Movie Recommendation System Project

Naveenan Yogeswaran (nry7)
Rutgers University
nry7@rutgers.edu

Rohit Rao (rpr79)
Rutgers University
rpr79@rutgers.edu

ABSTRACT

Our goal in this project was to develop a movie recommendation system from 'scratch' – that is, without the use of existing machine learning and artificial intelligence libraries. We developed this recommendation system for the MovieLens dataset. We integrated several different components of the data including userId, movieId, ratings, and genre of the movie as input features to our recommendation system. We built three models from scratch – a latent factor model, a content based model, and a hybrid model that utilized both the latent factor and content based models. The report goes into a more detailed analysis of our models and their performances.

KEYWORDS

Recommendation System, Latent Factor Model, Content Based Model, Hybrid Recommendation Model

1 INTRODUCTION

1.1 Background and Motivation

Recommendation systems are an important aspect of user experience in a variety of modern applications such as Facebook's friend recommendations, Amazon's product recommendations, Youtube's video recommendations, and Google's ad recommendations, just to name a few. These recommendation systems can often be very complex with the models that they use and the data that they utilize for input which can come from a variety of sources in order to make accurate and relevant predictions for their users.

The motivation for our project was to build two models – the latent factor model and the content based model – from scratch without using existing machine learning libraries so that we could better understand the details of their implementations. Then, we wanted to combine the results of those models to make a hybrid model that might be better than the individual model's themselves.

1.2 Problem Setting

We split the project into three major components. The first component was data preprocessing which was about splitting the data into training and testing data. The second component was about conducting rating prediction and evaluations in order to gauge the performance of our model. Lastly, the third part was about generating a top 10 movie recommendation list for each user.

1.3 Summary of Proposed Method

For the first component of data preprocessing we did an 80/20 split of the dataset where 80% was for training and 20% was for testing. This was done by going to each user and selecting 80% of the ratings for that user for training and putting the remaining 20% aside for testing. For the second part, we did rating predictions on the testing data and evaluated them using the Root Mean Square Error and Mean Absolute Error metrics. For the third component, we generated recommendation lists for users and tested them using the Precision, Recall, F1-Score, and Normalized Discounted Cumulative Gain metrics.

1.4 Information about the Dataset

We used the small version of the MovieLens dataset which was only 1mb. This dataset had a collection of approximately 100,000 ratings for about 9,000 different movies by about 600 users. There were many features in the dataset including userId, movieId, ratings, timestamps, title, genres, imdbId, tmbdId, and tags. However, throughout our project we only utilized the userId, movieId, ratings, and genre features in our models. Each rating given by users was between 0 and 5. Figure 1 below show the distribution of ratings. The genre category had 19 entries which are best represented by Figure 2 below showing the distribution sizes of each genre.

