Designing a query recommendation system

Munkhdelger Bayanjargal
Pooria Sajadi Parsa*
m.bayanjargal@unitn.it
seyed.sajadiparsa@studenti.unitn.it

1 INTRODUCTION

With the advancements in technology, a huge majority of products and services are being offered and sold on the internet. In addition, these products/services have massively increased both in number and variety which in turn makes it hard for consumers to find the product/service they are looking for.

Recommendation systems study customer behavior, provide customized recommendations, reduce the time and effort needed from the user to find the needed product/service, and ultimately raises the satisfaction of the customer.

Recommendation systems also benefit the providers, as it increases sales and revenue and increases the competitive advantage of the company using it by improving the customer experience.

Today recommendation systems are being used almost everywhere, and even the smallest improvement on these systems would hugely impact the competitive market.

There are many approaches to creating a recommendation system. We will discuss two of the basic approaches in simple terms below and then go into full detail further on.

Content-based collaborative filtering compares items' attributes and finds similar item sets so that if the user was satisfied with one of those items, we can recommend other similar items.

Item-based collaborative filtering predicts the users' rating for an item, based on the similarity of his/her ratings on other items to other users' ratings on the same items and recommends the items with the best-predicted ratings.

In this paper, we tried to combine two of them to include both the similarity of the nature of the items and the similarity of users' tastes which will result in a better outcome.

We proposed a method by which we first use the content and the attributes of queries to measure their similarity to other queries, and then we use collaborative filtering for similar item groups to predict the ratings. After all the required ratings have been predicted, we recommend N queries to each user based on 2 different methods, depending on whether the user is a new user or not. In this method problems such as "sparsity", "item cold start", "user cold start" and "overspecialization" are handled.

Furthermore we designed an experiment to see the effect of the number of users and queries and the size of the utility matrix on our results; and found out since the sparsity of our data is static throughout the tests, when increasing the size, the error measures increase too. however, because the amount of data increases too, the speed of the error measure increasing, decreases. on the other hand the speed of runtime works oppositely as it increases more.

The challenging part of developing a solution was that you can never have all the strengths without the weaknesses; and based on a situation any method can be better than another one. and if you solve a problem you might create another one. for instance, in our method for user cold start, we recommend some popular queries from different clusters. so we might have solved the cold start but on the other hand we created a problem where the system is biased toward popular queries.

*Both authors contributed equally to this research.

2 PROBLEM STATEMENT

2.1 Problem description

In a system, there are multiple users and multiple queries, each query consists of multiple tuples, and each user can pose a specific query and rate it. For each user and each query, if the query has not been rated by the user, predict the rating. Furthermore, recommend a set of queries with the highest predicted ratings for each user.

2.2 Inputs

- A relational table where each row is a set of tuples containing values and the first row contains the name of the fields (attributes).
- A set of users containing one user ID per line.
- A set of queries posed in the past where the first element of each row is the query ID and each of the other elements contains an attribute and its value. (e.g., Q13, Type=Movie, Genre=Horror)
- A User-Query utility matrix containing certain combinations of users and queries, a value from 0 to 100 indicating how happy that user is with the answer set of that query. the first row contains query IDs. As for other rows, the first element is the user ID followed by either an empty value or a value between 0 to 100.

2.3 Outputs

- A utility matrix, with the same structure and base values as input, however, with predicted values instead of null values.
- a set of recommended queries for each user.

3 RELATED WORK

3.1 Recommendation system

A recommendation system is a type of artificial intelligence system that utilizes algorithms and data to suggest items (such as products, music, or movies) to users based on their preferences and behaviors. The goal of a recommendation system is to improve the user experience by providing personalized and relevant suggestions that meet the individual needs and interests of each user.

3.2 Collaborative filtering recommendation system

Collaborative filtering is a well-known approach used in recommendation systems based on user preferences and behaviors. this approach assumes that user preferences do not change over time and will be most likely similar.

