gkiouleka Foteini

Thessaloniki, November 2019

INTERACTIVE RECOMMENDATION SYSTEM

Γκιουλέκα Φωτεινή

Πτυχίο Ηλεκτρολόγων μηχανικών, ΑΠΘ, 2016

Διπλωματική Εργασία

υποβαλλόμενη για τη μερική εκπλήρωση των απαιτήσεων του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΤΙΤΛΟΥ ΣΠΟΥΔΩΝ ΣΤΗΝ ΕΦΑΡΜΟΣΜΕΝΗ ΠΛΗΡΟΦΟΡΙΚΗ

Επιβλέπων/ουσα Καθηγητής/τρια
Ονοματεπώνυμο Καθηγητή/τριας

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την ηη/μμ/εεεε

Ονοματεπώνυμο 1	Ονοματεπώνυμο 2	Ονοματεπώνυμο 3
.....	.....	.....

Πληκτρολογήστε εδώ το ονοματεπώνυμο σας

.....

Περίληψη

Η χρήση των συστημάτων συστάσεων (recommender systems) έχει αρχίσει να γίνεται απαραίτητη στις μέρες μας και για τις ηλεκτρονικές επιχειρήσεις και για τους πελάτες, λόγω της ταχείας ανάπτυξης του Διαδικτύου σε συνδυασμό με το πρόβλημα της συσσώρευσης πληροφοριών. Τα συστήματα συστάσεων και εξατομίκευσης χρησιμοποιούνται ευρέως στο ηλεκτρονικό εμπόριο για να προτείνονται προϊόντα ή υπηρεσίες σε χρήστες (π.χ συστάσεις για αγορές, ανάγνωση ειδήσεων, συνδέσεις κοινωνικής δικτύωσης, ταινίες κ.α.). Ένα σύστημα συστάσεων παίρνει συνήθως ως είσοδο προσωπικές πληροφορίες από τον χρήστη, χρησιμοποιώντας έναν αλγόριθμο δημιουργεί τις συστάσεις και εμφανίζει κάποιες προτάσεις στον χρήστη. Ένας από τους πιο διαδεδομένους αλγορίθμους που χρησιμοποιείται είναι το συνεργατικό φιλτράρισμα (Collaborative filtering). Η βασική ιδέα είναι να εντοπιστούν οι χρήστες που μοιράζονται τα ίδια ενδιαφέροντα με τον ενδιαφερόμενο χρήστη στο παρελθόν, ενώ ο αλγόριθμος στηρίζεται στο ότι οι χρήστες που έχουν παρόμοιες προτιμήσεις, βαθμολογούν και αξιολογούν με παρόμοιο τρόπο. Οι τεχνικές αυτές συνήθως λαμβάνουν ένα σύνολο με τις βαθμολογίες των χρηστών του συστήματος και παράγουν προβλέψεις σχετικά με το τι χρειάζεται ένας χρήστης, βασιζόμενες στους πιο κοντινούς (ως προς τις προτιμήσεις) σε αυτόν χρήστες

Στόχος της διπλωματικής εργασίας είναι η σχεδίαση και υλοποίηση ενός διαδραστικού συστήματος συστάσεων που λειτουργεί με τη μέθοδο του συνεργατικού φιλτραρίσματος. Το σύστημα θα λαμβάνει ανάδραση από το χρήστη και θα προσαρμόζει ανάλογα τις προτάσεις που εμφανίζει. Θα δοθεί έμφαση στην σχεδίαση της διεπαφής χρήστη και στην παρουσίαση των αποτελεσμάτων έτσι ώστε να ευνοείται η διαισθητική αλληλεπίδραση με το χρήστη

Λέξεις κλειδιά
recommendation system, collaborative filtering, user feedback, Rochio algorithm, Django, Python, Angular, RestFul web services, Celery, Redis

Abstract

Due to the rapid development of the Internet coupled with the problem of information accumulation, the use of recommender systems has become essential nowadays for e-businesses and customers. Recommendations systems are widely used in e-commerce to recommend products or services to users (eg marketplace suggestions, news reports, social links, movies, etc.). A recommendation system usually receives personal information from the user, using an algorithm that creates the recommendations and displays some suggestions to the user. One of the most widely used algorithms is Collaborative filtering. The basic idea is to identify users who share the same interests with the concerned user, while the algorithm is based on the fact that users who have similar preferences rate and evaluate in a similar way. These techniques usually take a set of system user ratings and make predictions about what a user needs.

The aim of the thesis is to design and implement an interactive recommendation system that works with the collaborative filtering method. The system will receive feedback from the user and tailor its suggestions accordingly. Emphasis will be given to the design of the user interface and the presentation of results so as to promote intuitive interaction with the user

Keywords:
recommendation system, collaborative filtering, user feedback, Rochio algorithm, Django, Python, Angular, RestFul web services, Celery, Redis

Contents

List of images

List of tables

1.Introduction

1.1 Problem - importance of recommendation systems

During the last few decades, recommender systems have taken more and more place in our lives. Some real-world examples include suggestions for products on Amazon, friends' suggestions on social applications like Facebook, Twitter, LinkedIn and video recommendations on Youtube, news recommendations on Google News and so on. Recommender systems are really critical in some industries as they can generate a huge amount of income. Their intention is to facilitate users to find what they need effectively and immediately, creating a delightful user experience while driving incremental revenue.

The main goal of recommender system is to provide relevant suggestions to online users to make better decisions from many alternatives available over the web. A better recommendation system is directed more towards personalized recommendations by taking into consideration the user's feedback, user-demographic details etc. An important catalyst in building successful recommendation engines is the ease with which the web enables user's feedback about their likes and dislikes. For example users are able to provide a feedback by rating items, reviewing items etc. Other forms of feedback are not quite as explicit but are even easier to collect them. For example, the simple act of viewing a recommended item can be considered as endorsement for that item.

The biggest challenge of recommendation systems is to find a way to recommend relevant items, personalized on user's preferences. The recommendations need to be adjusted on real time, based on user's feedback.

1.2 Goals of thesis

This thesis aims to build an interactive recommender system. Some initial recommendations are returned to users, as they have been calculated by implementing a collaborative algorithm. Based on users' feedback the recommendations are updated on real time and more relevant recommended items are displayed to users.

A demo books recommendation application has been developed. Some online available datasets including users, books and ratings were used to fill

application's database and a collaborative filtering algorithm was implemented in first place to calculate users' recommended books. The recommended books for every user are stored into database. User is able to register/login in this application, and gets a list with all books and a list with recommended books. If user views a recommended book, it is considered as positive feedback, otherwise it is considered as negative feedback.

This thesis has the purpose of suggesting a possible implementation of Rochio algorithm in order to build an interactive real time recommendation system and all the required stages to develop it are described. Some testing cases were carried out on books application in order to see how the interactive recommendation systems reacts on different user's actions. The code of demo application can be found on github repository: https://github.com/fotein1/book_recommender

1.3 Thesis layout

The structure of thesis is the following:
Chapter 2 discusses the fundamental concepts of recommendation systems' theory, including collaborative filtering algorithms, content based algorithms etc. In chapter 3, we analyze the demo books recommendation application, the database structure, what technologies we used to build backend/frontend part of application, the scripts used to load the online datasets on application's database, the scripts used to calculate similar books and users' recommendations and the implementation of Rochio algorithm to readjust users' recommendations. In chapter 4, we run some user test cases in demo application and we analyze how recommendations changed based on user feedback. Last but not least, the final conclusions of thesis are mentioned in chapter 5.

2. Recommendations - Theory background

2.1 Definition

Recommendation systems are powerful tool and techniques to analyse huge volumes of data, especially product information and user information, and they provide relevant suggestions based on data mining approaches. In technical term, a recommendation engine problem is to develop a mathematical model which can predict how a user will like an item.

2.2 Goals of recommendation system

The goals of a recommendation system should be the following:

- Relevance: A recommender system should recommend items that are relevant to the user. Users are more likely to consume items they find interesting.
- Novelty: Recommender systems are more helpful when the recommended item is something that the user has not seen in the past.
- Serendipity: The recommended items are somewhat unexpected to the user. Serendipity is different from novelty in that the recommendations are surprising to the user, rather than simply something they did not know about before. It may often be the case that a particular user may only be consuming items of a specific type, although a latent interest in items of other types may exist which the user might themselves find surprising. Unlike novelty, serendipitous methods focus on discovering such recommendations.

- Increasing recommendation diversity: Recommender systems typically suggest a list of top-k items. When all these recommended items are very similar, it increases the risk that the user might not like any of these items. On the other hand, when the recommended list contains items of different types, there is a greater chance that the user might like at least one of these items.

2.3 Types of recommendation systems

The basic principle of recommendations is that significant dependencies exist between user- and item-centric activity. For example, a user who is interested in a sci fiction book is more likely to be interested in another sci fiction book, rather than a historical book. In many cases, various categories of items may show significant correlations, which can be leveraged to make

more accurate recommendations.These dependencies can be learned in a data-driven manner from the ratings matrix, and the resulting model is used to make predictions for target users. The larger the number of rated items that are available for a user, the easier it is to make robust predictions about the future behavior of the user. Different learning models can be used to accomplish this task.The basic models of recommender systems work with two kinds of data, which are the user-item interactions, such as ratings or buying behaviour and the attribute information about the users and the items like the textual profiles or relevant keywords. Methods that use the former are referred as collaborative filtering methods, whereas methods that use the latter are referred as content-based-recommender methods.

Collaborative filtering models

In this type of recommendation engine, filtering items from a large set of alternatives is done collaboratively by user's preferences. The term "collaborative filtering" refers to the use of ratings from multiple users in a collaborative way to predict missing ratings

The basic assumption in a collaborative filtering recommendation system is that if two users shared the same interests as each other in the past they will also have similar tastes in the future. There are two types of collaborative filtering recommender systems:

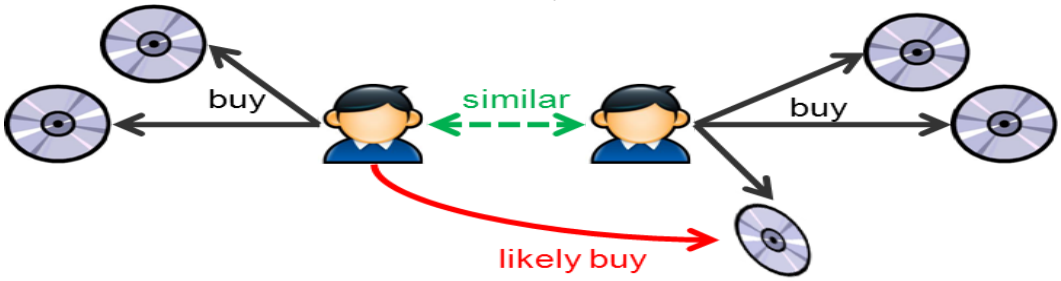
1. User based collaborative filtering:

In user based collaborative filtering, recommendations are generated by considering the preferences of similar users. If, for example user A and user B have similar books preferences and user A read Harry Potter books, which user B has not read yet then the idea is to recommend them to user B.

User based collaborative filtering is done in two steps:

- Identify similar users

- Recommend new items to an active user based on the rating given by similar users on the items not rated by the active users

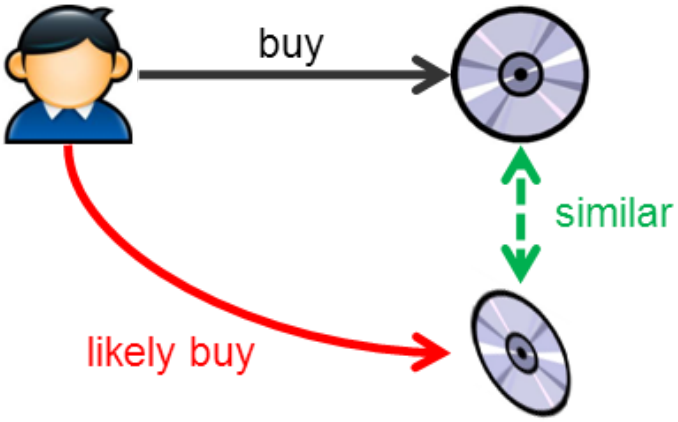


1. User based collaborative filtering diagram. Source https://dzone.com/articles/recommendation-engine-model

2. Item based collaborative filtering:

In item based collaborative filtering, recommendations are based on the similarity of items. Unlike user based collaborative filtering, we first find similarities between items and then recommend items which are similar to the items the active user has rated in the past. If, for example user A has rated Harry Potter books, we can recommend to him similar books.
The recommender systems are constructed in two steps:

- Calculate the item similarity based on the item preferences.
- Find the top not rated similar items to rated items by user and recommend them



2. Item based collaborative filtering diagram. Source http://2.bp.blogspot.com/-YEEM5PYuTAI/TmHGPmDYICI/AAAAAAAAAls/8VIm0-7PyYM/s1600/P1.png

The advantage of collaborative filtering system is that they are simple to implement and very accurate. However, they have their own set of limitations, such as the cold start problem which means that collaborative filtering systems fail to recommend to the first time users whose information is not available in the system.

Content based recommender systems

As the name indicates, a content based recommender system uses the content information of the items for building the recommendation model. The content based recommender system recommends items to users by taking the content or features of items and user profiles. The basic idea is that user interests can be modeled on the basis of properties (or attributes) of the items they have rated or accessed in the past.

Knowledge based recommender systems
In these recommendation systems, users interactively specify their interests, and the user specification is combined with domain knowledge to provide recommendations.

Cortext-aware recommendation system
User preferences may differ with the context, such as time of day, season, mood, place, location, and so on. A person at a different location, at a different time with different people may need different things. A context-aware recommender systems takes the context into account before computing or serving recommendations. This recommender system caters for the different needs of people differently in different contexts.

Hybrid recommender systems
This type of recommendation engines is built by combining various recommender systems to build a more robust system. For example, by combining collaborative filtering methods, when the model fails when new items don't have ratings, with the content-based systems, where the information about the items is available, new items can be recommended.

2.4 Collaborative filtering nearest neighborhood models

The standard method of Collaborative Filtering is known as Nearest Neighborhood algorithm. There are user-based CF and item-based CF. At

User-based CF, we have an n × m matrix of ratings, with user u_i, i = 1, ...n and item p□, j=1, …m. Now we want to predict the rating r_i□ if target user i did not watch/rate an item j. The process is to calculate the similarities between target user i and all other users, select the top X similar users, and take the weighted average of ratings from these X users with similarities as weights.

$$r_{ij} = \frac{\sum_k similarities(u_i,u_k) r_{kj}}{number\ of\ ratings}$$

While different people may have different baselines when giving ratings, some people tend to give high scores generally, some are pretty strict even though they are satisfied with items. To avoid this bias, we can subtract each user's average rating of all items when computing weighted average, and add it back for target user, shown as below.

$$r_{ij} = r_i + \frac{\sum_k similarities(u_i,u_k)(r_{kj} - r_k)}{number\ of\ ratings}$$

Two ways to calculate similarity are Pearson Correlation and Cosine Similarity.

