DAT158 – Movie Recommendation

Erlend Ranheim, Åsmund Vassbotn, Truls Bernås

DESCRIBE THE PROBLEM

SCOPE

- The goal of the project is to develop a website that provide movie suggestions to users based on their past interactions and previous liked movies. Recommendation systems is highly relevant across lots of different industries, be it Streaming, shopping, newsplatforms or social media. If the systems are less effective, less relevant media gets seen by the users, making for a worse user experience. But for our small project, the goal is to make a small handy application that users can use to get movie tips. But just from this small project, we have learned a lot about recommender systems.

- The solution we produced will be used in a webapp so that users can provide input ratings of movies, and then the webapp will produce a list of other movies that the system calculated as the best fit. The way people find movies today is to use search engines, recommendations from other people, movie databases like IMDB or people use streaming services and get recommendations directly from the platform. If one would do the task without machine learning one could implement rule-based algorithms that sort movies based on genres, actors etc. And then sort that list based on the users preferences.

- For our small project, a good way to measure the performance of the system is to make the user watch one of the recommended movies, and then have the user give a rating of the movie. When running the program once more with extra reviews, the model will give better recommendations. We could also implement a Clickthrough rate for our recommended movies, that would show what movies gets clicked on by our users, showing the most relevant movies.

- Our model relies heavily on the historical rating of movies from a lot of users from Letterboxd. If we want to keep the model updated, we continually want reviews from users. If the structure of the reviews from letterboxed changes, from 1-10 to for example 1-5, we have to alter our algorithm, but that would be difficult considering the importance of the historical data.

- The stakeholders of the project would be the users of the application and the developers of the application, which is us.
- Given the start of the project, which was give to us at the end of September, we have had around 45 days to complete the project.
  - Milestone 1

    - Define project idea and begin thinking about ways of implementing it in theory
- Milestone 2
    - Find datasets to use, and models that might work on the datasets.
- Milestone 3
    - Import data and clean the data so it is possible to use in the training of our model
- Milestone 4
    - Choose a module to use on the data to train a model
- Milestone 5
    - Build the model
- Milestone 6
    - Evaluate the model
- Milestone 7
    - Create webapp to allow user interactions with the model

- The required resources for the project can be deduced to time, hardware, software and personell.
    - Based on the amount of work that has to go into the project, a timespan of 45 days is enough to finish.
    - We needed computers which could handle the size of the datasets and manage the training process in an adequate amount of time.
    - After a bit of trial and error, we decided to use these softwares to complete our project. JupyterNotebook for writing the code to our model, github for code sharing, and Gradio for frontend representation.
    - A team of three students to work on the project

METRICS

- We would consider the project to be a success if are able to make a website that successfully provides good recommendations to users. Additionally it would be nice to get the predictions given to the users in a small enough timeframe.

- To measure the effectiveness of our recommendation system, we utilized Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE). These metrics asses how well the predicted ratings from the model align with the actual user ratings of the model.

  RMSE punishes big deviations between the predicted and actual values significantly more since the differences are quared. Using this metric, we are able to see if the model makes very bad predictions, as worse predictions are punished more heavily.

  MAE shows the average magnitude of the absolute errors. These values are not squared, so it shows a more straightforward view of the prediction deviance.

By measuring the RMSE and MAE values against each other, we can get a general idea of how to the model predicts movies. Since the rating system is ranging from 1 to 10, we want a RMSE and MAE score as close to zero as possible, indicating that the model correctly evaluates a suggested movie score as close to the users actual score. Our model performs on a MAE score of 1.07, and a RMSE score of 1.41. By understanding these values, we can finetune the parameters in the model even further to make these values as low as possible. In reality, the closer the actual rating we get, the more satisfied the user will be with the suggestions.

DATA

The data we used was collected from a dataset on Kaggle. This dataset is the result of scraped publicly available data. The data is taken from the top 4000 users (any given month) on a movie rating website called Letterboxd.

The data collected includes three separate datasets.