There are two main types of collaborative filtering:

3.2.1 User-based collaborative filtering.
This approach finds users similar to the target user based on the similarity of their ratings on the same items.

3.2.2 Item-based collaborative filtering.
This approach finds items similar to the target item based on the similarity of their ratings by the same users.

Collaborative filtering can capture complex relationships between users and items and does not require information about the attributes or content of the items. However, struggles with the "sparsity" and "cold start" problems and can be biased toward popular items.

3.3 Content-based filtering recommendation system

Content-based filtering is another popular approach that predicts item ratings by finding similar items regarding their content or attributes.

Content-based filtering doesn't require cooperation from other users and could recommend items that are not popular but fit the user's preference. However, is limited by the information provided in the content and struggles with the "overspecialization" problem.

3.4 Hybrid-filtering recommendation system

Hybrid-filtering is an approach that aims to combine both collaborative and content-based filtering's strengths and cover the weaknesses of both approaches to produce better results. Generally, there are two ways this algorithm is designed:

(1) Using both methods separately and combining the generated results.
If X and Y in order are the predicted ratings using collaborative filtering and Content-based filtering, we can use a model such as the linear model below to generate the final rating (Z) where (a) is decided by Database attributes such as sparsity.

$$Z = aX + (1 - a)Y$$

Figure 1: Linear model example

(2) Combining and mixing methods
Another way of combining these approaches is to mix the steps of these approaches, for instance, in our method we initially use content-based filtering to find similar queries and then use collaborative filtering for each similar group to generate results.

3.5 Jaccard similarity

Jaccard similarity is a method to measure the similarity between two sets of data based on the intersection and the union of these sets. One of the most important advantages of Jaccard similarity over other measures is that it can handle sparse data. We can calculate the similarity between A and B using the formula below:

$$J(A, B) = \frac{|A \cup B|}{|A \cap B|}$$

Figure 2: Jaccard similarity formula

3.6 Nearest neighbour

Nearest neighbor is a method to find the top k closest in distance items to another item; where k is predefined by the user.

3.7 K-means

K-means is a popular clustering method used to create a k number of clusters. It selects k points of representations and allocates other points to these clusters while trying to keep the range of these clusters as small as possible.

4 SOLUTION

In this section, we address our algorithm's main steps and go through each step's detail. Algorithm 1 presents the main steps of our solution.

Algorithm 1 Hybrid filtering

```
1:  Create a Query-Query similarity matrix
2:  for each empty value in the utility matrix do
3:      a = get the similar query set
4:      b = get the users' ratings for the query
5:      c = get the avg rating of the query
6:      Call weighted sum function with a,b,c
7:  Cluster all queries with k means
8:  for each user do
9:      if Cold start for user then
10:         Recommend the top queries from each cluster
11:     else
12:         Sort filled ratings for the user
13:         Recommend the top N-2 queries from sorted queries
14:         Recommend 2 random high-rated queries from clusters
```

4.1 Creating the Query-Query similarity matrix (Line 1)

Regarding the query set, each query consists of many conditions or attributes, and some of these conditions can have multiple values. For instance, in our data set which consists of movies, the field "Actors" could have many values ("Actors = Actor 1, Actor 2, Actor 3, ..., Actor 20"). In this case, the system searches across queries and chooses the 5 most frequent actors.

Our first step is to separate these values into corresponding conditions in the scope of all attributes ("Actors = Actor 1", "Actors

= Actor 2", etc). Following this step, we obtain query profiles with their set of attributes (characteristics).

In the next step for each query, we calculate the Jaccard similarity of the profiles (attributes) of all queries compared to each other and create a Query-Query similarity matrix.

