Interactive Recommendation Systems

Diploma thesis of Fotini Gkiouleka

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 filtering algorithm. Based on users' feedback the recommendations are updated on real time and more relevant recommended items are displayed to users.
- In order to adjust recommendations, an implementation of Rochio algorithm is suggested and some common problems in the field of recommendation systems are tried to be solved like cold start problem, user does not give feedback for recommended items etc.
- For the purposes of this thesis a demo books recommendation application has been developed and some testing cases have been carried out in order to see how the recommendation system reacts on different user's actions. The code of demo application can be found on github repository: https://github.com/fotein1/book_recommender

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.

Goals of recommendation system

- **Relevance**: A recommendation system should recommend items relevant to the user's preferences
- **Novelty**: A recommender system are more helpful when recommended items have not been seen by user in the past.
- **Serendipity**: The recommended items are somewhat unexpected to user.
- **Increasing recommendation diversity**: When the recommender system suggests items of different types, there is a greater chance that the user might like at least one of these items.

Types of recommendation systems

- Collaborative filtering models: These models are based on user-tems interactions such as rating or buying behavior. 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.
  - User based collaborative filtering: Consider the preferences of similar users
  - Item based collaborative filtering: Consider the similarities of items.
- Content based recommender systems: These models are based on content information of items
- Knowledge based recommender systems: Users interactively specify their interests
- Cortext-aware recommendation system: Take into account some other parameters like such as day, season, mood, region etc.
- Hybrid recommender systems: Combine different recommendation models

Collaborative filtering nearest neighborhood models

The standard method of Collaborative Filtering is known as Nearest Neighborhood algorithm

- User based CF: Find the most similar users on the target user(nearest neighbors) and weight their ratings of an item as an prediction of the rating of this item for the target user
- Item based CF: 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
- For the demo application 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.

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

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

Relevance feedback Rochio algorithm

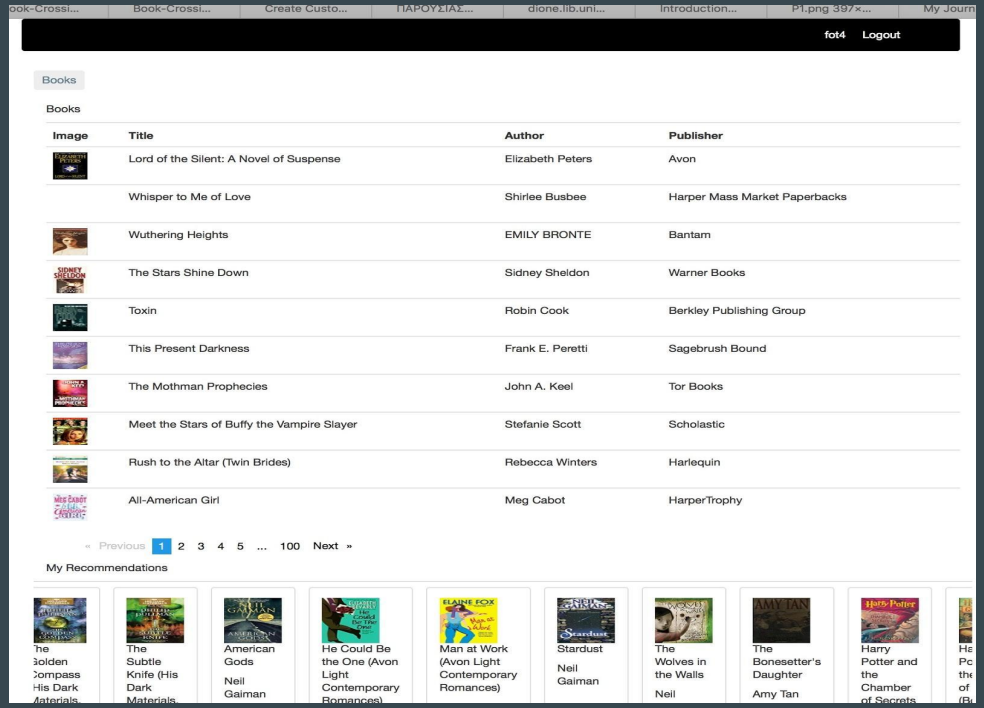
- The idea of *relevance feedback* (RF) is to involve the user in the retrieval process so as to improve the final result set. The Rocchio Algorithm is the classic algorithm for implementing relevance feedback. It models a way of incorporating relevance feedback information into the vector space.
- The algorithm proposes using the modified query qm