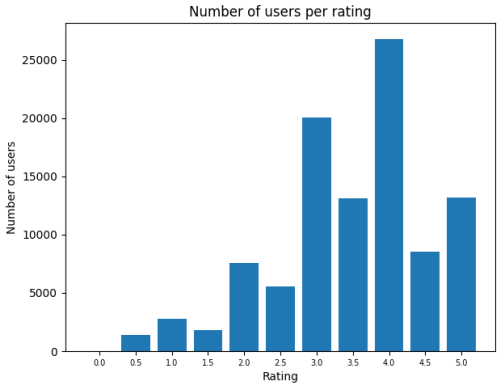

Figure 1

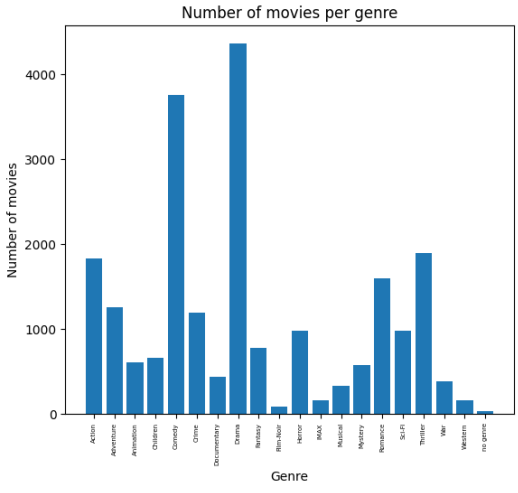

Figure 2

1.5 Summary of Experiment Results

We found that the content based model performed better on the MovieLens dataset than the Latent Factor Model. We also found that the mixed list generated by our hybrid model was better when compared to both the latent factor model and the content based model.

2 RELATED WORK

The ideas behind the latent factor models and content based models have been around for a while. In fact these models are so well studied that our textbook, *Mining of Massive Datasets* by Leskovec, Rajaraman, and Ullman, has detailed explanations of them, their use cases, and a variety of other information about these models and several others. [1]. Since these models have been around for so long, we wouldn't doubt that there probably exists other implementations of hybird models between these two models, but we did not refer to any such research papers during the course of our project.

3 PROBLEM FORMALIZATION

First, we will talk about the formalization of the rating prediction problem. The problem of predicting ratings can be thought of as generating ratings for users that are as close as possible to the actual ratings in the testing.csv file.

To measure this notion of "as close" we utilize the Root Mean Square Error and Mean Absolute Error with the goal of minimizing those errors. The root mean square error is $\sqrt{\frac{1}{n}\sum_i^n (p_i - r_i)^2}$ and the mean absolute error is $\frac{1}{n}\sum_i^n |p_i - r_i|$ where n is the total number of ratings, p is the predicted rating, and r denotes the actual ratings. The root mean square error is sensitive to outliers in the predictions which means it will spike for predictions that are quite far from the actual rating. The mean absolute error is not as sensitive to outliers but that doesn't necessarily make it purely better than root mean square error. Our goal is to try and minimize both of these errors as much as possible.

Next, we will talk about the formalization of the top 10 prediction list generation. This problem can be thought of as taking the rating prediction model and applying it to movies a user has not seen yet from the training dataset. Then, sorting those predicted ratings in decreasing order and taking the top 10 to generate the recommendation list. We measure the performance of the recommendation list using the precision, recall, f1-score, and normalized discounted cumulative gain which are described in more detail below.

Precision: In general, precision is defined as a measure of exactness which follows the formula: $\frac{|\text{true positives}|}{|\text{true positives}|+|\text{false positives}|}$. In the specific context of our problem, precision is defined as the percentage of testing items in the ten recommended items. We calculate precision for the model by calculating precision for each user and then taking the average over all users.

Recall: In general, recall is defined as a measure of completeness which follows the formula: $\frac{|\text{true positives}|}{|\text{true positives}|+|\text{false negatives}|}$. In the specific context of our problem, recall is defined as the percentage of recommended testing items in all the testing items for a user. We calculate recall for the model by calculating recall for each user and then taking the average over all users.

F1-Score: The F1-Score is a combination of precision and recall meant to almost compare the two. It's optimal value is 1 and it can be easily calculated using the formula $F = (2 * \text{Precision} * \text{Recall})/(\text{Precision} + \text{Recall})$.

Normalized Discounted Cumulative Gain: The normalized discounted cumulative gain is a metric that takes into account the position of items within the recommendation list itself. In order to calculate it, we first calculate the discounted cumulative gain which is given by the formula $DCG_p = \sum_{i=1}^{p} \frac{rel_i}{\log_2(i+1)}$ which we can see takes into account the different position of each element within the list. Then, we calculate the idealized discounted cumulative gain using the formula $IDCG_p = \sum_{i=1}^{|REL|} \frac{2^{rel_i}-1}{\log_2(i+1)}$ where items are ordered by decreasing relevance. The IDCG almost acts like an optimal or ideal dcg value. So, then the normalized discounted cumulative gain can be calculated using the formula: $NDCG_p = \frac{DCG_p}{IDCG_p}$ so that it is normalized between the interval $[0, 1]$.

