Design of AI Systems (DAT410)
Assignment 2: Recommender Systems

Luca Modica
Hugo Manuel Alves Henriques e Silva

January 29, 2024

1 Reading and reflextion

1.1 Introduction

As the start of the document refers, recommendation systems suggest items of interest and enjoyment to people based on their preferences. Netflix, being an on-line movie subscription rental service, greatly benefits from a cohesive recommendation system, since the less accurate movie suggestions are, the less likely are people to continue using the service. Obviously, by being one of the most popular movie rental services on the Internet, Netflix has a lot of data to work with and receives daily over 2 million ratings. The goal of this project was to improve the accuracy of the Netflix recommendation system by 10%.

1.2 First Design Feature: Data

Some of the main challenges of this project are based on the data. The used dataset was large and sparse, meaning that there was a huge portion of missing data because most users would not rate the movies, which lead to the number of ratings being way smaller than the number of possible ratings. This meant that the data was very unbalanced, and that the system would have to be able to predict ratings that are not present in the data. As referred in the document: "*almost 99% of the potential user-item pairs have no rating*".

Contrary to the data of this recommendation system, the datasets of **music recommendation systems may differ**. It is easy to state that the duration of a music is much smaller than the duration of a movie, which causes the existence of many more data points compared to a movie recommendation system. Moreover, it is easier to capture different types of feedback from a user in a music recommendation system, from adding the song to the library or playlist, to skipping the song, to even listening to the song multiple times. Such things are harder to mimic in a movie recommendation system, since people are less

likely to watch a movie multiple times or even skipping it after a few minutes of watch time.

1.3 Second Design Feature: Models

In order to address the issues of this dataset, a variety of models that complemented each other were used, as singular models could not perform well enough. Some of the models used were neighbourhood models, which are able to predict ratings based on the ratings of similar users and SVD models, which are able to predict ratings based on the ratings of similar movies. Also, models that incorporate more than the ratings themselves were used in order to know more about users' behaviors, such as which movies were rated, despite the rating itself.

In addition, regularization was used in order to prevent overfitting, which is a common problem in recommendation systems. As the used models include a lot of parameters, many of which must be estimated based on smaller number of ratings, we are prone to overfitting.

Considering again the example of a music recommendation system, this can be handled and modeled by time-series models, which are able to capture the time dependency of the data. Properties related to skipping a song or listening to it multiple times can be captured by time series models such as Long Short-Term Memory (LSTM) Network, which can capture the sequential listening habits of a user.

Regarding overfitting, this issue might not be as pronounced in music recommendation systems, since the interaction data might be denser in music recommendation systems. This means the models will have more data to work with, i.e., more user actions per item. However, overfitting may still occur such as in cases of more popular and trendy songs. Dropouts and regularization are techniques that can be used to prevent the model from memorizing the data, and instead learn the underlying patterns.

2 Implementation

Also based on the context of the readings, in this section we will describe our implementation of a system for movie recommendations. The available data for the development are the following.

- `user_reviews.csv`: each row represents a user, while each column a movie. Each value represents the rating a specific user will give to a specific movie, ranging from 1 to 5; a value of 0 will be assigned if a user hasn't rated a movie yet.
- `movie_genres.csv`: the rows represent the movies in `user_reviews.csv`, while each column represents genre features for a movie. The values are boolean ones, 1 if the movie is part of a specific genre and 0 if not.

The main goal of this recommendation system is to suggest 5 movies for each of the first 5 users of the dataset (Vincent, Edgar, Addilyn, Marlee and Javier) that they haven't rated already. What will be described below is the approach to compute proper suggestions required by the task; this followed by the strength and weakness of the system, in order to justify possible future improvements.

2.1 Our approach: memory-based collaborative filtering

The recommender system implemented follows a **memory-based collaborative filtering (CF) approach**. We use historical ratings of users (that is, the provided data) to find similar users or movies; then, we are able to recommend movies that similar users liked or interacted with. In this case, the criteria chosen to recommend items to a user is based on similar ones ("Users who are similar to you also liked..."); this approach is also called *user-item collaborative filtering*.