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

where *q*0 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

Books recommendations application

- login/register user
- get books lists
- get recommendations
- rate book
- view book



Technologies

- Django: Python based framework, used to build the required web services for application (backend part of application)
- Angular: Javascript based framework, used to build the user interface (frontend part of application)
- Redis: in-memory data structure store, used to cache data
- Celery: An asynchronous task queue/job queue based on distributed message passing, used to execute heavy tasks on background.
- Python libraries: **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. **Scikit-learn** package was used to implement some machine learning algorithms.

Datasets

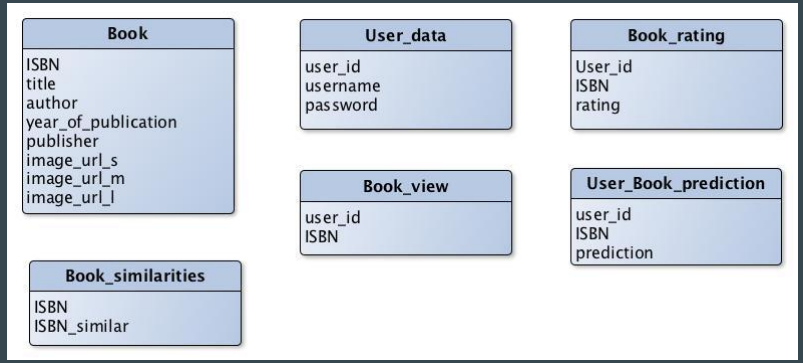
For the purposes of this thesis the Book Crossing dataset has been used. This dataset has been compiled by Cai-Nicolas Ziegler in 2004, and it comprises of three tables for users, books and ratings.

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

Database structure

- We have opted to work with sqlite database. We have created the relevant django models to add the following tables in database.

  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



ETL (Extract Transform Load) process

- After installing django framework, setting up django server and creating the above db tables we filled the local database with the online datasets. This process is known as ETL.
- We extracted data from csv files using python pandas library, we calculated the users' predictions and books' similarities and we loaded the data the into database.
- To implement the ETL process we created the created the following django scripts, which can be executed by command line:
  load_users.py
  load_books.py
  load_ratings.py
  load_books_smilarities.py
  load_user_predictions.py

Calculate similarities of books

- 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, implementing the adjusted cosine similarity equation.

```
                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
```

Predict users' ratings for books not rated yet

- In order to predict the user's rating for a book, we calculate the weighted sum of user's ratings for the similar books.

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

```
"""
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))
```

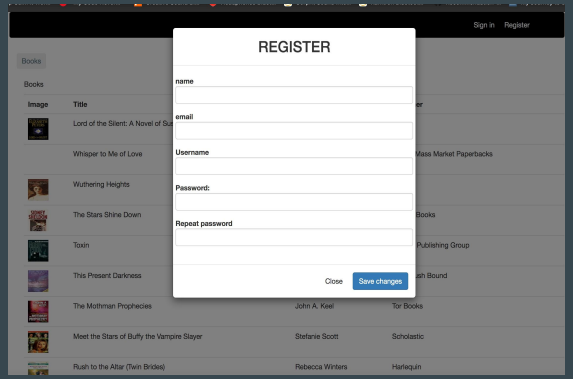
Backend part of books recommendation application

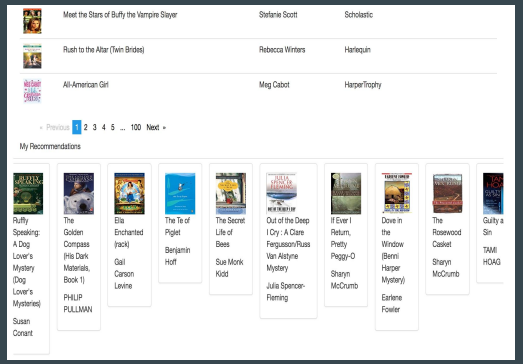
We created the following RestFul web services:

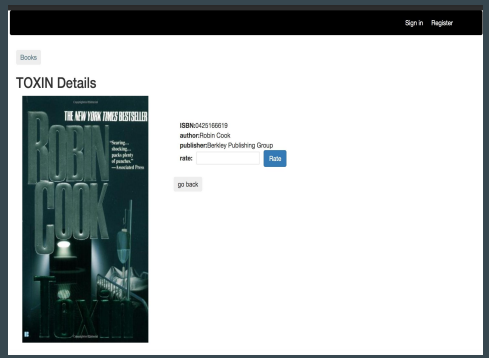
- register: POST /api/accounts
- login: POST /api/sessions
- logout: DELETE /api/sessions
- Get books list: GET /api/books?page=1
- Get books details: GET /api/books/{ISBN}
- View book: POST /api/books-views
- Rate book: POST /api/books-rates
- Get recommendations: GET /api/books-recommendations/users/{user_id}

Frontend part of books recommendation application

- The frontend part of books recommendation application was built with angular framework. For every view a separate angular module was created







Design of recommendations system - Get recommendations

- A web service has been created to return the recommendations to user. 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
- If user does not have any prediction stored into database, a list with top rated books is returned to him.

```
// 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_cac
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)
```

Design of recommendations system - User feedback

- 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.
- When user views a recommended book, it is considered as positive feedback, otherwise a negative feedback
- We find the similar books of recommended books viewed by user and we increase their prediction by the positive weight of Rochio algorithm
- We find the similar books of recommended books not viewed by user and we reduce their prediction by the positive weight of Rochio algorithm
- We update the users' predicted rating into database
- If user does not have any prediction stored into database yet, we insert the adjusted predicted values into db.
- The adjusted recommendations are set in cache

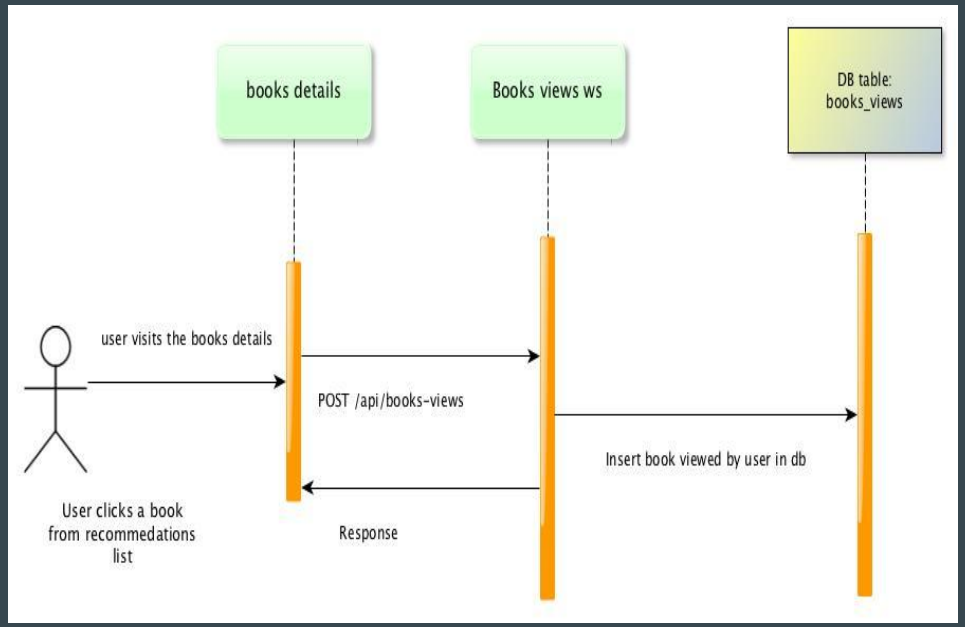
$$positive\_prediction = intitial\_prediction + \frac{0.75*(prediction_1+....+prediction_n).}{n}$$