4 THE PROPOSED MODEL

We explain the three models we used and how we created each of them below.

Latent Factor Model: With the Latent Factor Model, our goal was to initialize and optimize matrices p and q such that the product of a user's row in p and the movie's row in q will give the predicted rating that the user will give that movie.

First, we needed to map the user ids to indices ranging from 0 to the number of users. This can be done by subtracting the user id by 1. We also needed to map the movie ids to indices ranging from 0 to the number of movies. This can be done with a dictionary that maps the movie id to its corresponding index.

Next, we need to initialize matrices p and q. We initialize matrix p to have size uxn where u is the number of users and n is the number of factors. We then initialize matrix q with size mxn where m is the number of movies and n is the number of factors. We initialize the values in p and q to be random values between 0 and 1. We experimented with different number of factors and different initialization strategies which is shown in the Experiments section.

With matrices p and q initialized, we now need to update these matrices to be able to predict ratings. We can do this using stochastic gradient descent. Our goal is to minimize the loss function:

$min_{P,Q} \sum_{trainingdata}(r_{xi} - q_i p_x)^2 + [\lambda \sum_x ||p_x||^2 + \lambda \sum_i ||q_i||^2]$

We can do this using stochastic gradient descent by updating the vectors p_x and q_i using the gradient of the loss function whenever we examine review r_{xi}. We would update these vectors with all the reviews in the training data. We will also go through the training data for multiple rounds. We experimented with different number of rounds which is shown in the Experiments section.

In our model, we set λ to 0.1 and the learning rate to 0.01.

To test the effectiveness of the model, we used the RMSE and MAE values. We aimed for a lower value.

Content Based Model: In general, content based models exclusively use information about the items to derive predictions for a user. In our case, we utilized the genre information about each movie in order to make predictions and ratings.

First, we built some preliminary methods to help our calculations and keep track of different values later on. So, we built the

map_movie_id_to_index method again – similar to the latent factor model, we figured that it would be easier to have indexes corresponding to each movie rather than the random movieId's which had a very large range. We also built a map_genre_to_index method so that we could refer to each genre by an index rather than the genre name or string. Next, we built a get_num_of_movies_per_genre method to map the total number of movies in each genre because this is a quantity that is important in the movie profile calculation explained later. Lastly, we built a map_movie_index_to_genre_index method which maps movie indexes to a list of the genre indexes that the movie falls under.

After the preliminary methods, the first part we built was the movie profile vectors. These vectors are 19 elements long since there are 19 genre categories. In order to build the profile vectors, we read in all the movies in the training dataset along with the genres that the movie fell under. Then, for each element in the vector, we calculate its score using the term frequency - inverse document frequency (TF-IDF) formula where term frequency is defined as the number of times a term (genre) i appears in a document (movie) j and inverse document frequency is $\log(\frac{N}{df_i})$ where N is the total number of documents (movies) and df_i is the document (movie) frequency; the inverse document frequency is essentially a scaling factor based on how frequent a term (genre) appears in among all documents (movies). Running this TF-IDF formula for every movie generates a movie profile vector for every movie.

Next, we generated a user profile vector for each user in the training data. This user profile vector was a weighted average of the movie profiles for movies that the user has already rated – this means the average of the rating times the tf-idf vector for each movie.

In order to generate predictions for the ratings and recommendation lists we first needed a way to compare tf-idf vectors for movies the user hasn't seen yet with their user profile. We used the cosine similarity metric in order to make this comparison between vectors. The cosine similarity between 2 vectors is given by the formula $cos(x, i) = \frac{x*i}{||x||||i||}$. Then, generating predicted ratings for the testing data is simply a matter of comparing the user profile with the movie profile, getting the similarity score, and then multiplying by 5. Similarly, generating movies to add on the top 10 recommendation list is simply going through all movies a user hasn't seen yet, comparing the movie profiles with the user profile, sorting by the similarity scores in decreasing order, and then taking the top 10 as recommendations.