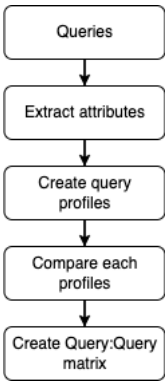

Figure 3: Similarity matrix creation steps

4.2 Predicting ratings (Line 2 - Line 6)

In this section, We need to predict the ratings for all the blank values in the utility matrix. To do this, we iterate through each of these blank values and do the following steps:

- Step 1: Obtain all similarity scores between the corresponding query and all other queries from the similarity matrix and select queries with similarity scores above average.
- Step 2: Obtain all ratings of the corresponding query from the utility matrix and normalize by subtracting the average.
- Step 3: calculate the average rating of the corresponding query in the utility matrix.
- Step 4: Calculate the predicted rating $\mathbf{p_{u,i}}$ using the weighted sum formula in Figure 4 and fill in the blank value.
  Where S is the set of queries similar to **i**, $\mathbf{s}_{i,j}$ is the similarity score between query **i** and query **j**, $\mathbf{r}_{u,j}$ is the rating of user **u** on query **j**, and $\mathbf{b}_{u,i}$ is the baseline prediction for user **u** and query **i**[1].

$$p_{u,i} = \frac{\sum_{j \in S} s(i,j)(r_{u,j} - b_{u,i})}{\sum_{j \in S} |s(i,j)|} + b_{u,i}$$

Figure 4: Weighted sum

We can see that the predicted rating depends on the similarity score between the query and its neighbor and the more the similarity score, the more their ratings will be similar. For this reason, to get more accurate and faster results, for (S) we want to choose only the most similar queries; therefore we only use queries with similarities above the average similarity.

The concept of our approach includes some techniques of content-based recommendation in it. In the weighted sum calculation, the system calculates the similarity of queries based on their attributes. Therefore, the item cold start problem when there is a new unrated item is automatically solved in this step.

4.3 Clustering queries

This extra step is mainly used to try to avoid problems such as the "overspecialization" and "user cold start" problems using clustering. For the overspecialization problem, we can use clusters to recommend 2 random high-rated queries to users from different clusters to maintain diversity in the recommendations. For the new user or user cold start problem, we can recommend the highest-rated queries from each cluster to the new user to gain an understanding of the user's taste.

In this step, we use k-means to cluster queries based on their attributes. We use the elbow method to choose a suitable number of clusters (k).

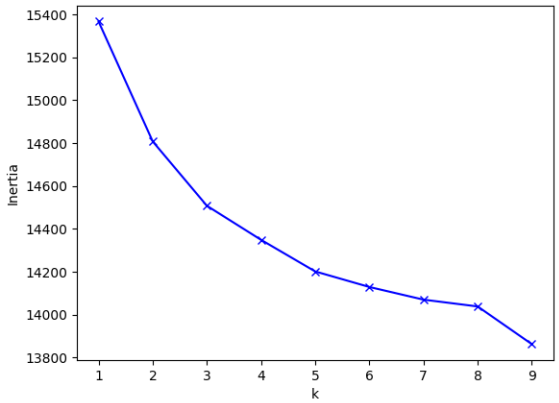

Figure 5: Elbow method - k

4.4 Recommending queries (Line 8 - Line 14)

After the prediction process and clustering, now we need to recommend to each user a total of N queries.

Initially, we divide users into two groups; users who have rated one or more queries in the past, and users who have never rated a query.

For the users who have rated at least one query in the past, we sort all their predicted ratings and recommend the top N-2 queries in descending order. in addition, we recommend 2 random high-rated queries from the different clusters created in the last section.

For the users who have never rated a query, we divide N by the number of clusters K and recommend the top N/k highest-rated queries from each cluster.

5 EXPERIMENTAL EVALUATION

5.1 Data-set properties

The data used in this project was received from FilmTV.it where each query represents a movie available on this website, with attributes such as the original title, year, genre, duration, country, director, actors, average vote, and votes.