Pearson Correlation: $Sim(u_i, u_k) = \frac{\sum_j (r_{ij} - r_i)(r_{kj} - r_k)}{\sqrt{\sum_j (r_{ij} - r_i)^2 \sum_j (r_{kj} - r_k)^2}}$

Cosine similarity: $Sim(u_i, u_k) = \frac{r_i \cdot r_k}{|r_i||r_k|} = \frac{\sum_j (r_{ij} - r_i)(r_{kj} - r_k)}{\sqrt{\sum_j (r_{ij} - r_i)^2 \sum_j (r_{kj} - r_k)^2}}$

Basically, the idea is to find the most similar users to your target user (nearest neighbors) and weight their ratings of an item as the prediction of the rating of this item for target user. Without knowing anything about items and users themselves, we think two users are similar when they give the same item similar ratings . Analogously, for **I**tem-based CF, we say two items are similar when they received similar ratings from a same user. Then, we will make prediction for a target user on an item by calculating weighted average of ratings on most X similar items from this user. One key advantage of Item-based CF is the stability which is that the ratings on a

given item will not change significantly over time, unlike the tastes of human beings.

Neighborhood methods have several advantages related to their simplicity and intuitive approach. Because of the simple and intuitive approach of these methods, they are easy to implement and debug. It is often easy to justify why a specific item is recommended, and the interpretability of item-based methods is particularly notable. Such justifications are often not easily available in many of the model-based methods discussed in later chapters. Furthermore, the recommendations are relatively stable with the addition of new items and users. It is also possible to create incremental approximations of these methods.

The main disadvantage of these methods is that the offline phase can sometimes be impractical in large-scale settings. The offline phase of the user-based method requires at least $O(m^2)$ time and space. This might sometimes be too slow or space-intensive with desktop hardware, when m is of the order of tens of millions. Nevertheless, the online phase of neighborhood methods is always efficient. The other main disadvantage of these methods is their limited coverage because of sparsity.

2.5 Relevance feedback Rochio algorithm

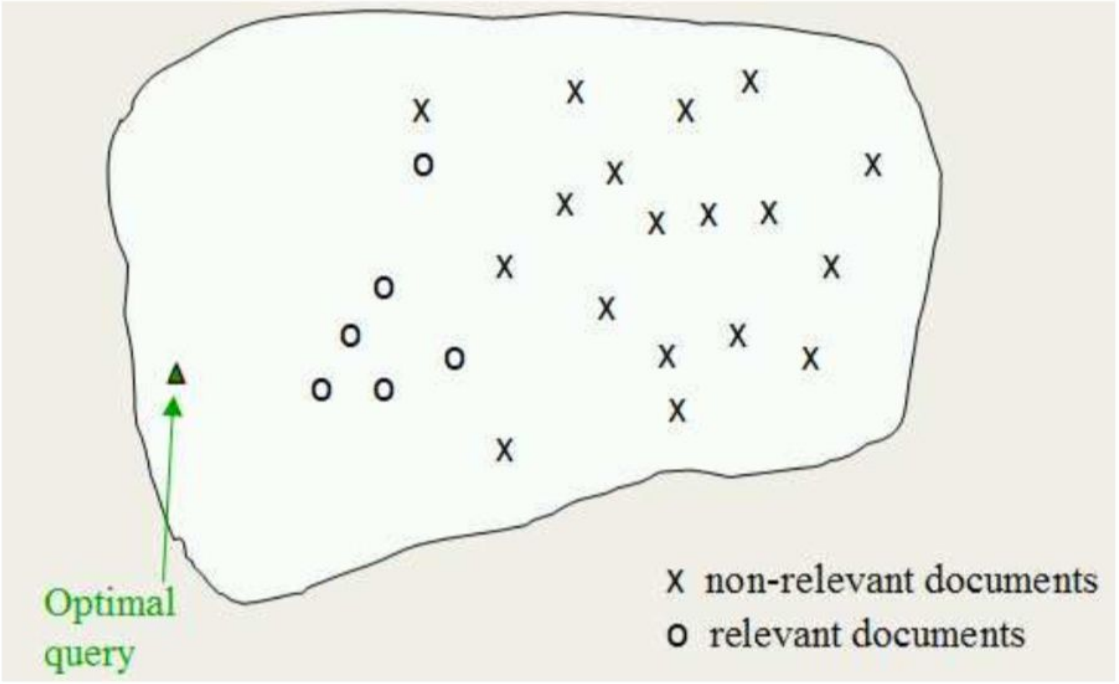
The idea of *relevance feedback* (RF) is to involve the user in the retrieval process so as to improve the final result set. In particular, the user gives feedback on the relevance of the results he gets. The relevance feedback is used widely by information retrieval. The art of IR is to get the relevant objects from a large collection of information objects (usually documents). A user formulates a query in which he tries to communicate his information need. The relevance feedback can be also implemented on recommendation systems.

The Rocchio Algorithm is the classic algorithm for implementing relevance feedback. It models a way of incorporating relevance feedback information into the vector space. The algorithm proposes using the modified query q_m

$$q_m = a\, q_0 + b \frac{1}{|D_r|} \sum_{d_j \in D_r} d_j - c \frac{1}{|D_{nr}|} \sum_{d_j \in D_{nr}} d_j$$

where $q0$ is the original query vector, *Dr* and *Dnr* are the set of known relevant and nonrelevant documents respectively, and *a*, *b*, and *c* are weights

attached to each term. These control the balance between trusting the judged document set versus the query: if we have a lot of judged documents, we would like a higher b and c. Starting from q_0, the new query moves you some distance toward the centroid of the relevant documents and some distance away from the centroid of the non relevant documents. This new query can be used for retrieval in the standard vector space model. We can easily leave the positive quadrant of the vector space by subtracting off a non relevant document's vector. In the Rocchio algorithm, negative term weights are ignored. That is, the term weight is set to 0.



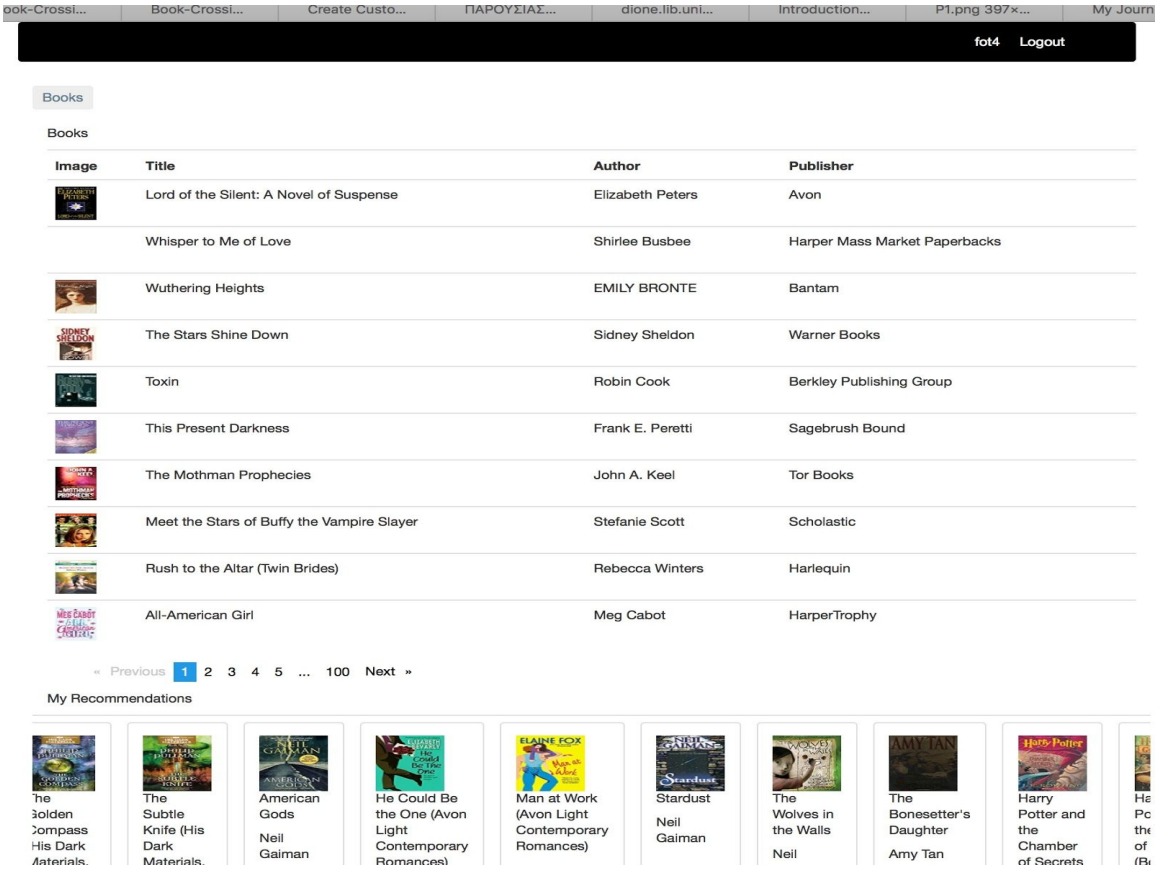
3.Optimize initial query with Rochio algorithm Source:

The relevance feedback with Rochio algorithm can be implemented on recommendation system with the following process:

- The system returns an initial set of retrieval results.
- If the user clicks some items from results, these items are considered as relevant while the rest of them are considered as not relevant.
- The system computes a better representation of the information need based on the user feedback.
- The system displays a revised set of retrieval results.

3. Books recommendations application

We are going to build a books recommendation application. User will be able to register/login in this application, he will get a list with all books and a list with recommended books. Based on user's feedback (eg click a recommended book) the recommended books are readjusted, implementing the Rochio algorithm. In this chapter we are going to analyze all the parts of application.



4.Books recommendation applications

3.1 Technologies

In order to build the above application the following technologies have been used:

Django

Django is a high level Python free and open-resource framework, which follows the model template view. It consists of an object-relational-mapper (ORM) that mediates between data models (defined as Python classes) and a relational database (Model), a system for processing HTTP requests with a web templating system (View), and a regular expression based URLdispatcher (Controller). Django also offers a lightweight and standalone web server for development and testing and powerful and flexible toolkit for building Web APIs(Django REST framework). We ran the backend part of application on Django server, we created the required database tables using django models and we also built the required RestFul web services with the Django framework.

Redis

Redis is an open source (BSD licensed), in-memory data structure store, used as a database, cache and message broker. It supports data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperloglogs, geospatial indexes with radius queries and streams. Redis was used to cache user's recommended books in order to reduce the database hits and improve the performance of application.

Celery

Celery is an asynchronous task queue/job queue based on distributed message passing.It is focused on real-time operation, but supports scheduling as well.The execution units, called tasks, are executed concurrently on a single or more worker servers using multiprocessing.Tasks can execute asynchronously (in the background) or synchronously (wait until ready). The celery queue was used on books recommendations application to run on background some heavy tasks, like readjusting user's recommendations by implementing the Rochio algorithm.

Angular

Angular is a Typescript-based open source software engineering framework used for building single-page web apps. AngularJS uses the Model-View-Controller(MVC) architecture, which is used in web app development. This type of architecture consists of: Model – the data structure that manages information and receives input from the controller, View – the representation of information and Controller – responds to input and interacts with the model. In the context of AngularJS, the model is the framework, while the view is HTML, and the control is JavaScript. We used

Angular framework to build the frontend part of books recommendations applications.

Python libraries/Packages

We used Pandas, a Python Data Analysis Library to read, analyse and convert csv files to datasets. NumPy, a fundamental package for scientific computing with Python. was used in many calculations. Last but not least the python scikit-learn package was used to implement some machine learning algorithms.

3.2 Datasets

For the purposes of this thesis the Book Crossing dataset has been used. This dataset has been compiled by Cai-Nicolas Ziegler in 2004, and it comprises of three tables for users, books and ratings. Explicit ratings are expressed on a scale from 1–10 (higher values denoting higher appreciation) and implicit rating is expressed by 0.

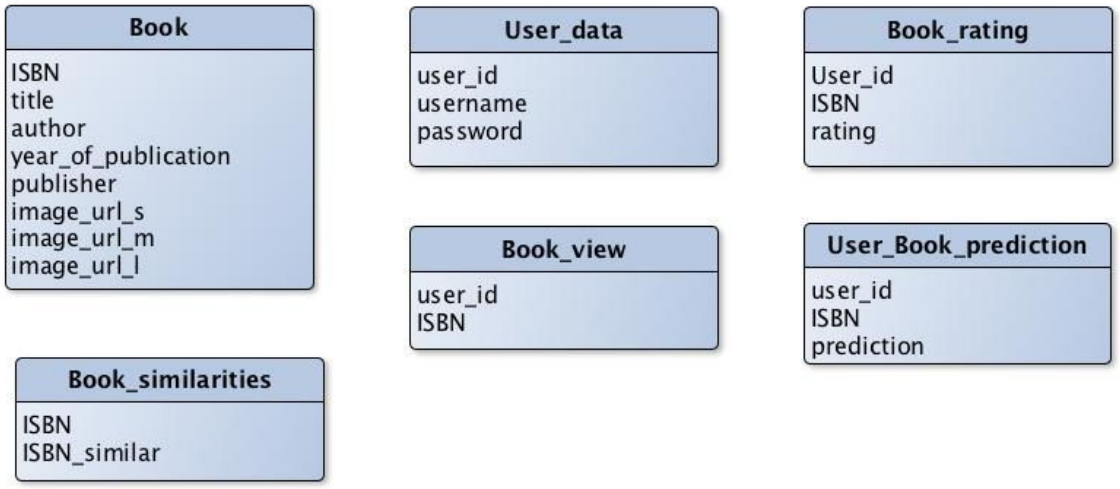
- BX-Book-Ratings.csv (1149780 items)
- BX-Books.csv (271360 items)
- BX-Users.csv (278858 items)

3.3 Database structure

We have opted to work with sqlite database. We have created the relevant django models to add the following tables in database. The django models are located in model.py file. For example the django model to create the Book table is the following:

```
class Book(models.Model):

 ISBN       = models.CharField(unique=True, primary_key=True, max_length=255)

  title      = models.CharField(max_length=255)

  author     = models.CharField(max_length=255)

  year_of_publication = models.CharField(max_length=255)

  publisher   = models.CharField(max_length=255)

  image_url_s = models.CharField(max_length=255)

  image_url_m = models.CharField(max_length=255)
```

image_url_l = models.CharField(max_length=255)



5.Database structure

Book: db table to store books data
User_data: db table to store user data
Book_rating; db table to store books ratings of users
Book_view: db table to store books user has viewed
User_Book_prediction: db table to store the predicted ratings for books user has not rated yet
Book_similarities: db table to store the similar books for every book

3.4 ETL(Extract Transform Load) process

After installing django framework, setting up django server and creating the above db tables we need to fill the local database with the online datasets. This process is known as ETL. ETL is short for ***e****xtract,* ***t****ransform,* ***l****oad*, three database functions that are combined into one tool to pull data out of one database(csv files in our case) and place it into another database.

- **Extract** is the process of *reading data* from a database. In this stage, the data is collected, often from multiple and different types of sources.

- **Transform** is the process of *converting the extracted data* from its previous form into the form it needs to be in so that it can be placed into another database. Transformation occurs by using rules or lookup tables or by combining the data with other data.

- **Load** is the process of *writing the data* into the target database.