Hybrid (Mixed List) Model: In our hybrid model, our goal was to take our top N recommendation lists from our Latent Factor Model and our Content Based Model and merge the lists to create an improved top 10 recommendation list.

In order to merge the lists, we created a formula that will determine a new score for each movie in each of the top N recommendation lists provided by the Latent Factor Model and the Content Based Model. We call this new score the Mixed-score.

$Mixed - score = R * S * C^B$

In this formula, R represents the predicted rating for the movie from the Latent Factor Model. S represents the similarity score for the movie from the Content Based Model. C represents a constant value that acts as a multiplier whenever a movie appears in both top N recommendation lists. B is 0 if the movie is not in both top N recommendation lists, and 1 if it is in both top N recommendation lists.

The reason we multiply R and S together is because we want the Mixed-score to increase as our predicted rating gets higher and our similarity score gets higher. The reason we multiply by C^B is because we want to boost the ranking of a movie if it appears in both top N recommendation lists provided by the Latent Factor Model and the Content Based Model. If a movie appears in both lists, then this means that the movie matches the user's taste and the user is predicted to like the movie which means this should be a good movie to recommend for the user. The reason we don't include the movie's position in the list in the Mixed-score formula is because the position is implied in the R and S values. A higher R value indicates a better ranking in the Latent Factor Model list and a higher S value indicates a better ranking in the Content Based Model list.

To merge the lists, we first read the recommendations from the top N recommendation lists from our Latent Factor Model and our Content Based Model. We then go through each movie in both of these lists. For each movie, we calculate its Mixed-score using the data we have on its predicted score and its similarity score. We also make sure to check if this movie is in both lists in order to determine if we should apply the C^B multiplier. After we calculate its Mixed-score, we store the movie and its Mixed-score as a tuple in an unsorted list. After we went through all the movies in both lists, we sort the list holding the movie and Mixed-score tuples. We then take the top 10 movies with the highest Mixed-score as our final top 10 recommendation list for movies.

5 EXPERIMENTS

Latent Factor Model Experiments Results: Our first few experiments were done on the Latent Factor Model to determine how adjusting various parameters can affect the effectiveness of the Latent Factor Model. The parameters we tested were the number of factors, the number of rounds through the training set, and different initialization strategies for the p and q matrices. We used RMSE and MAE to determine how well the model performed.

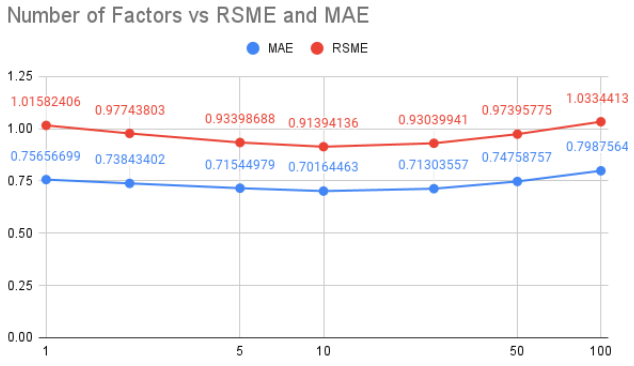

Figure 3

When testing the number of factors, we fixed the number of rounds to 10 and initialized p and q with random values between 0 and 1. We tested the values 1, 2, 5, 10, 25, 50, and 100. As seen in

Figure 3, we noticed that the MAE and the RMSE values decrease as the number of factors increase. This is expected as increasing the number of factors will allow the Latent Factor Model to get more information on each user and each movie. However, we noticed that after around 10 factors, the RMSE and MAE values decrease. This most likely occurred because the model began over fitting to the training data as a result of the many factors. Because of this, the model could no longer generalize to data outside of the training data.

