CSCI 5525 Project Final Report: Building a Movie Recommendation System

Becca Dura

1 Project Overview

Since recommendation systems are becoming so vital to the success of many companies, such as Netflix and Amazon, I have decided to compare various methods used when creating recommendation systems. The 3 main types of methods used to create these systems are popularity methods, content-based methods, and collaborative filtering methods; all of which are described in the succeeding sections. To compare these methods, I built nine recommendation system models using variations of the three methods that all aim to recommend movies to users and maximize a user's happiness with the movies recommended to them. While there has been a lot of research into each individual method, there is not a lot of existing research that actually compares the different methods once the models are built and their parameters are tuned, which was a large motivator for this project.

2 Data

The MovieLens 25M dataset from the GroupLens Research Group at the University of Minnesota [1] was used in conjunction with a dataset from Kaggle that contains metadata for all of the movies in the MovieLens dataset [2].

Due to limited computing resources, I had to subset the MovieLens dataset. Since it is easier to properly compare how well the models are performing when there is more labeled data, in this case more ratings by users for movies, I chose to subset the dataset to include movies with a fair amount of ratings (at least 14,500 ratings) and users that had rated most of those movies (at least 250 movies). This led to my dataset containing ratings from 1,482 distinct users for 335 distinct movies.

3 Popularity Models

Popularity models are very simple models used to recommend the overall most popular movies to a user without taking into account any information about specific users or movies. These models are not typically used by companies, but I created them as a base-line for this project. Two different popularity models were created. The first model simply finds the average rating for each movie and recommends the movies with the highest average rating to each user. The second model computes a weighted average for each movie to account for the fact that some movies may not have a lot of ratings. The formula for computing that weighted average, which is used by IMDB, is below:

$$W = \frac{Rv + Cm}{v + m}$$

where W = the weighted average of the ratings, R = the average rating for the movie, v = the number of ratings for the movie, C = the average rating across all movies, and m

= the minimum number of ratings required for the movie to be considered in the list of top recommendations (10,000 ratings was the minimum used in my model).

4 Content-Based Models

Content-based models determine how similar movies are to each other and recommend movies similar to movies a specific user has already enjoyed. I built two different content-based models by merging the MovieLens dataset with the Kaggle movie metadata dataset and using TF-IDF (term frequency-inverse document frequency) scores, which aim to determine the relevance of each word in each "document" in a collection of "documents". The formula used to compute a TF-IDF score for a specific word in a specific "document" is below:

$$w_{i,j} = tf_{i,j} * \log \frac{N}{df_i}$$

where i is a word, j is a document, $tf_{i,j}$ is the number of times i appears in j, df_i is the number of documents containing i, and N is the total number of movies.

Both of those models were built using the TF-IDF Vectorizer function in the Sklearn library in Python, because it allowed me to easily use both unigrams and bigrams, meaning I could look at individual words (unigrams) and consecutive word combinations (bigrams) that appear frequently in the "documents".

The first model calculates the TF-IDF score using movie summaries (i.e. movie summaries are considered the "document"), and the second one calculates the TF-IDF score using the genres each movie is classified under. After that, both models compute the cosine similarity score (formula to compute this is below) between the TF-IDF scores of each combination of 2 movies and recommend movies similar to a specified movie. For these models, I set up the models so they use one of the user's highest rated movies to find similar movies that are then recommended to that user.

$$CosineSimilarity(A, B) = \frac{A * B}{||A|| * ||B||}$$

5 Collaborative Filtering Models

Collaborative filtering models determine how similar users' preferences in movies are by calculating the cosine similarity score between the movie ratings for each combination of 2 users and finding movies that similar users have enjoyed. I have created both a memory-based and a few model-based collaborative filtering models. The memory-based system uses k-nearest-neighbors to find the k most similar users to a specific user (i.e. hightest similarity scores with that user) and computes the average rating among those users for each movie. The movies with the highest average ratings are then recommended to the user. The optimal k value (30) was found by minimizing the RMSE (root mean square error; formula shown below) using a validation set, as shown in Figure 1 in Appendix A.

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

I also built some model-based systems using neural networks in Tensorflow. Both neural networks used mini-batches of size 32. The first neural network is very basic. The input layer contains the movie and user IDs and is followed by three dense layers with 128 units, 10 units, and 1 unit, respectively. Each of those dense layers is followed by a ReLU activation layer to

ensure some non-linear functions were used. Finally, the Adam optimizer was used to find the optimal learning rate (1e-5) by minimizing the RMSE using a validation set. When fitting the model to the dataset, I stopped when the loss no longer decreased by 0.01 within two epochs.

The second neural network was more complex, because it uses embedding layers to reduce the number of dimensions in the model. It does this by embedding the movies and users into separate layers, reshaping those layers according to the number of dimensions we are using, and computing the dot product between each of the reduced dimensions for the users and movies. The Adam optimizer was used with a learning rate of 0.001 to find the optimal number of dimensions (20) by minimizing the RMSE using a validation set. When fitting the model to the dataset, I stopped when the loss no longer decreased by 0.01 within two epochs. Figure 2 in Appendix A shows how the validation set RMSE changed as the number of dimensions increased for this model.