To implement the ETL process we created some django scripts. These scripts are located in folder management/commands of django project. At these scripts we load the csv files, we read and convert them to datasets using pandas library commands and we store the data at the relevant database tables. We can execute these scripts through command line:

load books:

With this script we load books in our database table books. The script load_books.py has the following structure:

```
import pandas as pd
import numpy as np
from django.core.management.base import BaseCommand, CommandError
from django.contrib.auth.models import User
from books.models import Book

class Command(BaseCommand):
   def handle(self, *args, **options):
              books = pd.read_csv('data/BX-Books.csv', sep=';', error_bad_lines=False,
encoding="latin-1")
       books.columns = ['ISBN', 'bookTitle', 'bookAuthor', 'yearOfPublication', 'publisher',
'imageUrlS', 'imageUrlM', 'imageUrlL']
      books.apply(self.save_book_from_row, axis=1)

   def save_book_from_row(self, book_row):
      book = Book()
      book.ISBN = book_row['ISBN']
      book.title = book_row['bookTitle']
      book.author = book_row['bookAuthor']
      book.year_of_publication = book_row['yearOfPublication']
      book.publisher = book_row['publisher']
      book.image_url_s = book_row['imageUrlS']
      book.image_url_m = book_row['imageUrlM']
      book.image_url_l = book_row['imageUrlL']
      book.save()
```

We execute command python manage.py load_books to run the above script.

load_users:

In a similar way we load the users from users dataset on user_data database table with the script load_users.py.
We execute command python manage.py load_users to run the script.

load_ratings:
In a similar way we load the ratings from rating datasets on book_rating database table with the script load_ratings.py
We execute command python manage.py load_ratingss to run the script.

Load book_similarites:
To fill the user_book_similarities database table, we will need to calculate the similar books for every book. We decided to implement a collaborative item based filtering recommendation systems. The reason was that in our application the items(books) won't be changed so often. On the other hand new users could be registered and use the books recommendation system. As a result, it make more sense to us to calculate offline the similarities between books, store them in our database and access them when it is needed. Furthermore, generally item based algorithm has a better performance.

In order to find out the similar books, we used the item based neighbourhood models, we reduced the dataset size, taking into account users who have rated at least 100 books and books who have at least 100 ratings. Then we generated a user-term matrix based on rating table. Similarities need to be computed between the columns of rating matrix.

Before computing the similarities between the columns, each row of the ratings matrix is centered to a mean of zero. As in the case of user-based ratings, the average rating of each item in the ratings matrix is subtracted from each rating to create a mean-centered matrix.

This similarity is referred to as the adjusted cosine similarity because the ratings are mean- centered before computing the similarity value. We calculate the similar books and store them in our database with the following functions:

```
import pandas as pd
import numpy as np
from django.core.management.base import BaseCommand, CommandError
from django.contrib.auth.models import User
from books.models import Book_similarities
import sklearn.metrics as metrics
from sklearn.neighbors import NearestNeighbors
```

```
from scipy.spatial.distance import correlation, cosine
from sklearn.metrics import pairwise_distances
from sklearn.metrics import mean_squared_error
from math import sqrt
import sys, os
from contextlib import contextmanager

class Command(BaseCommand):
    metric = 'cosine'
    k = 6
    sample_limit = 10000
    user_id = 183

    def handle(self, *args, **options):
            ratings = pd.read_csv('data/BX-Book-Ratings.csv', sep=';', error_bad_lines=False,
encoding="latin-1")
        ratings.columns = ['userID', 'ISBN', 'bookRating']

        #Reduce the dataset size, take into account users who have rated at least 100 books
        #and books which have at least 100 ratings
        counts1 = ratings['userID'].value_counts()
        ratings_explicit = ratings[ratings['userID'].isin(counts1[counts1 >= 100].index)]
        counts = ratings_explicit['bookRating'].value_counts
          ratings_explicit = ratings_explicit[ratings_explicit['bookRating'].isin(counts[counts
>= 100].index)]

        #Generate a user-item ratings matrix from the ratings table.
                    ratings_matrix   =   ratings_explicit.pivot(index='userID',   columns='ISBN',
values='bookRating')
        for col in ratings_matrix:
            ratings_matrix[col].fillna(0, inplace=True)
        self.calculateSimilarBooks(ratings_matrix)


    """
        Calculate similarities of books
        @param obj self          The pointer of class
    """
    def calculateSimilarBooks(self, ratings_matrix):
        counter = 0
        for i in range(ratings_matrix.shape[1]):
            item_id = str(ratings_matrix.columns[i])
            similarities, indices= self.findksimilaritems(item_id, ratings_matrix)
            for i in range(0, len(indices.flatten())):
                if (similarities[i] != 0):
                    index = indices.flatten()[i]
                    similar_item_id = str(ratings_matrix.columns[index])
```

```
                self.saveSimilarBooks(item_id,similar_item_id)
            counter = counter + 1
            if (counter > self.sample_limit):
                break

    """
        Find similarities between items
        @param obj self          The pointer of class
        @param int item_id       The id of item
        @param arr ratings_matrix The matrix of ratings
    """
    def findksimilaritems(self, item_id, ratings_matrix):
        similarities=[]
        indices=[]
        ratings = ratings_matrix.T
        loc = ratings.index.get_loc(item_id)
        model_knn = NearestNeighbors(metric = 'cosine', algorithm = 'brute')
        model_knn.fit(ratings)
         distances, indices = model_knn.kneighbors(ratings.iloc[loc, :].values.reshape(1, -1),
n_neighbors = self.k + 1)
        similarities = 1-distances.flatten()
return similarities,indices

    """
        Save user book predictions into db

        @param obj self       The pointer of class
        @param int item_id    The id of item
        @param int similar_id The similar id
    """
    def saveSimilarBooks(self, item_id, similar_id):
        book_similarities = Book_similarities()
        book_similarities.ISBN = item_id
        book_similarities.ISBN_similar = similar_id
        book_similarities.save()
```

We execute command python manage.py calculate_similar_items to run the above script.

Load user books ratings predictions

The basic idea is to leverage the user's own ratings on similar items in the final step of making the prediction. For example, in a book recommendation system, the item peer group will typically be similar books. The ratings history of the same user on such books is a very reliable predictor of the

interests of that user. To predict the user's ratings for the books, we calculate the similar books as above. This calculation is a weighted sum. A weight function is a mathematical device used when performing a sum, integral, or average to give some elements more "weight" or influence on the result than other elements in the same set. The result of this application of a weight function is a weighted sum or weighted mean. The weighted mean is defined as:

$$\frac{\sum_{a \in A} f(a)w(a)}{\sum_{a \in A} w(a)}$$

This form of recommendation is analogous to "people who rate item X highly, like you, also tend to rate item Y highly, and you haven't rated item Y yet, so you should try it". We calculate the user's predictions as following:

```
class Command(BaseCommand):

    metric = 'cosine'
    k = 6
    sample_limit = 10000
    user_id = 183

    def handle(self, *args, **options):
            ……
            users = User_data.objects.all()
              for user in users:
                  self.recommendItem(user[‘user_id’], rating_matrix)
 """

    Predict user's rating for an item


    @param obj self          The pointer of class

    @param int user_id       The id of user

    @param int item_id       The id of item

    @param arr ratings_matrix A matrix of ratings

 """

  def recommendItem(self, user_id, ratings_matrix):

        if (user_id not in ratings_matrix.index.values):

            print('user is should be a valid integer')

            return

        else:
```

```
        counter = 0
        for i in range(ratings_matrix.shape[1]):
            if (ratings_matrix[str(ratings_matrix.columns[i])][user_id] != 0):
                item_id = str(ratings_matrix.columns[i])
                prediction = self.predict_itembased(user_id, item_id , ratings_matrix)
                self.save_user_book_prediction(user_id, item_id, prediction)

                counter = counter + 1
                if (counter > self.sample_limit):
                    break
"""
    Predict user's rating for an item

     @param obj self          The pointer of class
    @param int user_id       The id of user
    @param int item_id       The id of item
    @param arr ratings_matrix A matrix of ratings
 """
 def predict_itembased(self, user_id, item_id, ratings_matrix):
    prediction = wtd_sum = 0
    user_loc = ratings_matrix.index.get_loc(user_id)
    item_loc = ratings_matrix.columns.get_loc(item_id)
    similarities, indices= self.findksimilaritems(item_id, ratings_matrix)
    sum_wt = np.sum(similarities) - 1
    product = 1
    for i in range(0, len(indices.flatten())):
        if indices.flatten()[i] == item_loc:
            continu
    else:
            product = ratings_matrix.iloc[user_loc, indices.flatten()[i]] * (similarities[i])
            wtd_sum = wtd_sum + product
    prediction = int(round(wtd_sum/sum_wt))
```

```
if prediction <= 0:
    prediction = 1
elif prediction > 10:
    prediction = 10
return prediction
```

We execute command `python manage.py load_users_books_predictions` to run the script.

3.5 RestFul web services

We have built with Django API framework the required RestFul web services to serve client and return the required data. RESTful Web Services are basically REST Architecture based Web Services. In REST Architecture everything is a resource and a resource is accessed by a common interface using HTTP standard methods. Following four HTTP methods are commonly used in REST based architecture.

- GET − Provides a read only access to a resource.
- POST − Used to create a new resource.
- DELETE − Used to remove a resource.
- PUT − Used to update a existing resource or create a new resource.
- PATCH - Used for a partial update of a resource.

A RESTful web service usually defines a URI(Uniform Resource Identifier) and provides resource representation such as JSON. Clients just make a HTTP request at the specified URI, with the required headers and payload body and get the response.RESTful web services are lightweight, highly scalable and maintainable and are very commonly used to create APIs for web-based applications. For the books recommendation application we created the required web services for the following functionalities:

- Register user
- Login user
- Get book list
- Get recommended books
- Rate a book

- View a book

All the web services are running on Django server and they are accessible at base url http://127.0.0.1:8000.

The details of every web service are mentioned at the following table:

1. *Restful web services definition*

Functionality	**URL**	**method**	**Request body**	**Response**

login user	/api/sessions	POST	{
			‘username’: ‘’
			‘password’: ’
			}
				status_code**:** 201
				{
				user_id’:’’
				}

Logout user	/api/sessions	DELETE	{
			user_id’:’’
			}
				status code: 204

Register user	/api/accounts	POST	{
			‘username’: ‘’
			‘password’: ’
			}
				status code: 201

Get books list	/api/books?page=1	GET		status code: 200

				[{ 'ISBN': ‘’, 'title': ‘’, 'author': ‘’, 'year_of_publication': ‘’, 'publisher': ‘’, 'image_url_s': ‘’, 'image_url_m' : ‘’, 'image_url_l': ‘’ }]
Get recommended books	/api/books-recommendations/users/1	GET		status code; 200 [{ 'ISBN': ‘’, 'title': ‘’, 'author': ‘’, 'year_of_publication': ‘’, 'publisher': ‘’, 'image_url_s': ‘’, 'image_url_m' : ‘’, 'image_url_l': ‘’ }]

Rate a book	/api/books-rates	POST	{
			user_id: ‘’,
			ISBN: ‘’,
			rating: ‘’
			}
				status code: 201
				{
				user_id: ‘’,
				ISBN: ‘’,
				rating: ‘’
				}

View a book	/api/books-views	POST	{
			‘user_id’: ‘’,
			‘ISBN’: ‘’
			}
				status code 201

Every web service in Django consists of url definition at url.py, the controller function at view.py, the model function at model.py and a serializer object defined at serializer.py, used mainly by GET methods where a json object with resource's details is returned to user. For example the login web service is built widh django framework as following:

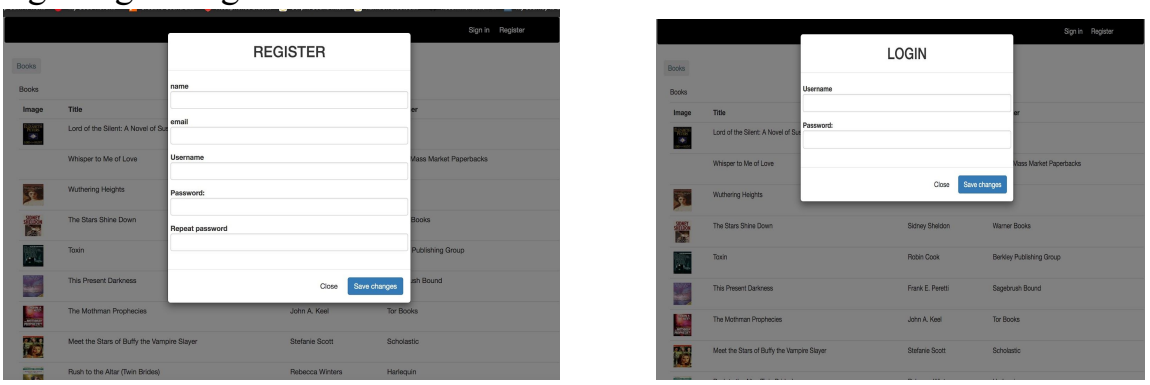
```
// urls.py
sessions_urls = [
  url(r'^$', sessionAPIView.as_view(), name='sessions')
]

urlpatterns = [
  url(r'^sessions', include(sessions_urls)),
]

// views.py
class sessionAPIView(APIView):
    def post(self, request):
        body_unicode = request.body.decode('utf-8')
        body = json.loads(request.body)
        username = body['username']
        password = body['password']

        try:
            user = User_data.objects.get(username=username, password=password)
        except:
            return HttpResponseNotFound( "User not found")

        request.session['member_id'] = user.user_id

        return Response({'user_id': user.user_id})
```

3.6 Frontend client

The frontend part of books recommendation application was built with angular framework. Angular is a platform and framework for building client applications in HTML and TypeScript. Angular is written in TypeScript. It implements core and optional functionality as a set of TypeScript libraries that are imported into app. An Angular app is defined by a set of NgModules, the basic building blocks. An app always has at least a *root module* that enables bootstrapping, and typically has many more *feature modules.*
The following modules have been created for the books recommendation app:

- nav-menu

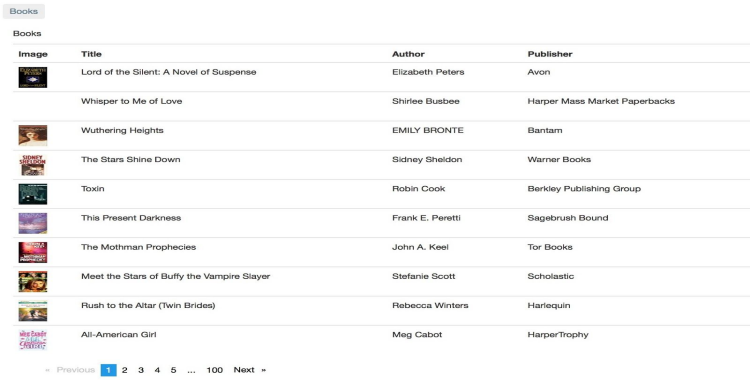
This module is responsible for the navigation menu. It consists of nav-menu.css file, nav-menu.html template and nav-menu.ts typescript file where the navigation menu component is defined. This component uses the login/logout/register functionalities from session service.