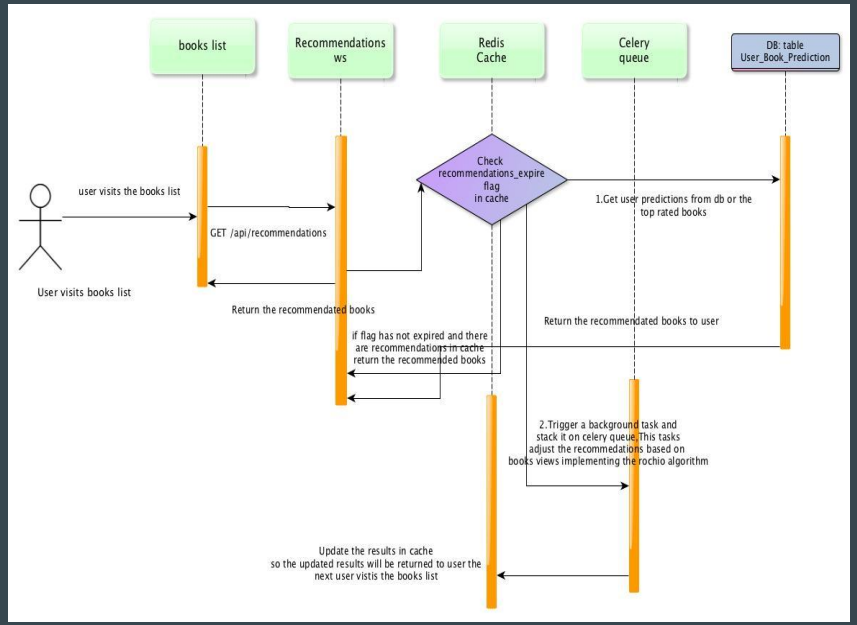
$$negative\_prediction = intitial\_prediction - \frac{0.15*(prediction_1+....+prediction_n).}{n}$$