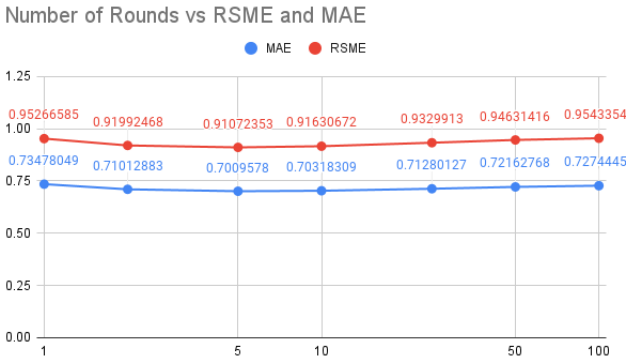

Figure 4

When testing the number of rounds, we fixed the number of factors to 10 and initialized p and q with random values between 0 and 1. We tested the values 1, 2, 5, 10, 25, 50, and 100. As shown in Figure 4, the RMSE and MAE decreased as the number of rounds increased. This is expected because we expect p and q to be further optimized by SGD with each round through the training data. However, we noticed that after around 5 rounds through the training data, the RMSE and MAE values started increasing as the number of rounds increased. Like with the number of factors, this is most likely due to over fitting. Because we are optimizing p and q using the same training data multiple times, we start over fitting to the training data. As a result, our model starts losing its ability to generalize outside of the training data.

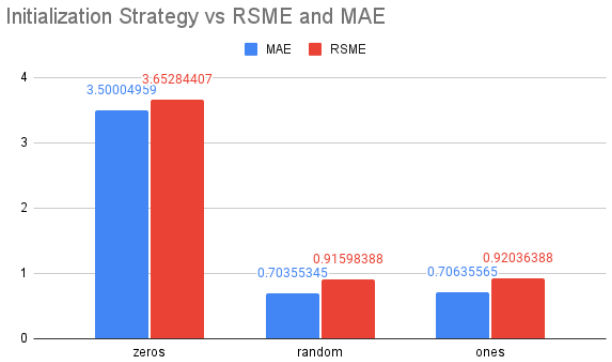

Figure 5

When testing the initialization strategies, we fixed the number of factors to 10 and the number of rounds to 10. We tested three initialization strategies. The "zeros" strategy initializes p and q with 0s. The "random" strategy initializes p and q with random values between 0 and 1. The "ones" strategy initializes p and q with 1s. As shown in Figure 5, the "zeros" strategy performed very poorly while the "random" strategy performed the best. The "ones" strategy also performed nearly as well as the "random" strategy. This most likely occurred due to the optimized p and q values being closer to 1 than 0. The reason the "zeros" strategy performed poorly may be due to 0 being too far from the optimal p and q values. As a result, there weren't enough rounds through the training data to get the values close to the optimal values. Meanwhile, the random values and the 1s were likely close to the optimal values and so they were able to converge to them faster.

Comparison of Models: To compare the models, we fixed the Latent Factor Model to have 10 factors, 10 number of rounds, and used the random initialization strategy. For the Hybrid model, we used the top 25 movie recommendation lists from the Latent Factor Model and the Content Based Model to produce a new top 10 movie recommendation list. We used precision, recall, F-measure, and NDCG to measure the models.

Model	Precision	Recall	F-Measure	NDCG
Latent Factor	0.003508197	0.001263519	0.001854065	0.441304367
Content Based	0.017704918	0.007792234	0.010755607	0.477866413
Mixed List	0.023868852	0.012317261	0.016182656	0.524932929