6. Login/Register view of books recommendations app

- books

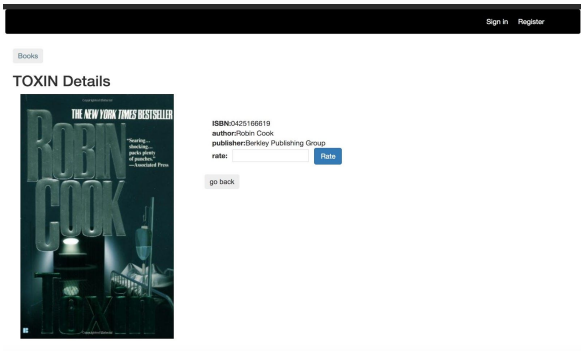
This module is responsible for the books list. It consists of books.css file, books.html template and books.ts typescript file where the books component is defined. This component calls the get list of books functionality of books service.



7. View of books list

- book-detail

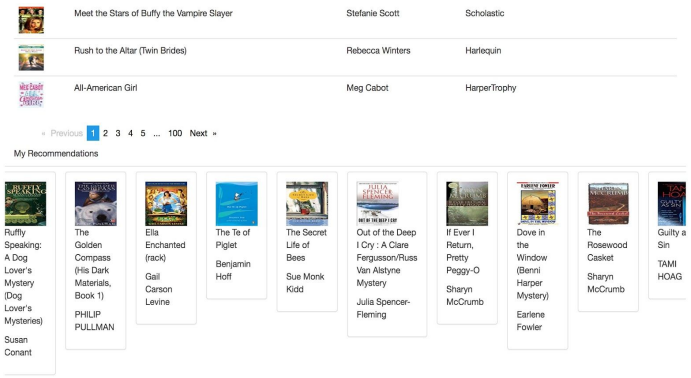
This module is responsible for the detailed view of book. It consists of book-detail.css, book-detail.html and book-detail.ts where the book-details component is defined. This component calls the functionalities get details of book, view book of books service and rate book from rate book service .



8. View of books details

- book-recommendations

This module is responsible for the list of recommended books. It consists of book-recommendations.css, book-recommendations.html and book-recommendations.ts where the book-details component is defined. This component calls the functionalities get recommended books of recommendations service.

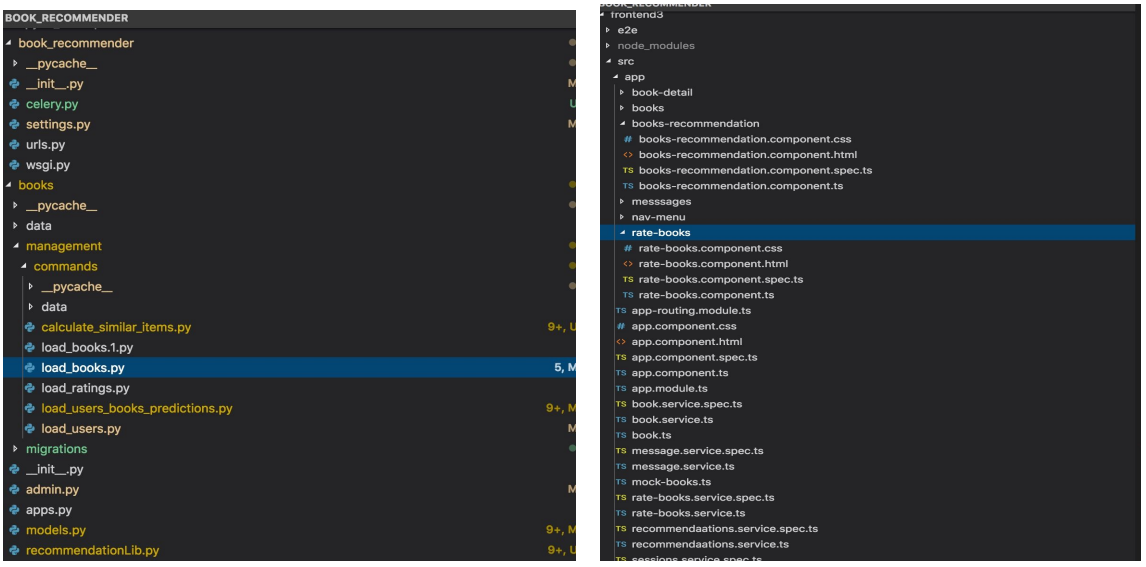


9. View of recommendations list

The frontend part of books recommendation app is running on npm server at url: http://localhost:4200/books

3.7 Project structure

The following images describe the folder structure of application's backend and frontend part.



10. Project structure

3.8 Design of recommendation system

Get recommendations

In this stage we have stored into our database the users' predicted ratings for books they have not rated yet. As mentioned earlier, a web service has been created to return the recommendations to user. At views.py, the controller function of web service is defined. We store in redis cache for every user a flag which expires every hour and the recommendations for this user. If the flag in cache has not expired yet, the cached recommendations are returned to user if exist. Otherwise, the user's recommendations are extracted again from database.

```
// views.py

class userBookRecommendationsAPIView(APIView):
    def get(self, request, user_id, *args, **kwargs):
        recommendations = []
        recommendation_expire_cache_name = 'recommendation_expire_' + user_id
        reocmmendations_user_cache_name  = 'recommendations_' + user_id

        recommendations = cache.get(reocmmendations_user_cache_name)
       if recommendation_expire_cache_name in cache and reocmmendations_user_cache_name in cache and
recommendations:
            recommendations = cache.get(reocmmendations_user_cache_name)
        else:
            recommendations = []
            recommendations = getUserPredictions(user_id, recommendations, False)

            try:
```

```
            find_recommended_items.delay(user_id, recommendations)
        except:
            return HttpResponseNotFound("Recommendations not found")

        return Response(recommendations)

    if not recommendations:
        recommendations = getUserPredictions(user_id, recommendations, True)

    return Response(recommendations)
```

All the functions used by the recommendation system are included in a library file recommendationLib.py. If predictions for the logged in user are stored into our db, the recommended books are returned to him, as they have been calculated by the above scripts. The results are returned ordered by prediction value. If there are not predictions for the logged in user into database, the top rated books are returned as recommendations.

```
// recommendationLib.py
"""
    Get user predictions

    @param int user_id          The id of user
    @param int recommendations  An array with recommendations
    @param boo get_default      True to get the default recommended items

    @return arr                 An array with predictions
"""
def getUserPredictions(user_id, recommendations, get_default)

prediction = User_Book_prediction.objects.filter(user_id=user_id).order_by('-prediction')
    if not predictions or get_default == True:
        try:
                                            ratings    =    Book_rating.objects.values('ISBN').annotate(score    =
Sum('rating')).order_by('-score')[:100]
            for rating in ratings:
                result = rating
                try:
                    book   = Book.objects.get(ISBN=result['ISBN'])
                    result['book'] = bookSerializer(book, many=False).data
                except:
                    result['book'] = ''

                recommendations.append(result)
        except:
            return HttpResponseNotFound("Recommendations not found")

    if predictions and get_default == False:
        for prediction in predictions:
            result = bookUserPredictionSerializer(prediction, many=False).data
            try:
                book   = Book.objects.get(ISBN=result['ISBN'])
```

```
            result['book'] = bookSerializer(book, many=False).data
        except:
            result['book'] = ''

        recommendations.append(result)
    return recommendations
```

User feedback

As we can see from the code of above controller function, when the cached flag for the logged in user expires after an hour, the user's predictions are extracted again from database while a task is triggered and added on celery queue to adjust user's recommendations. Based on user's feedback(books which user has viewed) the predictions are adjusted and the updated recommended items are set in cache. The celery task is defined at task.py file:

```
// task.py

@shared_task

def find_recommended_items(user_id, recommendations):

    recommendation_expire_cache_name = 'recommendation_expire_' + user_id

    reocmmendations_user_cache_name  = 'recommendations_' + user_i

    recommendations = getUserPredictions(user_id, recommendations, False)

    book_views = getUserBookViews(user_id)

    adjustPredictionsByUserFeedback(recommendations, book_views, user_id)

    recommendations = getUserPredictions(user_id, recommendations, False)

    cache.set(recommendation_expire_cache_name, True, timeout=10)

    cache.set(reocmmendations_user_cache_name, recommendations, timeout=10)
```

In order to adjust the recommendations for the logged in user, we get the books user has viewed, by calling the relevant function defined at recommendationLib.py.

```
// recommendationLib.py

'''Get user book views

    @param int user_id  The id of user

    @return arr             An array with book views
```

```
"""
def getUserBookViews(user_id):
    book_views = []
    try:
        user_book_views = Book_view.objects.filter(user_id=user_id)
    except:
     return HttpResponseNotFound("User has not viewed books yet")
    if user_book_views:
        for view in user_book_views
            result = bookViewSerializer(view, many=False).data
            book_views.append(result['ISBN']
        return book_views
    return HttpResponseNotFound("User has not viewed books yet")
```

Afterwards, we find the similar books of every recommended book viewed by user. We also find the similar books of recommended books which have not been viewed by user.

```
"""
    Adjust predictions based on user feedback
     @param arr recommendations An array with recommendations
    @param arr book_views      An array with book views
    @param int user_id         The user id
"""
def adjustPredictionsByUserFeedback(recommendations, book_views, user_id):
    ISBN_positive_similar_ids = []
    ISBN_negative_similar_ids = []
    for recommendation in recommendations:
     if recommendation['ISBN'] in book_views:
```

```
            try:
                similar_books = Book_similarities.objects.filter(ISBN=recommendation['ISBN'])
                for book in similar_books:
                    if book['ISBN_similar'] not in ISBN_positive_similar_ids:
                        ISBN_positive_similar_ids.append(book['ISBN_simila
                    ISBN_positive_similar_ids.append(recommendation['ISBN'])
            except:
                ISBN_positive_similar_ids.append(recommendation['ISBN'])
            if recommendation['ISBN'] not in book_views:
            try:
                similar_books = Book_similarities.objects.filter(ISBN=recommendation['ISBN'])
                for book in similar_books:
                    book = bookSimilaritiesSerializer(book, many=False).data
                    if book['ISBN_similar'] not in ISBN_negative_similar_ids:
                        ISBN_negative_similar_ids.append(book['ISBN_similar'])
                ISBN_negative_similar_ids.append(recommendation['ISBN'])
            except:
                ISBN_negative_similar_ids.append(recommendation['ISBN']
        insert_predictions = False
    user_predictions = User_Book_prediction.objects.filter(user_id=user_id).order_by('-prediction')[:37]
    if not user_predictions:
        insert_predictions = True
    if  ISBN_positive_similar_ids:
        savePositivePredictionsOfRecommendedItems(ISBN_positive_similar_ids, user_id, user_predictions)
    if ISBN_negative_similar_ids:
                        saveNegativePredictionsOfRecommendedItems(ISBN_negative_similar_ids,  user_id,
user_predictions)
```

After having separated books into ISBN_positive_similar_ids and ISBN_negative_similar_ids arrays, we call the relevant functions to update

their prediction value. When user clicks and views a recommended book, it is considered as positive feedback, while if user has not clicked the recommended book, it is considered as negative feedback. We use the algorithm Rochio to readjust the recommendations. We are having a vector with the original predictions (books and predicted ratings). We have stored in database the books every user has viewed and we also store the similar books as they have been calculated by the above scripts. Taking into account the books user has viewed and also their similar books we calculate the new predicted rating as following:

$$positive\_prediction = intitial\_prediction + \frac{0.75*(prediction_1+....+prediction_n).}{n}$$

$$negative\_prediction = intitial\_prediction - \frac{0.15*(prediction_1+....+prediction_n).}{n}$$

The rochio weight is calculated with the following function in recommendationLib.py:

```
"""
   Calculate weight = rochio_weight*(sum_of_predictions/predicted_itemss
   @param arr recommendations An array with recommendations
   @param arr book_views      An array with book views
"""
def calculateWeight(rochio_weight, predicted_items_counter, predictions_sum):
   return rochio_weight*predictions_sum/predicted_items_counter
"""
```

If user has predictions stored into database, we calculate their new prediction value with the above function and update their value into database. If user does not have any initial prediction stored into database, we get the books viewed by user and their similar books and we insert them into user_prediction database with a prediction equal with '1'. Books not viewed by user and their similar are inserted itno db with a prediction value '0'. The function to update the user's predictions with the positive weight is the following:

```
    Adjust the predicted items with a positive weigh

    @param arr ISBN_positive_similar_ids An array with positive similar ids

    @param int user_id              The user id

@param boo insert_predictions  True, to insert the predictions into db, false to update the existed predictions

"""

def savePositivePredictionsOfRecommendedItems(ISBN_positive_similar_ids, user_id, insert_predictions):

    predicted_items_counter = 0

    predictions_sum = 0

   if insert_predictions is False:

        predictions = User_Book_prediction.objects.filter(ISBN__in=ISBN_positive_similar_ids)

        if predictions:

            for prediction in predictions:

                prediction = bookUserPredictionSerializer(prediction, many=False).data

                predicted_items_counter = predicted_items_counter + 1

                predictions_sum = predictions_sum + prediction['prediction']


positive_weigh=calculateWeight(ROCHIO_POSITIVE_WEIGHT,predicted_items_counter, predictions_sum)

        if positive_weight < 0 or positive_weight > 10:

            positive_weight =  10


User_Book_prediction.objects.filter(ISBN__in=ISBN_positive_similar_ids).update(prediction=F('prediction') + abs(positive_weight))

    if insert_predictions is True::

        for item_id in ISBN_positive_similar_ids:

            positive_weight = 1

            user_book_predicition = User_Book_prediction()

            user_book_predicition.user_id = user_id
```

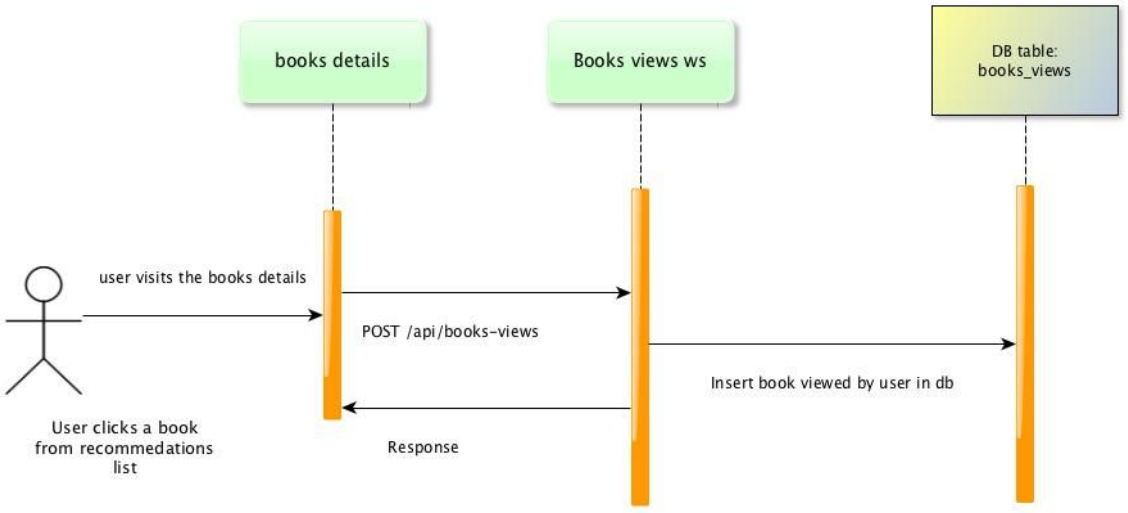
```
            user_book_predicition.ISBN = item_id

            user_book_predicition.prediction = positive_weight

            user_book_predicition.save()
```