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
"""
```

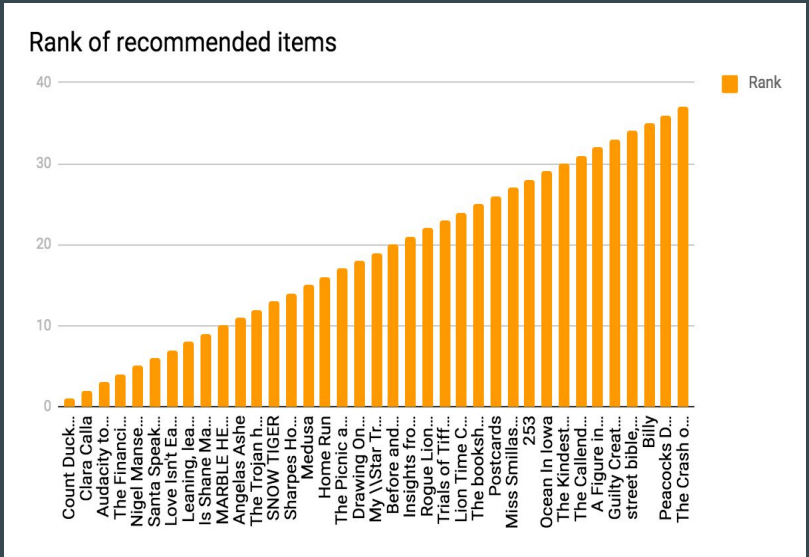
Design of recommendation system - Workflow



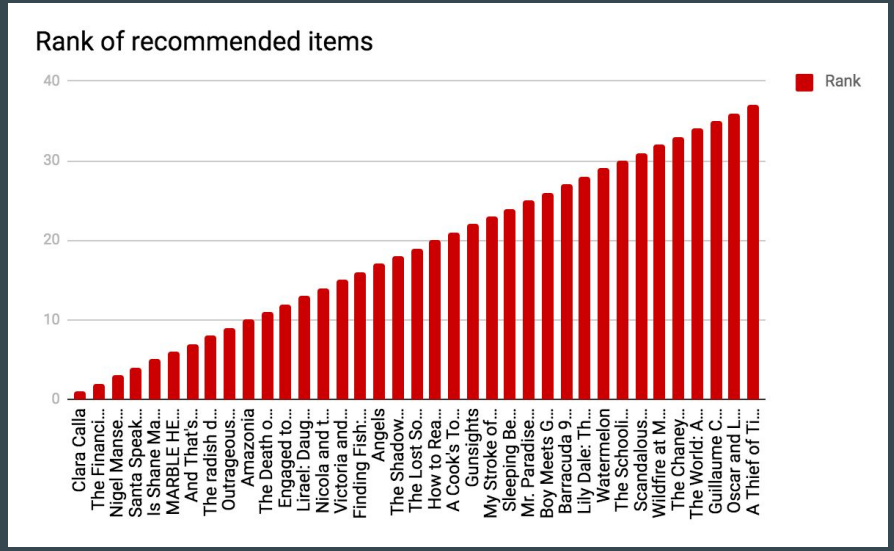


1st test case: Existed user clicks some recommended items

- Initial recommendations

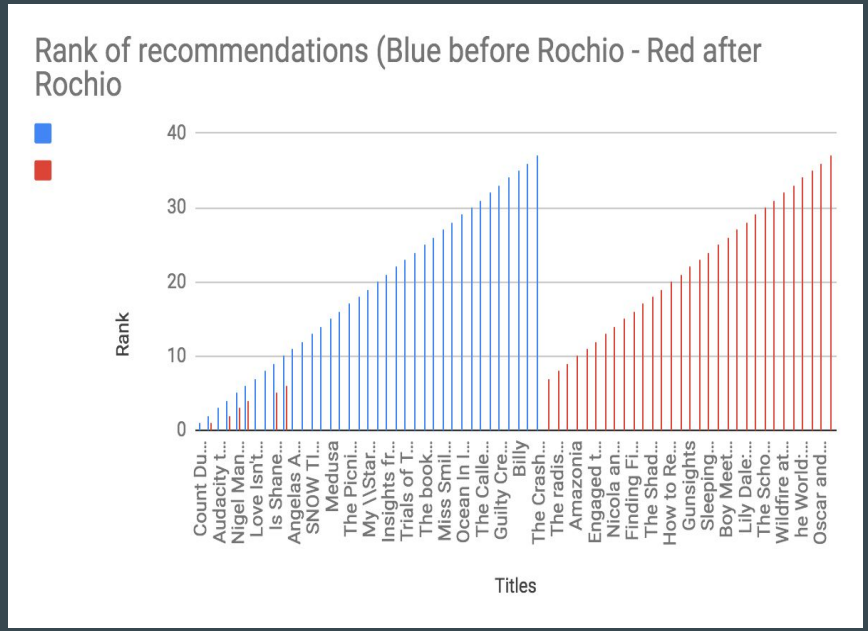


- Adjusted recommendations



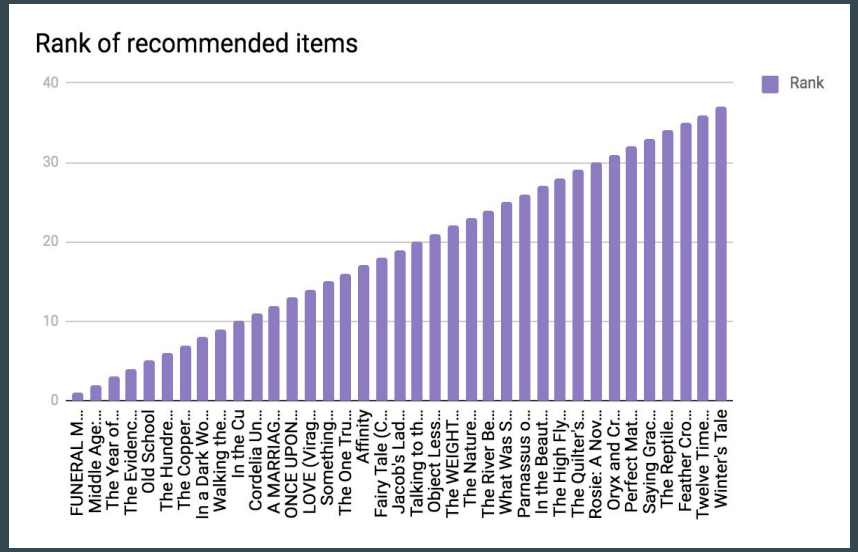
1st test case: Existed user clicks some recommended items

- 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..