5.1.1 *General properties.*

As shown in Table 1, this data-set contains 40,047 movies and 19 attributes. For the recommendation system, we minimized some of these attributes. For instance, the attribute "Actors" has an average of 102 characters in each movie. As a result, we calculated the frequency level of all actors and then chose the top 5 actors in each film.

Data-set	Items	Users	Queries	Total Attributes
FilmTV.it	40,047	2000	2000	19

Table 1: FilmTV.it data-set general specifications

5.1.2 *Sparsity.*

Figure 6 illustrates the sparsity of the utility matrix, we are dealing with approximately 81 percent sparsity; The higher this amount is the harder it is for the algorithm to get good results.

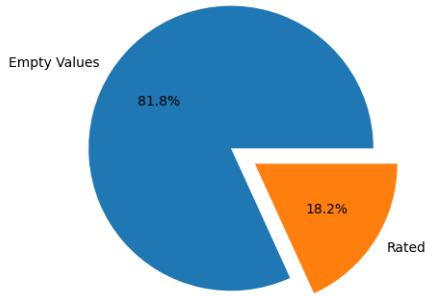

Figure 6: Sparsity of utility matrix

5.1.3 *Ratings.*

Based on the distribution of Figures 7, 8 and 9 we can confirm that ratings were distributed uniformly and as expected the average rating of queries and an average rating of users, are both centered around the average of all ratings in the utility matrix.

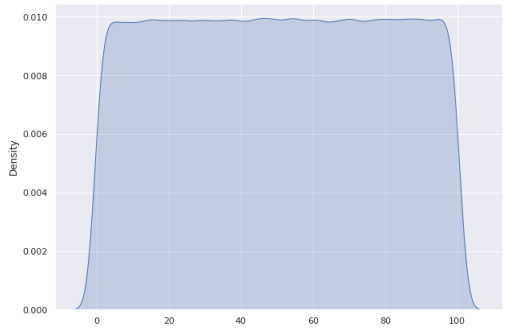

Figure 7: Ratings distribution in utility matrix

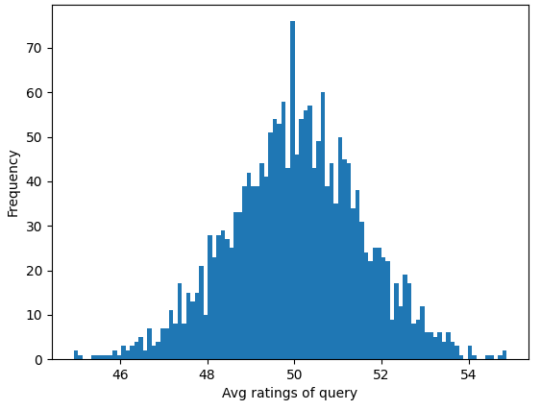

Figure 8: Average ratings of queries distribution

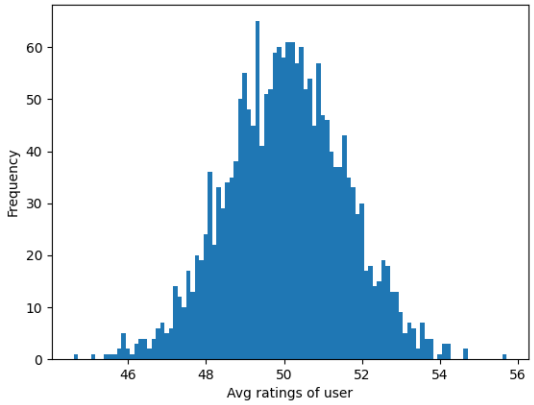

Figure 9: Average ratings of users distribution

5.2 Offline evaluation

For offline evaluation, we divide the utility matrix into 25% and 75% to test set and train set, respectively. Two testing metrics for measuring the recommendation quality are chosen which are Mean absolute error (MAE) and Root Mean Squared Error (RMSE)[1].