At this point, a way to quantify the effective similarity is needed. The metric used to compute this information is the cosine similarity: It allows to compute the similarity by measuring the angle between 2 vectors. In this context, the users are represented as vectors in an n-dimensional space, where "n" is the number of items in the catalogue. The interpretation of the value calculated is the following.

- if the angle is small (< 90 degrees), the cosine similarity will have a value closer to 1. This means that 2 vectors tend to be proportional, and thus the users preferences are very similar.
- if the angle instead is large (> 90 degrees), we will have the opposite case: the cosine similarity will tend to have a value close to -1 and the vectors will be opposite. In other words, the users preferences are very different.
- if we have an angle of 90 degrees, the cosine similarity will be 0 instead. The 2 vectors are orthogonal, which means that those users have no similarities at all.

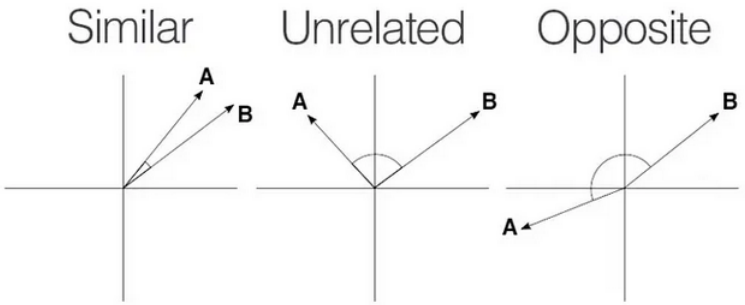

Figure 1: Geometric meaning of the cosine similarity (source).

Using this approach, the first step of the system in order to accomplish the goal is to compute the cosine similarity for each pair of users: It will result in a pivot table with all the possible user combinations, along with the cosine similarity value.

Using the matrix from the previous step, the next and crucial step is to calculate ratings for unforeseen movies: our design choice is to rely on the weighted average of ratings from similar users. This will then allow us to retrieve the list of the top 5 movies to suggest to the selected users. These 2 steps are wrapped and implemented in the following python method:

```python
def predict_ratings(user_name, user_reviews, user_sim_df, top_n=5):
    unrated_movies = user_reviews.loc[user_name][
        user_reviews.loc[user_name].isna()]

    # Predict ratings for each unrated movie
    predictions = {}
    for movie in unrated_movies.index:
        # Similar users who have rated this movie
        similar_users = user_reviews[movie].dropna().index
        similar_users = similar_users[similar_users != user_name]

        # Calculate the weighted average rating
        ratings = user_reviews.loc[similar_users, movie]
        similarities = user_sim_df.loc[user_name, similar_users]
        weighted_ratings = ratings * similarities
        if similarities.sum() > 0:
            predicted_rating = weighted_ratings.sum() / similarities.sum()
            predictions[movie] = predicted_rating

    sorted_predictions = sorted(predictions.items(),
        key=lambda x: x[1], reverse=True)

    return sorted_predictions[:top_n]
```

Listing 1: Function to compute `top_n` movie recommendations for a specific user, based on the cosine similarity matrix.

Results

Thanks to the implementation above, we were able to compute 5 unseen movie suggestions to the 5 selected user The lists are reported below.

- **Vincent**
  - Das boot

  - Never Back Down 2: The Beatdown
  - Fool's Gold
  - Jonah: A VeggieTales Movie
  - Eagle Eye
- **Edgar**
  - The Contender
  - Maid in Manhattan
  - Mad City
  - The Death and Life of Bobby Z
  - Get Carter
- **Addilyn**
  - Space: Above and Beyond
  - Dutch Kills
  - Highlander: Endgame
  - The Tempest
  - Amadeus
- **Marlee**
  - The Grandmaster
  - Big
  - Boiler Room
  - To Kill a Mockingbird
  - Vicky Cristina Barcelona
- **Javier**
  - Thirteen
  - Bad Company
  - Superman
  - The Good Thief
  - Home

Strengths

The approach used in our implementation comes with several advantages, which makes it a good starting point to develop a recommender system.