We follow a similar process to update the prediction value of items with the negative value:

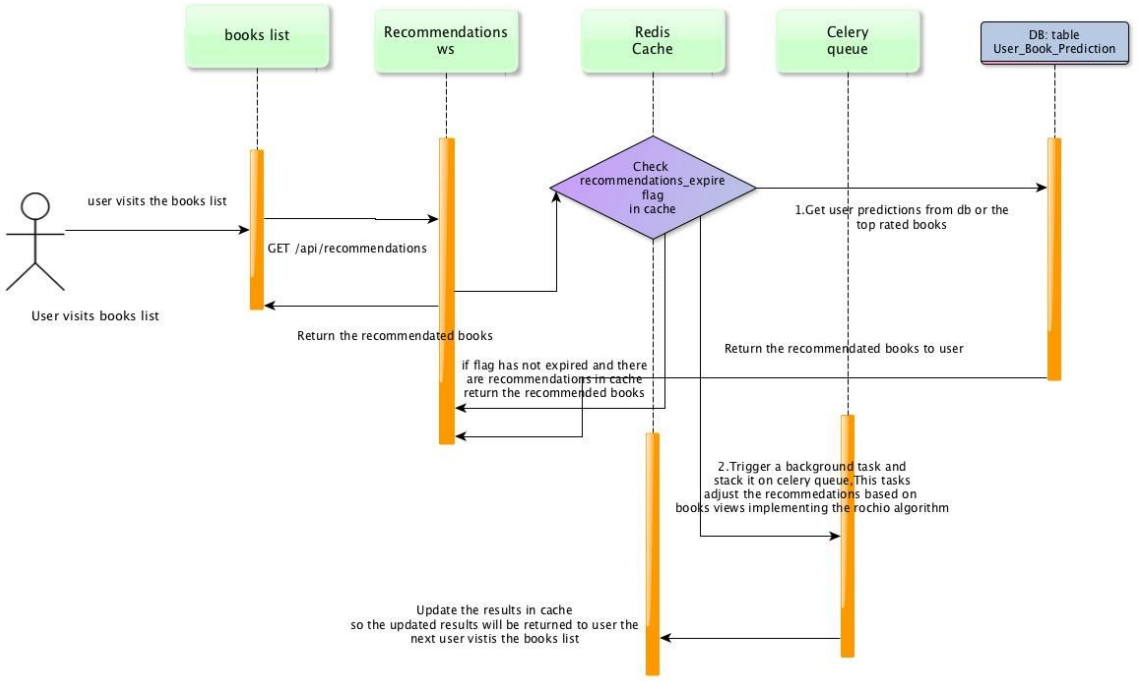
```
"""

    Adjust the predicted items with a negtive weight

     @param arr ISBN_negative_similar_ids An array with negative similar ids

    @param int user_id                          The user id

    @param boo insert_predictions               Flag to insert predictions

"""

def             saveNegativePredictionsOfRecommendedItems(ISBN_negative_similar_ids,              user_id,
insert_predictions):

    predicted_items_counter = 0

    predictions_sum = 0

    if insert_predictions is False:

    predictions = User_Book_prediction.objects.filter(ISBN__in=ISBN_negative_similar_ids)

    if predictions:

        for prediction in predictions:

            prediction = bookUserPredictionSerializer(prediction, many=False).data

            predicted_items_counter = predicted_items_counter + 1

            predictions_sum = predictions_sum + prediction['prediction']

       negative_weight    =    calculateWeight(ROCHIO_NEGATIVE_WEIGHT,    predicted_items_counter,
predictions_sum)

       if negative_weight < 0 or negative_weight > 10:

            negative_weight =


User_Book_prediction.objects.filter(ISBN__in=ISBN_negative_similar_ids).update(prediction=F('predicti
on') - abs(negative_weight))
```

```
if insert_predictions is True:
    for item_id in ISBN_negative_similar_ids:
        positive_weight = 0
        user_book_predicition = User_Book_prediction()
        user_book_predicition.user_id = user_id
        user_book_predicition.ISBN = item_id
        user_book_predicition.prediction = positive_weight
        user_book_predicition.save()
```

After one hour, we repeat the above process and we adjust again users' predictions. The following diagrams explains the whole process:



11. Workflow when user visits the details of book view.



12. Workflow when user visits the books list

4. Test cases of books recommendation system

After developing the books recommendation system, we carried out some test cases scenarios to see how the recommendation system reacts on different user's action:

1st scenario:Existed user with user_id=11676(having data for him into database) clicks some recommended items

When the user signs in books recommendation application, he gets a list with the initial recommendations, as they have been calculated with collaborative filtering algorithm.

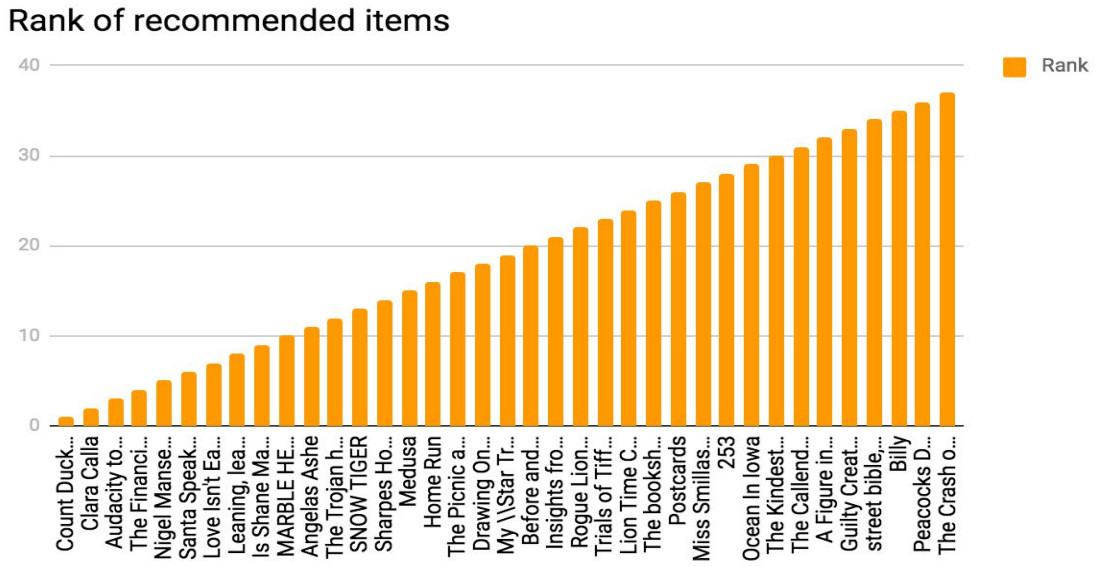
2. *1st test case: table with initial recommendations*

Rank of recommended items	**ISBN**	**Title**	**Author**
1	0001944711	Count Duckula: Vampire Vacation	Maureen Spurgeon
2	0002005018	Clara Calla	Richard Bruce

			Wright
3	0002118580	Audacity to believe	Sheila Cassidy
4	0002176432	The Financial post selects the 100 best companies to work for in Canada	Eva Inne
5	0002184974	Nigel Mansell My Autobiograph	Nigel Mansell
6	0002250810	Santa Speaks: The Wit and Wisdom of Santas Across the Nation	Michael Patrick Collins
7	0002251485	Love Isn't Easy (Passionate Peanuts)	Charles M. Schulz
8	0002255014	Leaning, leaning over water: A novel in ten stories	Frances Itani
9	0002258560	Is Shane MacGowan Still Alive?	Tim Bradford
10	000225929X	MARBLE HEART	Gretta Mulrooney
11	0002558122	Angelas Ashes	Frank Mccourt
12	0006131409	The Trojan horse	Hammond Innes
13	0006143199	SNOW TIGER	Bagley Desmond
14	0006171982	Sharpes Honor	Bernard Cornwell
15	0006176747	Medusa	Hammond Innes
16	0006177492	Home Run	Gerald Seymour
17	0006363121	The Picnic and Suchlike Pandemonium	Gerald Durrell

18	0006366023	Drawing On the Right Side of the Brain	Betty Edwards
19	0006379702	My \\Star Trek\\\" Memories\"	William Shatner
20	000648302X	Before and After	Matthew Thomas
21	0006492347	Insights from the Outfield (Peanuts at Work & Play)	Charles Schulz
22	0006498493	Rogue Lion Safaris	Simon Barnes
23	0006512062	Trials of Tiffany Trott	Isabel Wolff
24	0006512208	Lion Time Collected Stories 6	Robert Silverberg
25	0006543545	The bookshop	Penelope Fitzgerald
26	0006546684	Postcards	E Annie Proulx
27	0006547834	Miss Smillas Feeling for Snow	Peter Hoeg
28	0006550789	253	Geoff Ryman
29	0006551076	Ocean In Iowa	Peter Hedges
30	0006552390	The Kindest Use a Knife	Vanessa Jones
31	0006729835	The Callender Papers	Cynthia Voight
32	0006928323	A Figure in Hiding (The Hardy Boys)	Franklin W. Dixon
33	0007106572	Guilty Creatures	Sue Welfar
34	0007107900	street bible, the	Robert Lacey
35	0007110928	Billy	Pamela Stephenso
36	0007118465	Peacocks Dancing	Sharon Maas
37	0007139411	The Crash of	Patrick Ness

		Hennington	



13.1st test case: Diagram with initial recommendations for user

User clicks the following books from recommendations list:

3. 1st test case: table with viewed books

ISBN	**Title**	**Author**
0002176432	The Financial post selects the 100 best companies to work for in Canada	Eva Inne
0002005018	Clara Calla	Richard Bruce Wright
0002184974	Nigel Mansell My Autobiograph	Nigel Mansell
0002250810	Santa Speaks: The Wit and Wisdom of Santas Across the Nation	Michael Patrick Collins
0002258560	Is Shane MacGowan Still Alive?	Tim Bradford
000225929X	MARBLE HEART	Gretta Mulrooney

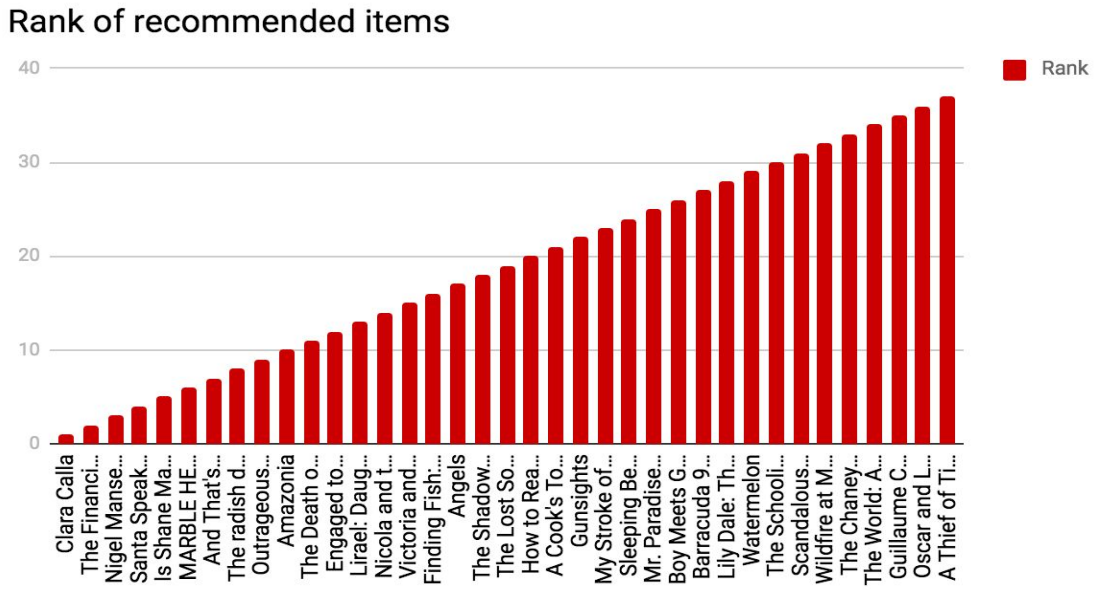
After an hour, we take into consideration the books user has viewed and we adjust the recommended books, implementing the rochio algorithm.

4. 1st test case: table with adjusted recommendations with rochio

Rank of recommended items	**ISBN**	**Title**	**Author**
1	0002005018	Clara Calla	Richard Bruce Wright
2	0002176432	The Financial post selects the 100 best companies to work for in Canada	Eva Inne
3	0002184974	Nigel Mansell My Autobiograph	Nigel Mansell
4	0002250810	Santa Speaks: The Wit and Wisdom of Santas Across the Nation	Michael Patrick Collins
5	0002258560	Is Shane MacGowan Still Alive?	Tim Bradford
6	000225929X	MARBLE HEART	Gretta Mulrooney
7	0030491916	And That's My Final Offer! (His A Doonesbury book)	G.B. Trudeau
8	0030686784	The radish day jubile	Sheilah B Bruce
9	0043720455	Outrageous fortune	Susan Kelly
10	0060002492	Amazonia	James Rollins
11	006000438X	The Death of Vishnu: A Novel	Manil Suri
12	006000469X	Engaged to Die : A Death on Demand Mystery (Hart,	Carolyn Har

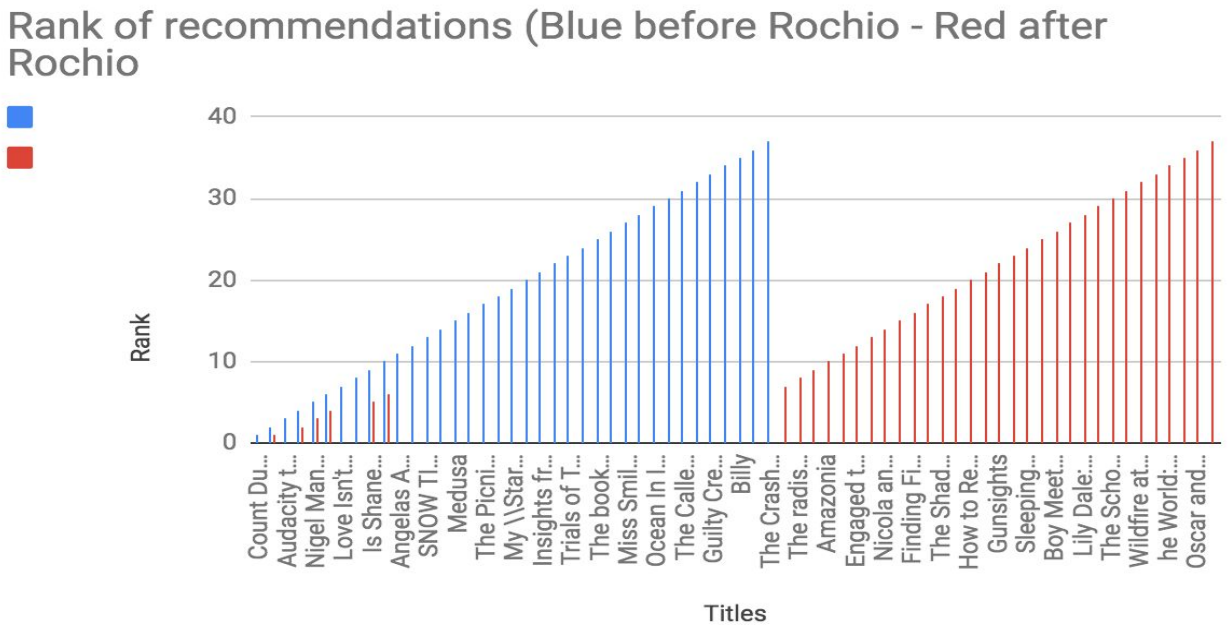
		Carolyn G)	
13	0060005424	Lirael: Daughter of the Clayr	Garth Nix
14	0060005521	Nicola and the Viscount (An Avon True Romance)	Meg Cabot
15	006000553X	Victoria and the Rogue (An Avon True Romance)	Meg Cabot
16	0060007788	Finding Fish: A Memoi	Antwone Q. Fisher
17	0060008032	Angels	Marian Keye
18	0060008369	The Shadows of Power	James W. Huston
19	0060009012	The Lost Son: A Life in Pursuit of Justice	Bernard B. Keri
20	006000942X	How to Read Literature Like a Professor : A Lively and Entertaining Guide to Reading Between the Lines	Thomas C. Foster
21	0060012781	A Cook's Tour : Global Adventures in Extreme Cuisines	Anthony Bourdain
22	0060013508	Gunsights	Elmore Leonard
23	0060014040	My Stroke of Luc	Kirk Douglas
24	0060083263	Sleeping Beauty (Margolin, Phillip)	Phillip Margoli
25	0060083956	Mr. Paradise: A Novel	Elmore Leonard