In addition to the memory-based and model-based systems, I built a hybrid model using matrix factorization, where SVD (singular value decomposition) was used for dimensionality reduction. The surprise library, a Python library containing many functions to help build recommendation systems, was used to help implement SVD and to find the optimal number of dimensions (60) through cross validation. Figure 3 in Appendix A shows how the validation set RMSE changed as the number of dimensions increased for this model.

Below is a table showing the optimal parameters as well as the RMSE for the training and testing sets for each model.

Model	Parameters	Train Set RMSE	Test Set RMSE
CF Memory-Based (KNN)	k = 30	1.19047	1.16224
CF Model-Based (NN)	lr = 1e-5	0.90068	0.92248
CF Model-Based (NN Embedding)	d = 20	0.53138	0.50656
CF Hybrid (SVD)	n = 90	0.69184	0.70355

6 Model Results

Three different measurements, which are described below, were calculated for each model after tuning the models' parameters to help better compare the models. After calculating each of those measures for each user, I took the average and reported it in the table below. When calculating those measures, I counted any movie where a user had given the movie a rating of at least 3.5 to be one they liked. I also set each model to only recommend 25 movies to each user.

- Precision is the percentage of results recommended to a user that the user would actually like. In other words, it is the percentage of movies recommended to a user that the user actually rated with at least a 3.5.

$$Precision = \frac{TP}{TP + FP}$$

  where TP = true positives and FP = false positives.

- Recall is the percentage of movies a user would like that are actually recommended to them. In other words, it is the percentage of movies a user rated with at least a 3.5 that were recommended to a user.

$$Recall = \frac{TP}{TP + FN}$$

  where TP = true positives and FN = false negatives.

- The F1 score combines precision and recall.

$$F1 = \frac{2 * (Precision * Recall)}{Precision + Recall}$$

One key thing to note with this project is that not all of the data was labeled, meaning not all users had previously rated every movie in the dataset. This is a large reason why many of the measures, especially precision, are smaller. It is also one of the largest sources of error within this project, because a user may end up loving a movie they have not seen before, but since they have not previously rated it, it is not counted as a movie that the user actually enjoys. This is why I subsetted the dataset the way I explained in the Data section above.

Below is a table containing the results and optimal parameters for each model.

Model	Parameters	Precision	Recall	F1 Score
1. PM Average	N/A	0.81420	0.11374	0.19758
2. PM Weighted Average	N/A	0.85042	0.11872	0.20627
3. CB Movie Description	N/A	0.00944	0.07502	0.01666
4. CB Genres	N/A	0.00935	0.07415	0.01651
5. CF Memory-Based (KNN)	k = 30	0.04653	0.11529	0.06524
6. CF Model-Based (NN)	lr = 1e-5	0.01209	0.49304	0.02354
7. CF Model-Based (NN Embedding)	d = 20	0.01439	0.60165	0.02804
8. CF Hybrid (SVD)	n = 60	0.01496	0.41834	0.02881

7 Model Comparisons

To compare the models, I mainly looked at the precision since I was interested in how many of the movies recommended to a user are movies that the user would enjoy, as that would maximize a user's happiness with the recommendation system. One of the first things I noticed was that the popularity method models had significantly higher precision than the other models. This makes sense since we are recommending the most popular overall movies to the users, and the most popular movies recommended are most likely to already be rated (since users have already seen them). Since they have more ratings, there is less of an issue of the ratings being assumed to be zero when calculating precision. This is why we cannot truly compare the precision of the popularity models to the precision of the content-based and collaborative filtering models without having all the ratings from each user for each movie.

However, we must still acknowledge how high the average precision is when recommending movies using the popularity methods, which is why I would still use this method for users with no previous ratings. Content-based and collaborative filtering methods cannot work at all with the cold start problem, which is when a user has not rated any movies before, because there is no data on which movies the user has previously enjoyed. Therefore, the model using the popularity method with a weighted average would be best when a brand new user is looking for movie recommendations, as the weighted average method performs slightly better since it also takes into account the number of ratings for each movie.

When we do have previous data on a user, it is clear that the collaborative filtering memory-based method, which uses k-nearest neighbors, is most likely to recommend movies a user will enjoy. This makes sense since collaborative filtering methods are looking at what similar users have enjoyed, which can be helpful when a user has not rated many movies and can help encompass a user's specific tastes and preferences better than content-based methods can. Content-based methods focus on finding movies similar to a movie a user has previously enjoyed, but if a user enjoys multiple genres of movies, content-based methods cannot easily take that

into account. Within collaborative filtering methods, it is likely that the memory-based model performs the best because it is the most simplistic model, which reduces the chance of it overfitting to the given data. It also does not try to reduce the number of dimensions of the data like the neural network with embedding and the hybrid models do, which leads to it being less likely that important components/features within the data get lost with dimensionality reduction.