- The previous recommended books which have not been clicked by the user, have been replaced by new recommendations as a negative wight has been added on their initial prediction value. Some new books are appeared on recommendations list like Outrageous fortune, Amazon

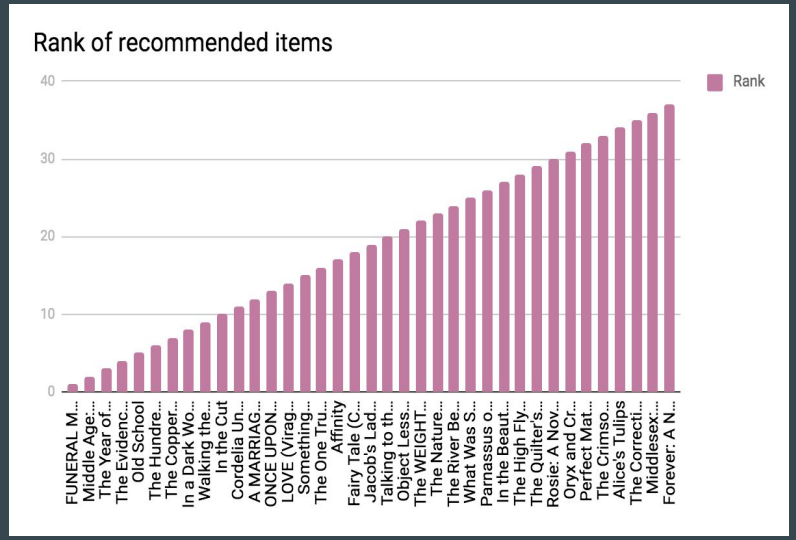


2nd test case: Existed user clicks all the recommended items

- Initial recommendations

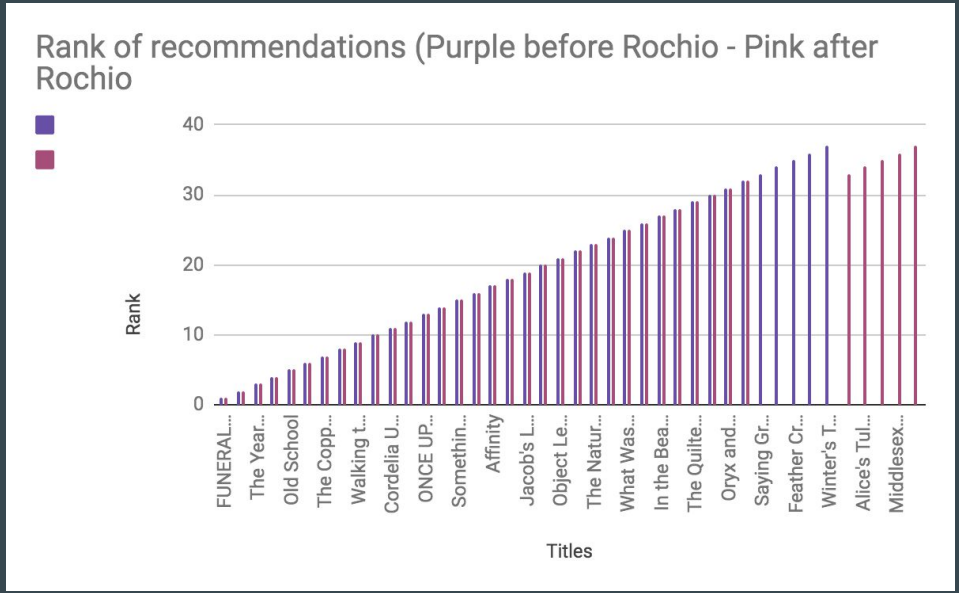


- Adjusted recommendations



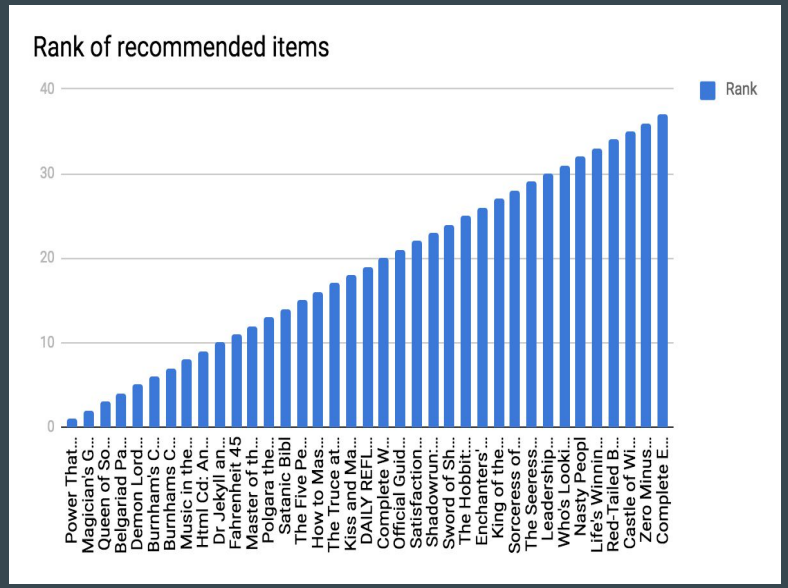
2nd test case: Existed user clicks all the recommended items

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”