26	0060085452	Boy Meets Girl	Meg Cabo
27	0060086637	Barracuda 945	Patrick Robinso
28	0060086661	Lily Dale: The True Story of the Town that talks to the Dead	Christine Wicker
29	0060090367	Watermelon	Marian Keyes
30	0060090626	The Schooling of Claybird Catts	Janis Owen
31	0060092653	Scandalous Again	Christina Dodd
32	0060093579	Wildfire at Midnight	Mary Stewart
33	0060104910	The Chaneysville Incident: A Novel	David Bradley
34	0060155027	he World: An Illustrated History	Geoffrey Parker
35	0060156961	Guillaume Chequespierre and the Oise Salon: An Anthology	John Hulme
36	0060159081	Oscar and Lucind	Peter Carey
37	0060159383	A Thief of Time: A Novel (Harper Novel of Suspense)	Tony Hillerman



14.1st test case: Diagram with adjusted recommendations for user

As we can see from the above table and diagrams, the clicked recommended books and their similar books are ranked higher on the list, such as the "Clara Calla", "The Financial post selects the 100 best companies to work for in Canada", "The Financial post selects the 100 best companies to work for in Canada", "Santa Speaks: The Wit and Wisdom of Santas Across the Nation" etc.. On the other hand, the previous recommended books which have not been clicked by the user, they have been replaced by new recommendations as a negative wight has been added on their initial prediction value. For example some new books are appeared on recommendations list like Outrageous fortune, Amazon etc.



15.1st test case: Comparative diagram of recommendations before/after implementing rochio.

2nd scenario:Existed user with userid=276680 (having data for him into database) clicks all recommended items

When user signs in books recommendation application, he gets a list with the initial recommendations, as they have been calculated with collaborative filtering algorithm.

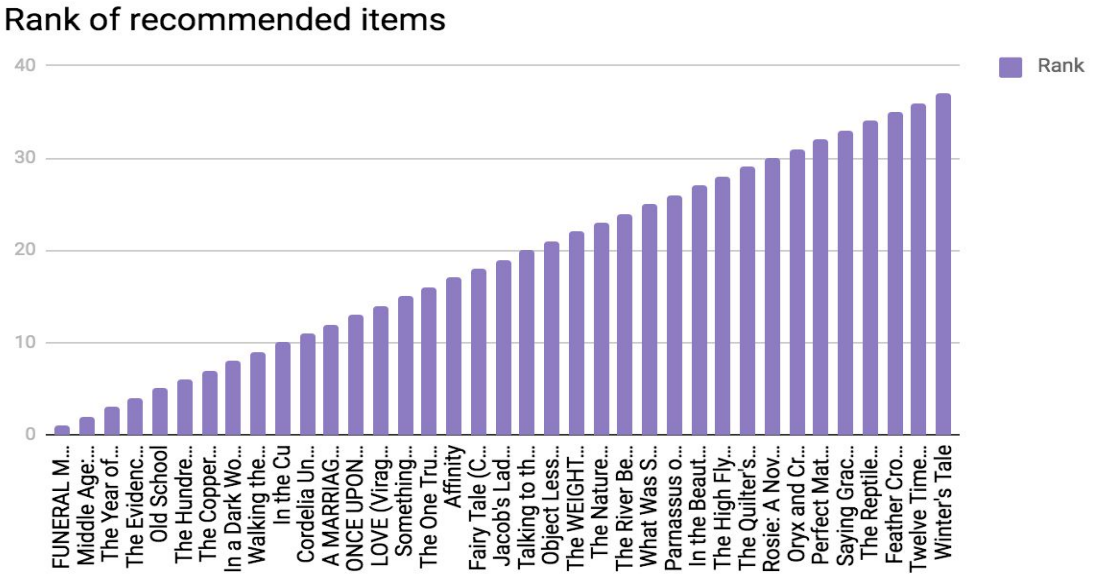
5. 2nd test case: table with initial recommendations

Rank of recommended items	**ISBN**	**Title**	**Author**
1	0020236107	FUNERAL MAKERS	Cathie Pelletier
2	0060934905	Middle Age: A Romance	Joyce Carol Oates
3	0312280696	The Year of Jubilo : A Novel of the Civil War	Howard Bahr
4	0316095575	The Evidence Against Her: A	Robb Forman Dew

		Novel	
5	0375401466	Old School	TOBIAS WOLF
6	0375701524	The Hundred Secret Senses	Amy Tan
7	0385307756	The Copper Beech	Maeve Binchy
8	038572117X	In a Dark Wood: A Novel	Amanda Craig
9	0425169626	Walking the Dead	Scott Spence
10	0452281296	In the Cut	Susanna Moore
11	0670880973	Cordelia Underwood: Or the Marvelous Beginnings of the Moosepath League	Van Reid
12	0671516949	A MARRIAGE MADE AT WOODSTOCK	Cathie Pelletier
13	0671724479	ONCE UPON A TIME ON THE BANKS : ONCE UPON A TIME ON THE BANKS	Cathie Pelletier
14	0671883925	LOVE (Virago Modern Classics)	Elizabeth Von arnim
15	089587167X	Something Blue	Jean Christopher Spaugh
16	1402201435	The One True Ocean	Sarah Beth Martin
17	1573221562	Affinity	Sarah Water
18	1888173408	Fairy Tale (Common Reader Editions)	Alice Thomas Ellis
19	0140282653	Jacob's Ladder: A	Donald McCaig

		Story of Virginia During the War	
20	0316196452	Talking to the Dead : A Novel Tag: Winner of the Orange Prize	Helen Dunmor
21	0449001016	Object Lessons (Ballantine Reader's Circle)	Anna Quindlen
22	067179387X	The WEIGHT OF WINTER	Cathie Pelletier
23	0743203232	The Nature of Water and Air	Regina McBride
24	0803727356	The River Between Us	Richard Peck
25	0805073337	What Was She Thinking?: Notes on a Scandal: A Novel	Zoe Heller
26	1888173564	Parnassus on Wheels (Common Reader Editions)	Christopher Morley
27	0449911217	In the Beauty of the Lilies	John Updike
28	0345439481	The High Flyer: A Novel (Ballantine Reader's Circle)	Susan Howatch
29	0684849720	The Quilter's Apprentice : A Novel	Jennifer Chiaverini
30	0140264795	Rosie: A Novel	Anne Lamott
31	0385503857	Oryx and Crake	Margaret Atwood
32	0743418735	Perfect Match: A Novel	Jodi Picoult
33	0060176784	Saying Grace: A	Beth Gutcheon

		Novel	
34	0060283130	The Reptile Room (A Series of Unfortunate Events, Book 2)	Lemony Snicket
35	0060925493	Feather Crowns	Bobbie Ann Maso
36	0066214750	Twelve Times Blessed (Mitchard, Jacqueline)	Jacquelyn Mitchard
37	0156001942	Winter's Tale	Mark Helprin



16. 2nd test case: Diagram with initial recommendations

User clicks all the recommended items and after an hour, the adjusted recommendations are returned to him. We get the following results:

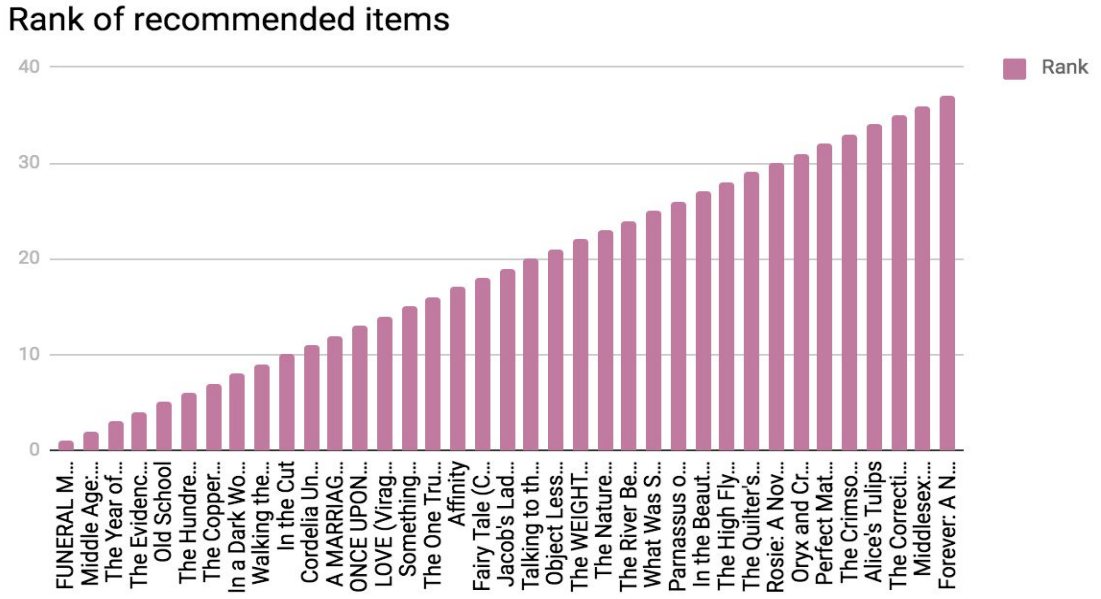
6. 2nd test case: table with adjusted recommendations

Rank of recommended items	**ISBN**	**Title**	**Author**

1	0020236107	FUNERAL MAKERS	Cathie Pelletier
2	0060934905	Middle Age: A Romance	Joyce Carol Oates
3	0312280696	The Year of Jubilo : A Novel of the Civil War	Howard Bahr
4	0316095575	The Evidence Against Her: A Novel	Robb Forman Dew
5	0375401466	Old School	TOBIAS WOLF
6	0375701524	The Hundred Secret Senses	Amy Tan
7	0385307756	The Copper Beech	Maeve Binchy
8	038572117X	In a Dark Wood: A Novel	Amanda Craig
9	0425169626	Walking the Dead	Scott Spence
10	0452281296	In the Cut	Susanna Moore
11	0670880973	Cordelia Underwood: Or the Marvelous Beginnings of the Moosepath League	Van Reid
12	0671516949	A MARRIAGE MADE AT WOODSTOCK	Cathie Pelletier
13	0671724479	ONCE UPON A TIME ON THE BANKS : ONCE UPON A TIME ON THE BANKS	Cathie Pelletier
14	0671883925	LOVE (Virago Modern Classics)	Elizabeth Von arnim

15	089587167X	Something Blue	Jean Christopher Spaugh
16	1402201435	The One True Ocean	Sarah Beth Martin
17	1573221562	Affinity	Sarah Water
18	1888173408	Fairy Tale (Common Reader Editions)	Alice Thomas Ellis
19	0140282653	Jacob's Ladder: A Story of Virginia During the War	Donald McCaig
20	0316196452	Talking to the Dead : A Novel Tag: Winner of the Orange Prize	Helen Dunmor
21	0449001016	Object Lessons (Ballantine Reader's Circle)	Anna Quindlen
22	067179387X	The WEIGHT OF WINTER	Cathie Pelletier
23	0743203232	The Nature of Water and Air	Regina McBride
24	0803727356	The River Between Us	Richard Peck
25	0805073337	What Was She Thinking?: Notes on a Scandal: A Novel	Zoe Heller
26	1888173564	Parnassus on Wheels (Common Reader Editions)	Christopher Morley
27	0449911217	In the Beauty of the Lilies	John Updike
28	0345439481	The High Flyer: A Novel (Ballantine Reader's Circle)	Susan Howatch

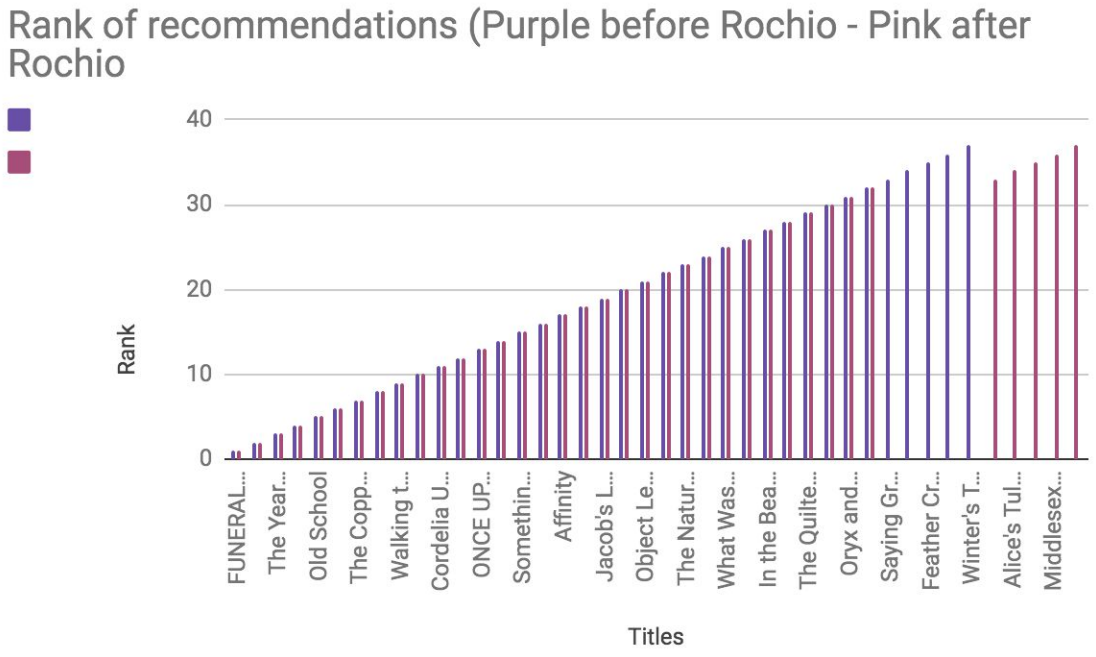
29	0684849720	The Quilter's Apprentice : A Novel	Jennifer Chiaverini
30	0140264795	Rosie: A Novel	Anne Lamott
31	0385503857	Oryx and Crake	Margaret Atwood
32	0743418735	Perfect Match: A Novel	Jodi Picoult
33	0156028778	The Crimson Petal and the White	Michel Faber
34	0312283784	Alice's Tulips	Sandra Dallas
35	0312421273	The Corrections: A Novel	Jonathan Franzen
36	0312422156	Middlesex: A Novel	Jeffrey Eugenides
37	0316341118	Forever: A Novel	Pete Hamill



17. 2nd test case: Diagram with the adjusted recommendations

The recommended items are quite similar with the original, but some new recommended items are appeared on the end of the list. These books are similar with some books clicked by user. As it is obvious from the following diagram, only the last recommended books are different like “Perfect Match:

A Novel", "The Crimson Petal and the White" , "Alice's Tulips", "The Corrections: A Novel", "Middlesex: A Nove", "Forever: A Novel"



18. 2nd test case: Comparative diagram of recommendations before/after rochio.