Figure 6

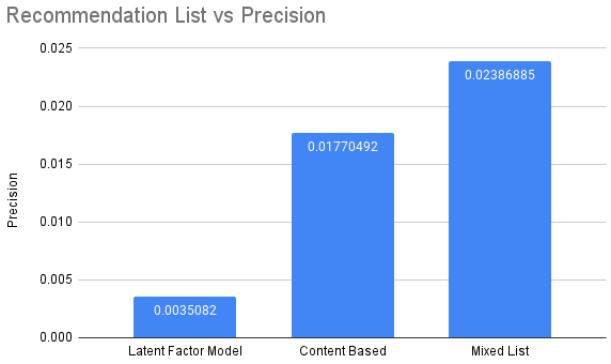

Figure 7

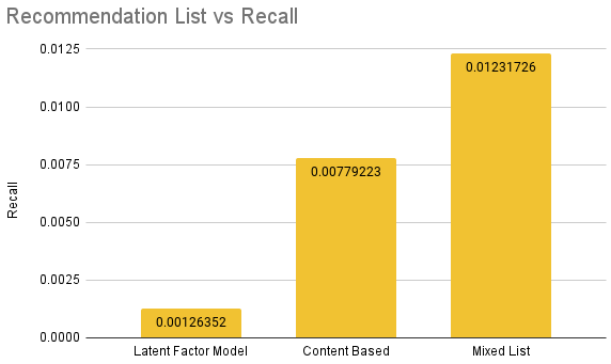

Figure 8

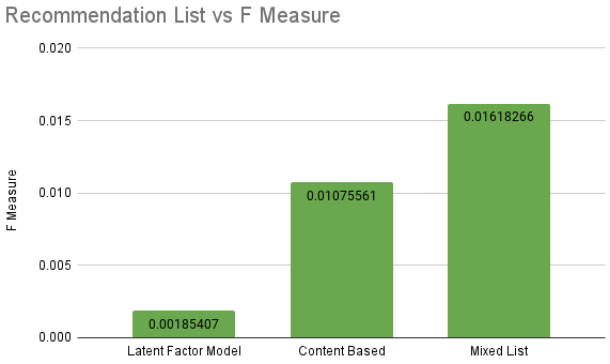

Figure 9

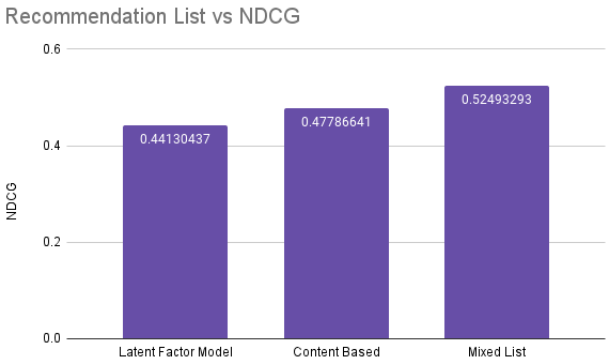

Figure 10

As shown in Figures 6-10, the Hybrid Model outperforms the Latent Factor Model and the Content Based Model in all four metrics. The Hybrid Model performed the best in precision which indicates that the Hybrid Model's list contained more movies on average that the user did end up watching according to the testing data. The Hybrid Model also performed better in recall which indicates that the Hybrid Model was able to recommend more movies that the user did end up watching in the testing data compared to the other models. As a result of its high precision and high recall, the Hybrid Model performed the best in F-measure. Lastly, the Hybrid Model performed the best in NDCG which indicates that the ordering of the rankings in the Hybrid Model's top 10 recommendation lists were also better on average. The reason the Hybrid Model works better than the other models is discussed in the next experiments.

Comparison of Different Incoming Recommendation List Sizes: We wanted to further experiment with the Hybrid Model by seeing how the sizes of the recommendation lists provided by the Latent Factor Model and the Content Based Model affects the final top 10 recommendation list of the Hybrid Model. For example, instead of using the top 10 recommendation lists from the Latent Factor Model and the Content Based Model, what if we use the top 25 recommendation lists? We wanted to determine if this will improve our results and if there are trade offs to using larger sized lists from the other models to produce our final top 10 recommendation list. The sizes we experimented with are 10, 15, 25, 50, 100, 500, and 1000.