- Movies, which contain information about ~285 000 movies. The features include general things like name, id, genres, popularity etc.
- Users, which contain the publicly available user information like username, display name and number of reviews. This dataset contains ~8000 entries.
- Reviews, which is a large dataset with ~11 million reviews. Each review is a combination of the user id, movie id and the rating given to the movie by said user.

The main dataset used to train the model is the reviews which has 11 million entries. We believe this is sufficient to train our model. If necessary additional data could be gathered through additional scraping, though this was not done due to the existing size of our dataset.

If supervised learning was applied the ground truths would be attained directly from the reviews-dataset. A 7-to-10-star rating will be deemed positive while a 1-to-3-star rating will be deemed negative. A 4-to-6-star rating is neutral. To ensure accuracy the data has to be examined and filtered. For example, the image of a movie is not relevant to a movies quality and should therefore be removed.

The dataset is scraped from publicly available sources. The people who left the reviews have chosen to do so in a public forum and are accepting the terms that come with it. Therefore, there are no ethical considerations to be taken into account. If a user profile contains sensitive information that is the users concern and responsibility, not ours.

The reviews dataset does not have a lot of columns and therefore does not need much work. However, there is a potential need for balancing the dataset. If there are for example a lot of positive reviews but just a few negative reviews this needs to be addressed, and the dataset adjusted. The two other datasets have data that is not necessary for the training of the model and should be removed.

MODELING

We decided to explore a couple of different models that is designed for recommendation systems.
We could have gone with a content-based model using Linear Regression, only using features of movies to generate suggestions. This is a more simple approach which would work. But we quickly decided that we wanted an algorithm that used collaborative filtering to generate movie suggestions. This is because we believed that a model that used user data to recommend movies would give us a more realistic set of predictions.

When choosing a collaborative filtering method there are several choices. User-based Collaborative Filtering using K-Nearest Neighbors(KNN). This model will match one users tastes to another users taste and recommend movies that the user liked. Item-Based collaborative filtering using KNN, where predictions are given to users who liked the same movies. And Matrix Factorization Models using Singular Value Decomposition(SVG) or Alternating Least Squares(ALS).
Where the User-based and Item-based collaborative filtering models are memory based, it is not an efficient way to make our webapp, as it is not scalable for large datasets, like ours. Matrix Factorization models on the other hand is model-based, which means that it tries to learn the patterns between the rating and the movies. It does this by making a matrix based on the users preferences and the movies.

We also considered a hybrid model that used both a collaborative filtering model and content-based model. This way we could implement movies that other user liked, and also recommend newer movies without as many ratings or movies with less ratings.

We ended up using SVD (Singular Value Decomposition) implemented using scikit-surprise, considering that it was easier and more realistic to implement the a hybrid model.

DEPLOYMENT

We attempted to deploy the model to HuggingFace so we could share the application through a link. We did however, run into several technical issues when perusing this, so we backtracked to having the application run locally. The project must be cloned to the user's own machine and then be run. The user can then insert some movies they have seen, rate them, and get a movie recommendation.

To improve our system there are several things to be done. Our model is solely trained using collaborative filtering which has several drawbacks. The cold start problem is a known problem of our approach. If a new user joins it has no reviews nor preferences to base its predictions on. Therefore, it will often give bad recommendations. It is also less scalable since it requires more computation as the number of movies and users increases. Models like ours are also known to be popularity biased. The model will always choose movies with many good reviews and will not recommend movies with few reviews. Many niche movies will never be recommended which is a shame for the users who would enjoy these movies.

To fix these problems we would convert to a hybrid model which includes content-based filtering. The model would then take factors like genres of the movies into consideration when recommending a movie.

Currently the data is stored locally, and the application is running locally. The next logical step would be to move the data to its own database and get the application running on a separate server. Additionally, as movies are recommended to people and they rate said movies, the data should be collected for future testing and retraining of the model.

REFERENCES

- Scraped data from Letterboxd on Kaggle. Collected on 10.11.2024 from Kaggle: https://www.kaggle.com/datasets/samlearner/letterboxd-movie-ratings-data/
- Scikit documentation: https://scikit-learn.org/0.21/documentation.html