3rd scenario:Existed user with user_id=11677(having data for him into database) does not click any recommended item

When user signs in books recommendation application, he gets a list with the initial recommendations, as they have been calculated with collaborative filtering algorithm.

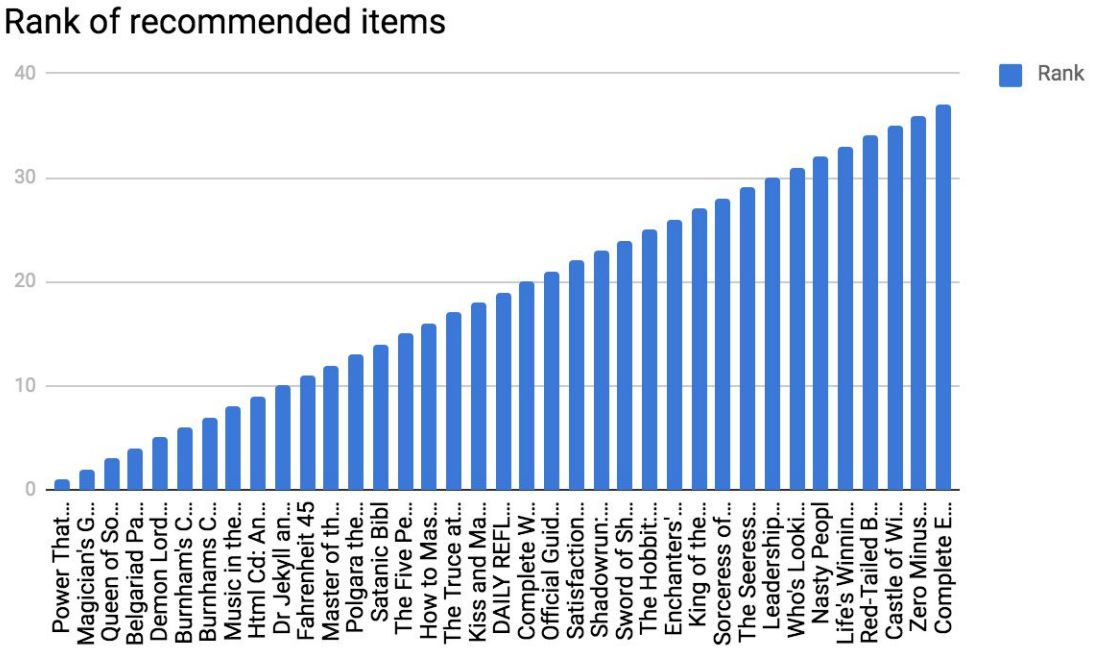
7. 3rd test case: table with initial recommendations

Rank of recommended items	**ISBN**	**Title**	**Author**
1	0345257189	Power That Preserves Covenant 3	stephen R Donaldson
2	0345300777	Magician's Gambit (Eddings, David. , the Belgariad, Bk. 3.)	David Eddings

3	0345300793	Queen of Sorcery (Eddings, David. , the Belgariad, Bk. 2.)	David Eddings
4	0345309979	Belgariad Part One (Eddings, David. , the Belgariad, Bk. 1.)	David Eddings
5	0345363310	Demon Lord of Karanda (Malloreon (Paperback Random House))	DAVID EDDINGS
6	0486240649	Burnham's Celestial Handbook, Volume 2, Rev. Edition	Robert Burnha
7	0486240657	Burnhams Celestial Handbook Volume 3	Robert Burnham
8	0028713303	Music in the Romantic Period: An Anthology with Commentary	F. E. Kirby
9	0132323311	Html Cd: An Internet Publishing Toolkit for Windows/Book and Cd-Rom	Vivian Neou
10	0140620516	Dr Jekyll and Mr Hyde (Penguin Popular Classics)	Robert Louis Stevenson
11	0345274318	Fahrenheit 451	Bradbury
12	0345276353	Master of the Five Magics 1	Lyndon Hardy
13	0345373243	Belgarath the Sorcerer	David Eddings

14	0345416627	Polgara the Sorceress	David Eddings
15	0380015390	Satanic Bible	Anton Szandor Lavey
16	0446384038	How to Master the Art of Selling	Tom Hopkin
17	0553095412	The Truce at Bakura (Star Wars)	Kathy Tyer
18	0609810022	Kiss and Make-Up	Gene Simmons
19	0671887173	DAILY REFLECTIONS FOR HIGHLY EFFECTIVE PEOPLE : Living The 7 Habits Of Highly Successful People Every Day	Stephen R. Covey
20	0831716991	Complete Works of William Shakespeare	William Shakespeare
21	0938636057	Official Guide to Success (Official Guide to Success)	Tom Hopkins
22	1558532862	Satisfaction Guaranteed	Byrd Baggett
23	1932564462	Shadowrun: 25000	Fanpro
24	034527444X	Sword of Shannar	Brooks
25	0345296044	The Hobbit: Or There and Back Again	J. R. R. Tolkien
26	0345300785	Enchanters' End Game (The Belgariad, Book 5)	David Eddings
27	0345358805	King of the Murgos	David Edding

		(Malloreon (Paperback Random House))	
28	0345369351	Sorceress of Darshiva (Malloreon (Paperback Random House))	David Eddings
29	0345377591	The Seeress of Kell (The Malloreon, Book 5)	David Eddings
30	0446391069	Leadership Secrets of Attila the Hun	Wess Roberts
31	0767913795	Who's Looking Out for You?	BILL O'REILLY
32	0880296879	Nasty Peopl	Jay Carter
33	1562450662	Life's Winning Tips	Dennis Connor
34	1882770099	Red-Tailed Boas (General Care and Maintenance of Series)	Philippe De Vosjoli
35	0345300807	Castle of Wizardry (The Belgariad, Book 4)	David Eddings
36	0515123366	Zero Minus Ten	Raymond Benson
37	0785815198	Complete Encyclopedia Of Pistols And Revolvers	A. E. Hartnik



19. 3rd test case: Diagram with initial recommendations

User does not click any recommended item. Every hour, we adjust the recommended items and a negative weight is added on their predictive value.

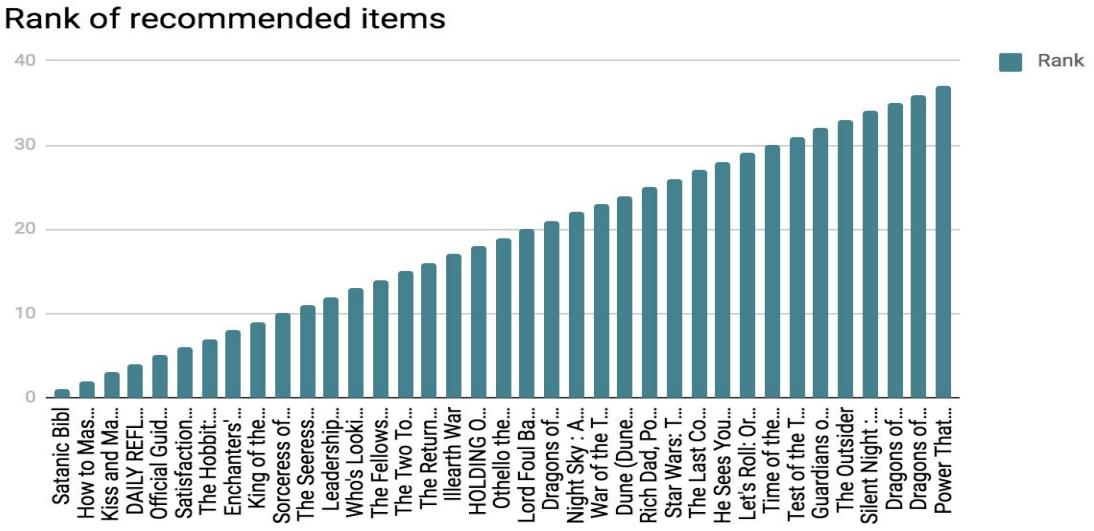
8. 3rd test case: table with adjusted recommendations

Rank of recommended items	**ISBN**	**Title**	**Author**
1	0380015390	Satanic Bible	Anton Szandor Lavey
2	0446384038	How to Master the Art of Selling	Tom Hopkin
3	0609810022	Kiss and Make-Up	Gene Simmons
4	0671887173	DAILY REFLECTIONS FOR HIGHLY EFFECTIVE PEOPLE : Living The 7 Habits Of Highly Successful	Stephen R. Covey

		People Every Day	
5	0938636057	Official Guide to Success (Official Guide to Success)	Tom Hopkins
6	1558532862	Satisfaction Guaranteed	Byrd Baggett
7	0345296044	The Hobbit: Or There and Back Again	J. R. R. Tolkien
8	0345300785	Enchanters' End Game (The Belgariad, Book 5)	David Eddings
9	0345358805	King of the Murgos (Malloreon (Paperback Random House))	David Eddings
10	0345369351	Sorceress of Darshiva (Malloreon (Paperback Random House))	David Edding
11	0345377591	The Seeress of Kell (The Malloreon, Book 5)	David Edding
12	0446391069	Leadership Secrets of Attila the Hun	Wess Roberts
13	0767913795	Who's Looking Out for You?	BILL O'REILLY
14	0345296052	The Fellowship of the Ring (Lord of the Rings (Paperback))	J. R. R. Tolkien
15	0345296060	The Two Towers (Lord of the Rings (Paperback))	J. R. R. Tolkien
16	0345296087	The Return of the King (Lord of the	J. R. R. Tolkien

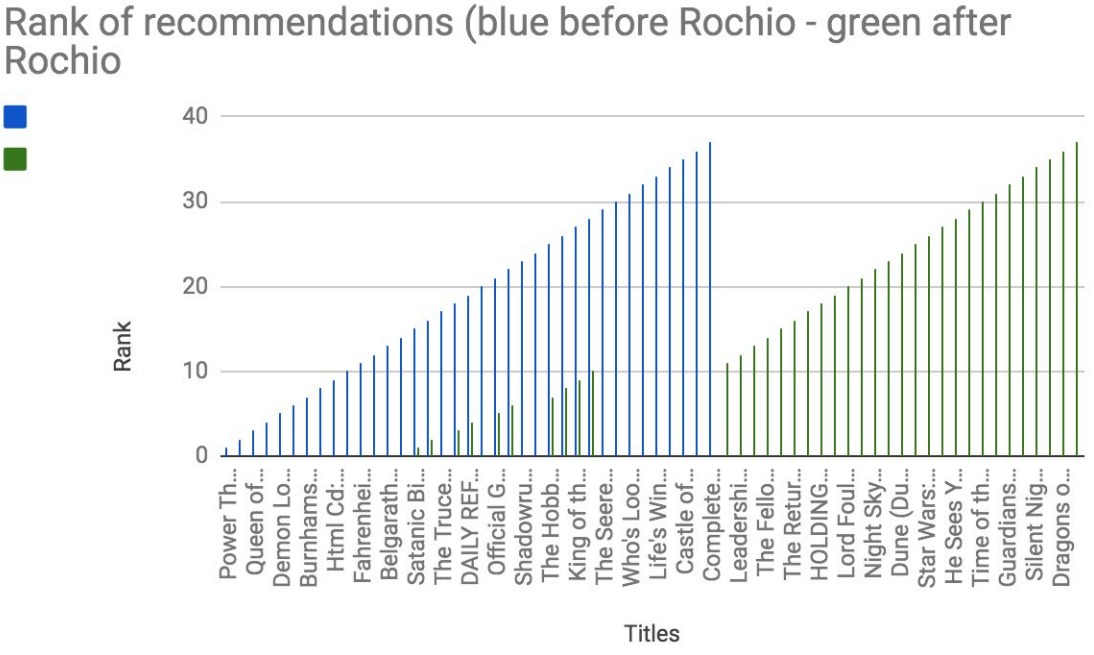
		Rings (Paperback))	
17	0345296567	Illearth War	Stephen R Donaldson
18	0684846713	HOLDING OUT : A NOVEL	Anne O. Faul
19	0140714103	Othello the Moor of Venice (The Pelican Shakespeare)	William Shakespeare
20	0345296575	Lord Foul Ban	Stephen R Donaldso
21	0880381744	Dragons of Winter Night	Margaret Weis
22	0307136671	Night Sky : A Field Guide to the Heavens	Mark Chartrand
23	0394745787	War of the Twins (Dragonlance Legends, Vol. 2)	Margaret Weis
24	042507160X	Dune (Dune Chronicles (Berkley Paperback))	Frank Herbert
25	0446677450	Rich Dad, Poor Dad: What the Rich Teach Their Kids About Money--That the Poor and Middle Class Do Not!	Robert T. Kiyosak
26	0553089285	Star Wars: The Courtship of Princess Leia (Star wars)	Dave Wolverton
27	0553091867	The Last Command (Star Wars, Vol 3)	Timothy Zahn

28	0743456866	He Sees You When You're Sleeping : A Novel	Carol Higgins Clark
29	0842373195	Let's Roll: Ordinary People, Extraordinary Courage	Lisa Beamer
30	0880382651	Time of the Twins Legends 1 (Dragonlance Legends Trilogy, Vol 1)	Margaret Weis
31	0880382678	Test of the Twins (DragonLance Legends, Vol 3)	Margaret Weis
32	0345352661	Guardians of the West (Book 1 of the Malloreon)	David Eddings
33	0440967694	The Outsiders	S. E. Hinton
34	067100042X	Silent Night : A Christmas Suspense Story	Mary Higgins Clark
35	0880381736	Dragons of Autumn Twilight (Dragonlance Chronicles, Vol 1)	Margaret Weis
36	0880381752	Dragons of Spring Dawning (Dragonlance Chronicles, Vol 3)	Margaret Weis
37	0345257189	Power That Preserves Covenant 3	stephen R Donaldson



20. 3rd test case: Diagram with adjusted recommendations

As we can see from the above diagram, after some time, new items are recommended to user while the order of existed recommended books has been also changed. Some books like "Satanic Bible" are still appeared on list, while recommendations list has been enriched with new suggestions like the "Fellowship of the ring"



21. 3rd test case: Comparative Diagram with recommendations before and after rochio.