Size	Precision	Recall	F-Measure	NDCG	Time(seconds
10	0.017344262	0.008209452	0.011086703	0.552293562	0.039134407
15	0.020786885	0.01006485	0.013451139	0.530209104	0.053011751
25	0.023868852	0.012317261	0.016182656	0.524932929	0.088814259
50	0.027967213	0.013933117	0.018573444	0.505410563	0.196135569
100	0.029508197	0.015197563	0.020051313	0.508440002	0.475085115
500	0.029704918	0.014311406	0.019289474	0.507372144	6.11726656
1000	0.029213115	0.014879193	0.019697255	0.507185978	21.12991753

Figure 11

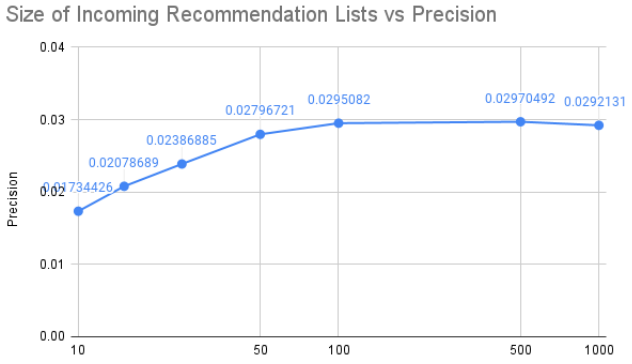

Figure 12

As shown in Figure 12, increasing the size of the incoming recommendation lists improves the precision of the Hybrid Model. This most likely happens because the Hybrid Model's strength is that it takes into consideration what the user is predicted to like and the user's preferences. So if a movie is ranked relatively high for both the Latent Factor Model and the Content Based Model, it'll receive a high Mixed-score in the Hybrid Model to help it reach the final top 10 recommendation list.

The reason increasing size increases precision is because it is possible that there exists a movie that is in the top 15 of both lists but not in top 10 of either. It is also possible that the movies that are in the top 10 of either list is not ranked high in the other list. So what happens is is that the movie that's in the top 10 gets put in the final top 10 recommendation list while the movie that was in the top 15 of both lists gets ignored because it didn't make the top 10 in either list. In reality, the movie that scored relatively high in both lists should be better for the user than the movie that scored really high in one list but not high in the other. By increasing the size of our incoming recommendations lists, we are taking those movies that didn't make the top 10 into account so they can get boosted to the final top 10 recommendation list if that movie appeared in both incoming recommendation lists and scored relatively high in both lists.

We also noticed that the precision flattens out after around a size of 100. There is even a decrease in precision as the size eventually gets to 1000. The reason the curve flattens out is most likely because the movies that ranked relatively high in both lists have already been taken into account with the smaller sizes. The reason for the slight decrease in precision around size 1000 is likely due to movies being ranked low to medium in both lists getting taken into account.

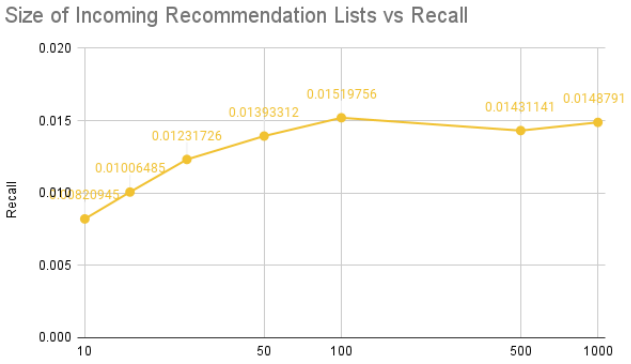

Figure 13

Similar to precision, the recall seems to increase and then slightly decrease as the size of the incoming lists increases. This can also be explained by the Hybrid Model's strength and the multiplier like in precision. By increasing the size of the incoming recommendation lists, we are taking into consideration the movies that were ranked relatively high in both lists but not high enough in either list. By taking these movies into account in our final top 10 recommendation list, we improved our top 10 recommendations which improved our recall. We also noticed the curve flatten out and slightly decrease which is most likely due to movies that score relatively high being considered already with smaller sizes and movies that were ranked low to medium in both lists being taken into account.