MAE takes the mean of the absolute difference between each prediction and rating for all held-out ratings of users in the test set and is computed as follows:

$$\frac{1}{n}\sum_{u,i}|p_{u,i} - r_{u,i}|$$

Figure 10: Mean absolute error (MAE)

RMSE has the effect of placing greater emphasis on large errors and is computed like MAE, but squares the error before summing it:

$$\frac{1}{n(r_{\text{high}} - r_{\text{low}})} \sum_{u,i} |p_{u,i} - r_{u,i}|$$

Figure 11: Root mean squared error (RMSE)

The results of the test are shown in Table 2 below, respectively[2].

Utility Matrix size	Sparsity	Number of blank spaces	MAE	RMSE
2000x2000	81.8	3.27M	21.47	25.48

Table 2: Offline evaluation result.

Since the original range of ratings is between 0 and 100 and the MAE and RMSE both are the same scale of the ratings, 21.47 and 25.48 are pretty good results considering the fact that the input ratings were generated completely random.

5.3 Experiments

In this section, we are going to compare Runtime, MAE, and RMSE on different scales of the utility matrix. We increased the size of the utility matrix and compared the results. For each size, we generated random data and tested the error metrics on the test set.

As seen in figure 12 MAE and RMSE increase with the size of the utility matrix. the reason behind this is that the sparsity of our data is static throughout the tests, so when increasing the size, the number of blank values increases too. Furthermore, because the amount of data increases too, the speed of the measure increasing, slowly decreases.

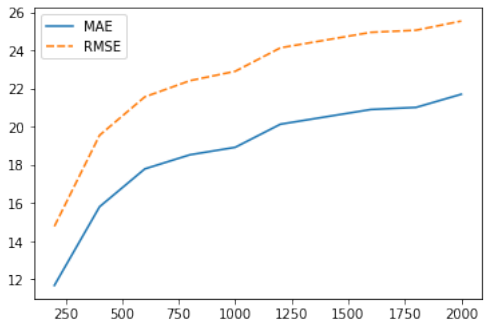

Figure 12: Experiment1

As seen in figure 13 the runtime increases with the size of the utility matrix as expected. furthermore, the speed of it increases, also increases, illustrating that for a huge amount of data, it might cause a problem.

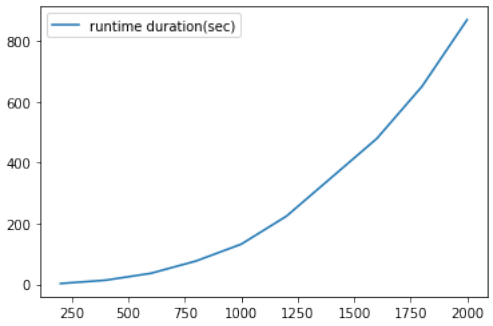

Figure 13: Experiment2

6 CONCLUSION

Recommendation systems have become a huge part of the competitive market and very. any slight improvement to these systems could make a drastic change for companies.

In this project, we tried to make a hybrid recommendation system which combines collaborative and content-based filtering's strengths by initially using the content of queries to find similar queries and then use collaborative filtering for each similar group to predict the ratings. furthermore, by using clustering we tried to solve some issues such as cold start for users and overspecialization.

We also experimented to see how the size of the utility matrix affects the metrics used to measure the errors and the runtime, and found out since the sparsity of our data is static throughout the tests, when increasing the size, the error measures increase too. however, because the amount of data increases too, the speed of the error measure increasing, decreases. on the other hand the speed of runtime works oppositely as it increases more.

REFERENCES

[1] Jon Herlocker, Joseph Konstan, Loren Terveen, John C.s Lui, and T. Riedl. 2004. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems* 22 (01 2004), 5–53. https://doi.org/10.1145/963770.963772

[2] Lalita Sharma and Anju Gera. 2013. A survey of recommendation system: Research challenges. *International Journal of Engineering Trends and Technology (IJETT)* 4, 5 (2013), 1989–1992.