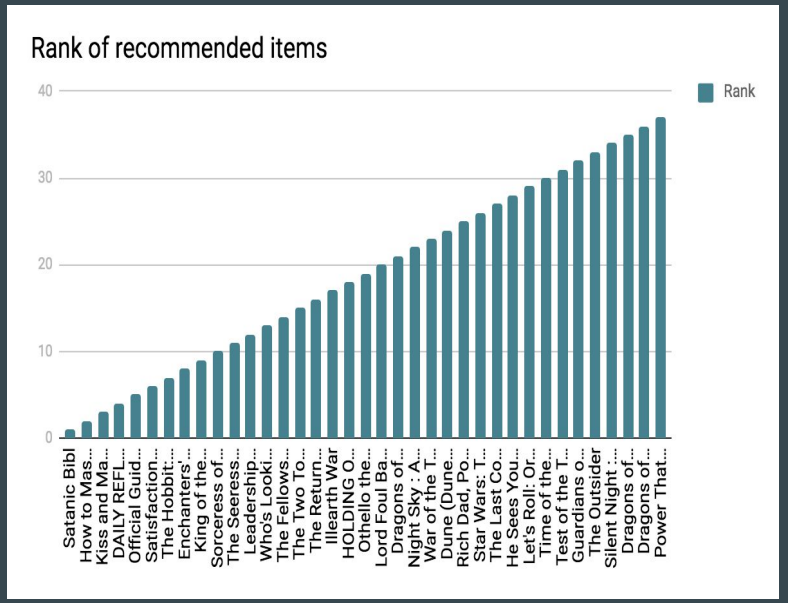


3rd case: Existed user does not click any recommended item

- initial recommendations

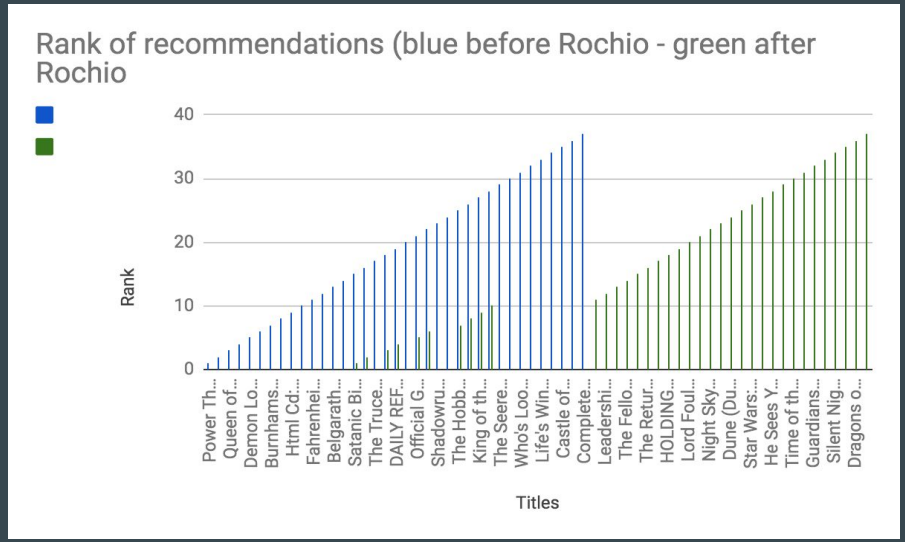


- Adjusted recommendations



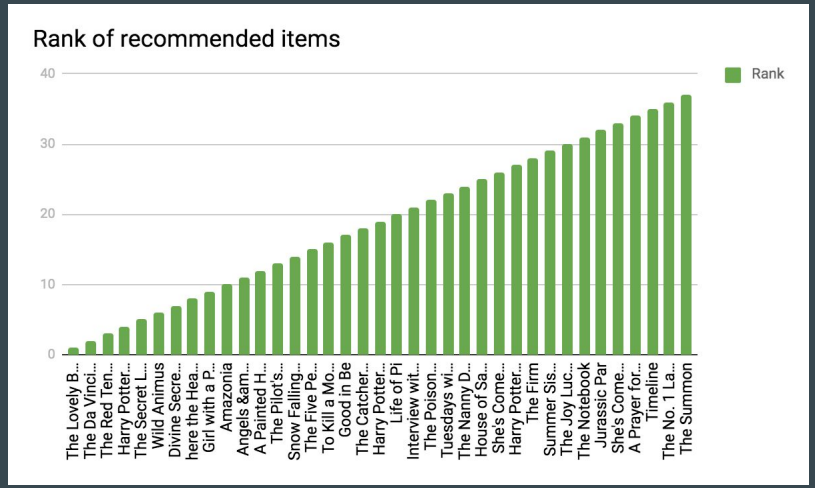
3rd case: Existed user does not click any recommended item

- 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”

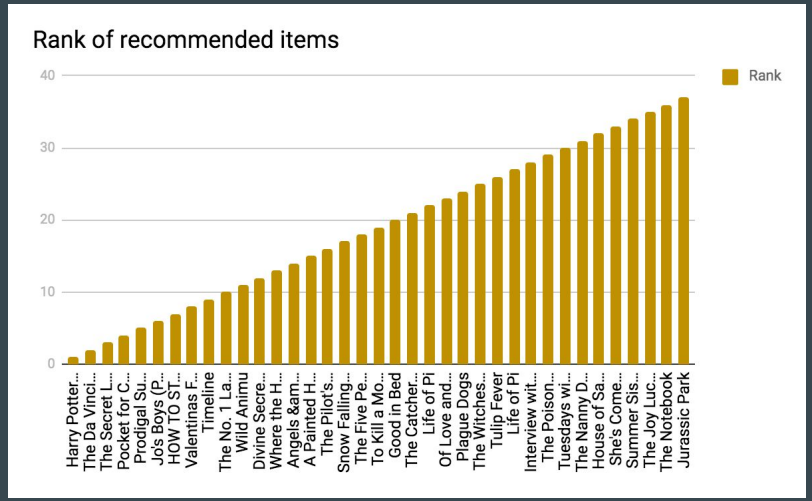


4rth test case: New user clicks some recommended items

- Initial recommendations

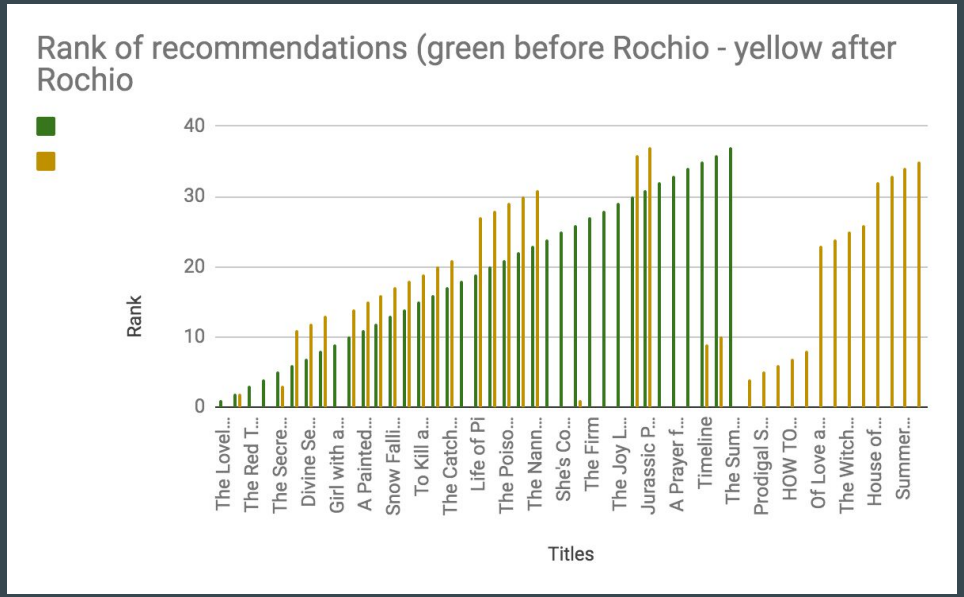


- Adjusted recommendations



4rth test case: New user clicks some recommended items

- New registered user gets a list with top rated recommended books. After clicking some recommended books, the recommendations are adjusted, 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.



Conclusion

- 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
- The algorithm to adjust user's recommended items does not require complicated calculations and user can get the updated recommended items immediately

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 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.

Thank you