4rth scenario: New registered user(not having data for him into database) clicks some of the recommended items

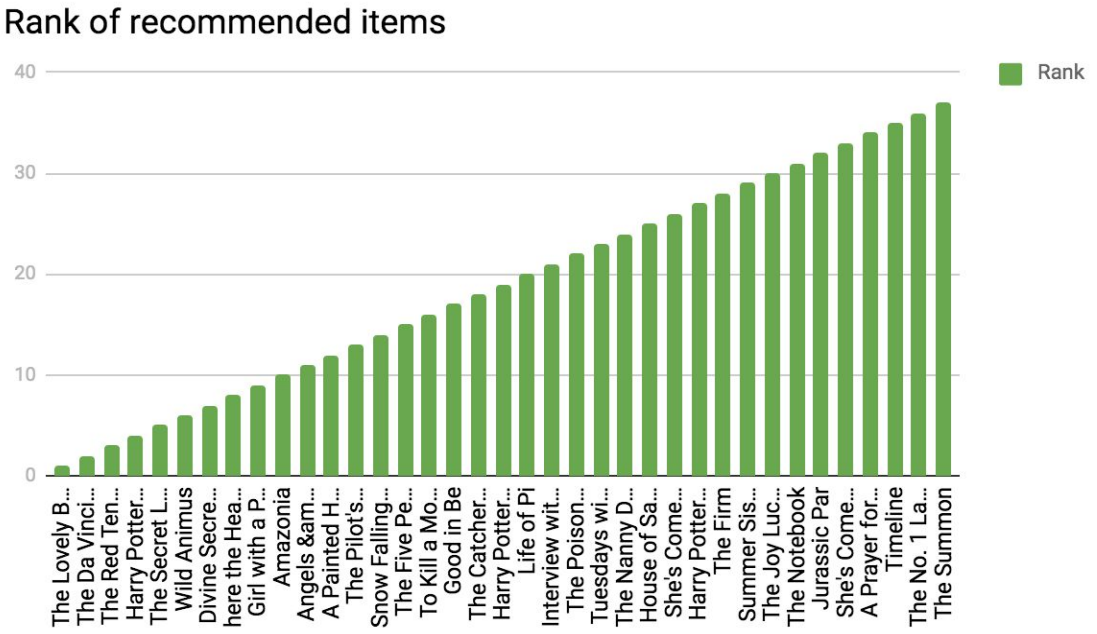
When a new user is registered on application, he gets a list with the top rated books as recommendations.

9. 4rth test case: table with initial recommendations

Rank of recommended items	**ISBN**	**Title**	**Author**
1	0316666343	The Lovely Bones: A Novel	Alice Sebol
2	0385504209	The Da Vinci Code	Dan Brown
3	0312195516	The Red Tent (Bestselling Backlist)	Anita Diaman
4	059035342X	Harry Potter and the Sorcerer's Stone (Harry Potter (Paperback))	J. K. Rowling
5	0142001740	The Secret Life of Bee	Sue Monk Kidd
6	0971880107	Wild Animus	Rich Shapero
7	0060928336	Divine Secrets of the Ya-Ya Sisterhood: A Novel	Rebecca Wells
8	0446672211	Where the Heart Is (Oprah's Book Club (Paperback))	Billie Letts
9	0452282152	Girl with a Pearl Earrin	Tracy Chevalie
10	0671027360	Angels & Demon	Dan Brown
11	044023722X	A Painted House	John Grisham

12	0316601950	The Pilot's Wife : A Novel	Anita Shreve
13	067976402X	Snow Falling on Cedars	David Guterson
14	0786868716	The Five People You Meet in Heaven	Mitch Albo
15	0446310786	To Kill a Mockingbird	Harper Le
16	0743418174	Good in Bed	Jennifer Weine
17	0316769487	The Catcher in the Rye	J.D. Salinger
18	043935806X	Harry Potter and the Order of the Phoenix (Book 5)	J. K. Rowlin
19	0156027321	Life of Pi	Yann Marte
20	0345337662	Interview with the Vampire	Anne Rice
21	0060930535	The Poisonwood Bible: A Nove	Barbara Kingsolve
22	0385484518	Tuesdays with Morrie: An Old Man, a Young Man, and Life's Greatest Lesson	MITCH ALBOM
23	0312278586	The Nanny Diaries: A Novel	Emma McLaughlin
24	0375727345	House of Sand and Fo	Andre Dubus II
25	0671021001	She's Come Undone (Oprah's Book Club)	Wally Lam
26	0439064872	Harry Potter and the Chamber of Secrets (Book 2)	J. K. Rowlin

27	044021145X	The Firm	John Grisham
28	0440226430	Summer Sisters	Judy Blum
29	0804106304	The Joy Luck Club	Amy Tan
30	0446605239	The Notebook	Nicholas Spark
31	0345370775	Jurassic Park	Michael Crichton
32	0671003755	She's Come Undone (Oprah's Book Club (Paperback))	Wally Lamb
33	0345361792	A Prayer for Owen Meany	John Irving
34	0440211727	A Time to Kill	JOHN GRISHA
35	0345417623	Timeline	MICHAEL CRICHTON
36	1400034779	The No. 1 Ladies' Detective Agency (Today Show Book Club #8)	Alexander McCall Smith
37	0440241073	The Summon	John Grisha



22. 4rth test case: Diagram with initial recommendations

User clicks the following recommended books: "0385504209", "0439064872" , "0142001740", "0140503528", "043935806X", "0439064872", "0345337662"

After a while he gets the following recommendations:

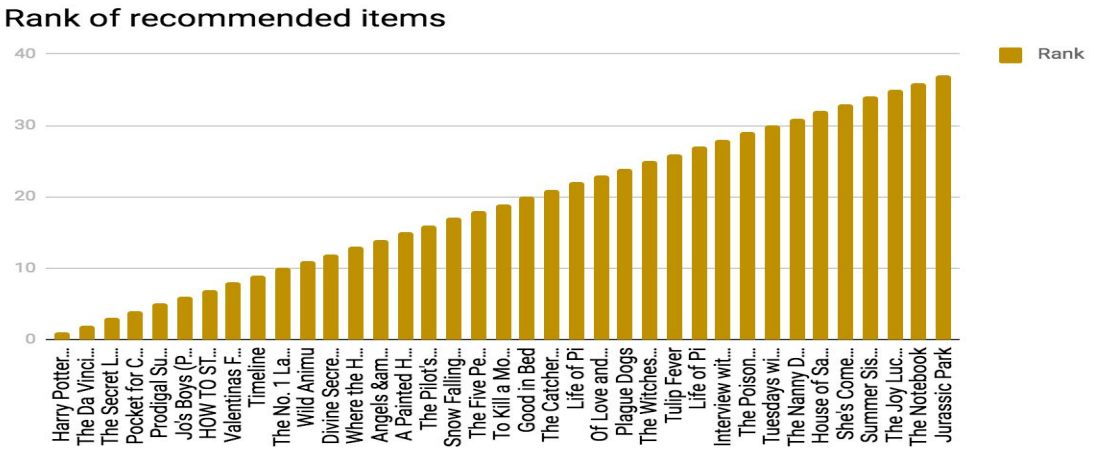
10.4rth case: table with adjusted recommendations

Rank of recommended items	**ISBN**	**Title**	**Author**
1	0439064872	Harry Potter and the Chamber of Secrets (Book 2)	J. K. Rowling
2	0385504209	The Da Vinci Code	Dan Brown
3	0142001740	The Secret Life of Bees	Sue Monk Kidd
4	0140503528	Pocket for Corduroy (Picture	Don Freeman

		Puffins)	
5	0060959037	Prodigal Summer: A Novel	Barbara Kingsolver
6	0140367144	Jo's Boys (Puffin Classics)	Louisa May Alcott
7	0020280505	HOW TO STAY ALIVE IN THE WOODS	Bradford Angier
8	0304353884	Valentinas Four Seasons Cookbook	Valentina Harris
9	0345417623	Timeline	MICHAEL CRICHTON
10	1400034779	The No. 1 Ladies' Detective Agency (Today Show Book Club #8)	Alexander McCall Smith
11	0971880107	Wild Animu	Rich Shapero
12	0060928336	Divine Secrets of the Ya-Ya Sisterhood: A Novel	Rebecca Wells
13	0446672211	Where the Heart Is (Oprah's Book Club (Paperback))	Billie Letts
14	0671027360	Angels & Demons	Dan Brown
15	044023722X	A Painted House	John Grisham
16	0316601950	The Pilot's Wife : A Novel	Anita Shreve
17	067976402X	Snow Falling on Cedars	David Guterson

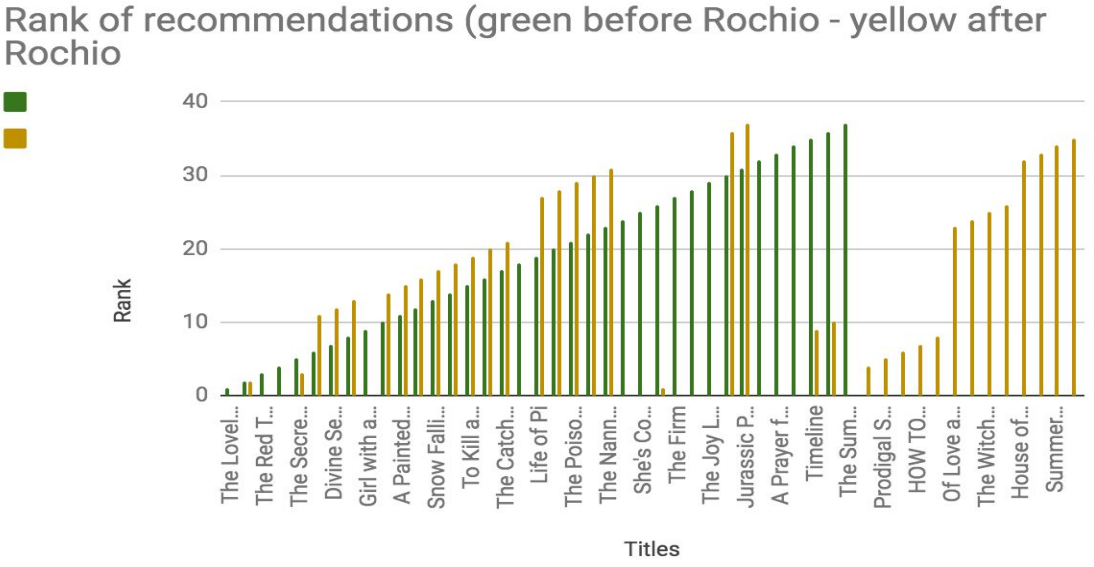
18	0786868716	The Five People You Meet in Heaven	Mitch Albom
19	0446310786	To Kill a Mockingbird	Harper Lee
20	0743418174	Good in Bed	Jennifer Weine
21	0316769487	The Catcher in the Rye	J.D. Salinger
22	0156027321	Life of Pi	Yann Martel
23	0140256369	Of Love and Other Demons (Penguin Great Books of the 20th Century)	Gabriel Garcia Marque
24	0449211827	Plague Dogs	RICHARD ADAM
25	0449912108	The Witches of Eastwick	John Updike
26	0385334923	Tulip Fever	DEBORAH MOGGACH
27	0156027321	Life of Pi	Yann Martel
28	0345337662	Interview with the Vampire	Anne Rice
29	0060930535	The Poisonwood Bible: A Novel	Barbara Kingsolver
30	0385484518	Tuesdays with Morrie: An Old Man, a Young Man, and Life's Greatest Lesson	MITCH ALBOM
31	0312278586	The Nanny Diaries: A Novel	Emma McLaughli
32	0375727345	House of Sand and Fog	Andre Dubus II
33	0671021001	She's Come Undone (Oprah's Book Club)	Wally Lam

34	0440226430	Summer Sisters	Judy Blume
35	0804106304	The Joy Luck Club	Amy Ta
36	0446605239	The Notebook	Nicholas Sparks
37	0345370775	Jurassic Park	Michael Crichton



23. 4rth case: Diagram with adjusted recommendations

As we can see, the order of results has been changed, while new recommended books are appeared on list which are considered similar with the books user has clicked.



24. 4rth case: Comparative diagram with recommendations before/after rochio.

5. Conclusion

Taking everything into consideration, the books recommendation system built for this thesis's purpose is quite effective. From performance aspect, the implementation is not so demanded, as the original recommendations and the similarity between items can be calculated offline and then we just need to adjust them based on user's feedback, implementing the Rochio algorithm. Some common problems like cold start and users not clicking any recommended item can be solved easily. In order to handle the users not having data about them (cold start problem) we can just return to user the top rated books and then adjust the results based on user's feedback. If user does not click any recommended item, a negative weight is added on item's prediction value, and new recommended items are appeared on list. The algorithm to adjust user's recommended items does not require complicated calculations and user can get the updated recommended items immediately. As future improvements, different approaches could be tried on the first part where the initial recommendations are calculated for the user. Maybe a deep learning model could result in more precise initial recommendations. The deep learning model could combine collaborative-filtering and content-based information and predict users' recommended items given their previous activities (search queries and videos watched) and static information (gender, location, etc.). Also the recommendations system could become more scalable if an unified analytics engine for large-scale data processing like Apache spark or Handoop was used to calculate the initial recommendations. As far as the second part of recommendations' adjustment based on users' feedback is concerned, more user's actions could be taken into account to adjust the recommendations like user's latest ratings, reviews etc. The Rochio algorithm could be implemented to train the initial deep learning model based on user's feedback.

6. Bibliography

6.1 Books

Charu C. Aggarval*: "Recommender Systems, The textbook"*, Yorktown Heights, NY, USA, Springer

Suresh Kumar Gorakala, "Building Recommendation Engine" , Birmingham, Uk, Paclt Publishing Ltd 2016

Francois Chollet, "Deep learning with Pyhton", Shelter Island, NY, Manning Publications Co. 2018

Christopher D.Manning, Prabhakar Raghavan, Hinrich Schutze, "An introduction to Information Retrieval", Cambridge England, Cambridge University Press 2009

Dietmar Jannach, Markus Zanker, Alexander Felfernig, Gerhard Friedich "Recommender Systems An Introduction", Cambridge University Press 2011

Francesco Ricci and Lior Rokach and Bracha Shapira, "Introduction to Recommender Systems Handbook", Springer Science+Business Media, LLC 2011, 2011

6.2 Articles

K. Goldberg, T. Roeder, D. Gupta, and C. Perkins, "Eigentaste: a constant time collaborative filtering algorithm," Information Retrieval, vol. 4, no. 2, pp. 133–151, 2001

Linden, B. Smith, and J. York, "Amazon.com recommendations: item-to-item collaborative filtering," IEEE Internet Computing, vol. 7, no. 1, pp. 76–80, 2003.

Mark van Uden, "Rochio: Relevance feedback in learning Classification Algorithms", Department of Computing Science, University of Nijmegen

Ladislav Peska, "Using the Context of User Feedback in Recommender Systems", Faculty of Mathematics and Physics, Charles University in Prague,, Prague, Czech Republic

6.3 Web sites

https://towardsdatascience.com/intro-to-recommender-system-collaborative-filtering-64a238194a26

https://towardsdatascience.com/recommender-systems-in-practice-cef9033bb23a

https://www.geeksforgeeks.org/python-implementation-of-movie-recommender-system/

https://www.djangoproject.com

https://angularjs.org

https://pandas.pydata.org

https://redis.io

http://www.celeryproject.org/