Due to the cold start problem, I would recommend that a company looking to build a recommendation system that best recommends movies users will enjoy combines the popularity method with weighted averages and the collaborative filtering memory-based method with k-nearest neighbors. This system would work best if it used the popularity method for newer users with no, or only a few, reviews and used the collaborative filtering memory-based method for returning users with a fair amount of previous ratings.

8 Limitations and Future Work

One of the largest limitations in this analysis was already explained above, and that is simply the lack of labeled data in the dataset (i.e. actual ratings). This lowers the precision, recall, and F1 scores, as ratings are assumed to be zero if a user has not rated a movie. One way to look further into how this affects those scores in the future would be to look at the variance in those measures, as there will likely be more variance with more missing data. In addition to that, since I only had limited computing power with my personal laptop, I was unable to use more of the MovieLens dataset. In the future, it would be nice to re-run some of these models on more data if more computing power was available to me.

There is still plenty more that could be done with comparing popularity methods, content-based methods, and collaborative filtering methods when building recommendation systems since not a lot of research has been done to compare them. With content-based systems, it would be worth creating models that combined the similarity scores calculated from both genres and movie overviews, as they could have more predicting power together. It would also be worth looking into building a model that finds movies similar to more than one of the user's favorite movies, as that may help capture a user's specific movie preferences better than finding movies similar to only one of that user's favorite movies. In addition to that, it would be worth testing out more features with the neural networks, especially dropout, which could reduce dependencies between features and the potential for overfitting.

Finally, it is worth noting that when putting these models into production, an additional step would need to be added so only movies the user has not watched recently are recommended. While I did not use the timestamps for the ratings in my analysis, because I wanted to be able to measure how well the models were performing in general, those timestamps would need to be used when putting the models into production to ensure only movies a user has not seen, or at least has not seen recently, are recommended.

9 Conclusion

While there is still much more research that could be done to compare popularity methods, content-based methods, and collaborative filtering methods for recommendation systems, this project did reveal a lot in terms of which methods work better for certain scenarios and why. Overall, for a cold start problem where we are recommending movies to a new user with no, or only a few, ratings, it is best to use the popularity method with a weighted average. On the other hand, if a user has rated a fair amount of movies, it is better to use the collaborative filtering memory-based method with k-nearest neighbors to recommend movies that similar users have enjoyed.

10 References

[1] “MovieLens 25M Dataset.” GroupLens. N.p., 10 Dec. 2019. Web. 22 Sept. 2020.

[2] Rounak Banik. “The Movies Dataset.” Kaggle.com. N.p., 2017. Web. 22 Sept. 2020.

11 Appendix A

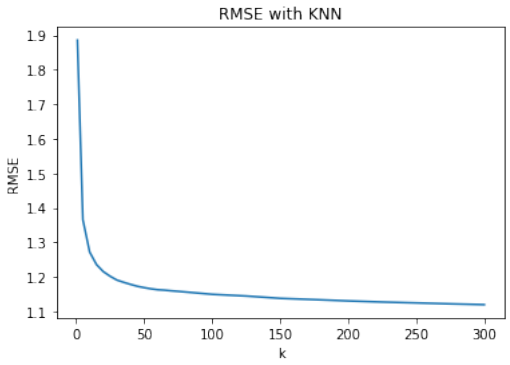

Figure 1: This figure shows how the RMSE varied as we increased the number of neighbors used in the k-nearest-neighbors algorithm for the collaborative-filtering memory-based model.

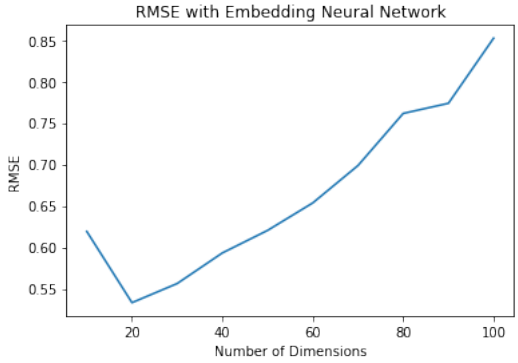

Figure 2: This figure shows how the RMSE varied as we increased the number of dimensions used in the collaborative-filtering model-based neural network with embedding.

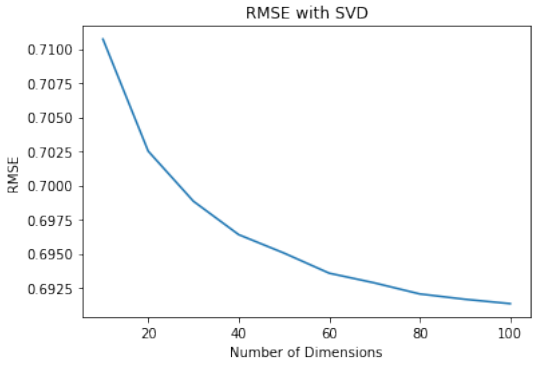

Figure 3: This figure shows how the RMSE varied as we increased the number of dimensions used in the collaborative-filtering hybrid model that used SVD.