- **simplicity**: the cosine similarity is measure that is easy to understand and implement. Thus, It will make the output of the system much more intepretable.
- **Content analysis not needed**: as it can be noticed, the recommender makes no use of the dataset `movie_genres.csv`. With the approach used the analysis of the movie information is not needed, since it purely works on user settings.
- **Personalization**: the recommendations are purely based on the users ratings, thus tailored the user preferences.

Weakness

On the other hand, a recommender system which is purely based on users ratings to make recommendation can comes with weaknesses and limitations. Some of them will be described below:

- **Cold Start Problem**: in case a user is new to the platform, with few ratings it could be difficult to make accurate suggestions.
- **Scalability**: if we are dealing with a dataset with many users and/or many movies, the cosine similarity calculation can be computationally expensive.
- **Popularity bias**: even though the system is purely based on individual preferences, popular movies might be suggested more often. This will lead to overshadow niche movies, which can be more appreciated by the user.

Possible future development

Similar issues to ones described above can be common in a recommender system, especially the goal is to maximize the accuracy of the suggestions. As also the readings about the Netflix Prize Challenge described, there could be many approaches to address some the most common issues in this field. Down below some of them will be described, which can be applied to our implementation.

- A first improvement can be using a hybrid system combing the collaborative filtering approach and the content-based one (that is, more focused on the features of the items to be recommended). This can help address the cold start problem.
- for a better scalability and efficiency, a model-based collaborative filtering approach can be used. An example would be using factorization techniques like Singular Value Decomposition (SVD): they are able to learn the latent

preferences of the users and the latent attributes of the items, leading to an approximated but quite accurate rating predictions.

- To then reduce the popularity bias it can be useful to introduce diversity in recommendations, also based on an arbitrary criteria.

3 Discussion

Assessing the quality of a recommendation system before deploying it to users can be a hard task. Down below we listed the main reasons why It can be challenging: they especially related to limits we can notice to the algorithms used in this context (for example, collaborative filtering) and the dataset used for such systems.

3.1 Lack of Ground Truth

Since the users' preferences are subjective and can change over time, recommender systems lack a true "correct" answer for a prediction. In other words, compared to the usual Machine Learning classification task there is no clear ground truth (or correct label).

3.2 Data Sparsity and Cold Start Problem

As in the previous task's example recommendation systems have to deal with sparse datasets, which means that most items have little to no ratings from most users. This sparsity nature makes it way harder to make accurate predictions and to assess the quality of the system, due to lacks of data.

Moreover, new user or items have almost no interaction to the already existing data, leading to the same problem related to sparse matrices. This is called the cold start problem.

3.3 Diversity and Serendipity

A good recommendation system should be able to recommend items that are not only popular, but also diverse. In this way we can also recommended items that might spike a new interest. This is called serendipity. However, most of these "shot in the dark" recommendations will not be spot on what the user wants, making complex to measure how much they can increase the quality of the system.

3.4 Bias in the Data

Recommendation systems are trained with data that are often biased towards certain items, especially popular ones. This is a factor the systems has to take into account when making predictions. The over-representation of popular items

in the data can lead to a bias in the recommendations, which can be hard to detect: It is essential to account for these data imbalances.

3.5 Evaluating Long-Term and Short-Term Effects

A recommendation system that is optimized for short-term effects (such as click-through rate (CTR)) might lack performance in maintaining users active and engaged in the long-term. It is not only important to keep users active and collect their feedback, but also keeping in mind the importance of making them come back to the system in the future; this is done by making useful and relevant recommendations and measure the user satisfaction changes over time. Overall, long-term user satisfaction is where the challenged lies, because it is harder to quantify and as its own name suggests, it takes time to measure. So changes in the system might take a while to be noticed.

3.6 User Context and Dynamic Preferences

Users' preferences are not static, they change over time. Depending on the time of the day, the mood of the user, the people who are around them, the weather, etc. the user might be more inclined to watch a certain movie or listen to a certain song. Dynamics like these are hard to capture as they are not always present in the data. An accurate recommendation today might not be the most accurate one for tomorrow.

3.7 Offline vs Online Evaluation

The use of offline evaluation metrics (historical data) might fail to capture real time patterns, user behaviour, feedback and interactions with the system. Therefore, there is not a direct translation to online performance (real-world performance).