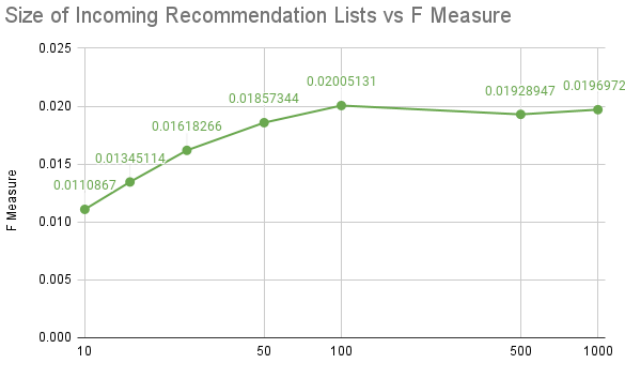

Figure 14

In Figure 14, the F-measure follows a similar trend as precision and recall. We notice an increase as the size of the incoming recommendation lists increases up until around size 100. After that, we see minimal changes to the F-measure as the size increases.

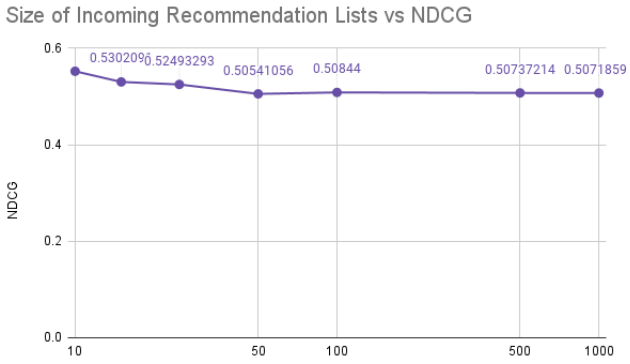

Figure 15

As shown in Figure 15, it appears that the NDCG decreases as the size of the incoming recommendation lists increases. So while our lists contains more correctly recommended items, the ordering of the items is not as good. One explanation for this could be that while the Hybrid Model is recommending more movies that the user does end up watching, its also recommending other movies that has a high Mixed-score that the user doesn't end up watching. By increasing the size of the incoming recommendations list, we are also taking more movies into consideration that may have relatively high scores in both models. However, some of these movies that have relatively high scores in both models may also be movies that the user doesn't end up watching. As a result, these movies may take the higher spots of the final top 10 recommendation list, which lowers the NDCG as the user didn't end up watching the movie.

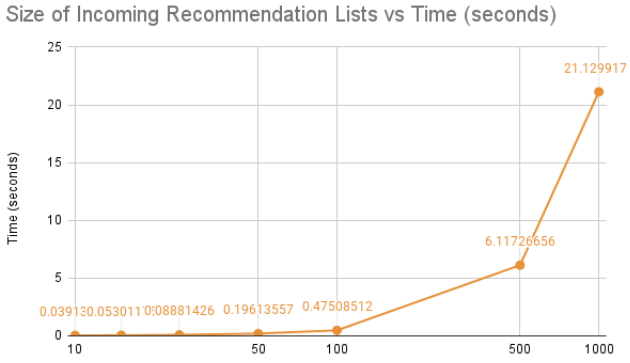

Figure 16

The time depicted in Figures 11 and 16 includes the time for reading the incoming recommendation lists, calculating the Mixed-scores, and taking the top 10 recommendations for each user.

As shown in Figure 16, the time in seconds increases as the size of the incoming recommendation lists increases. The time seems to increase at a slow rate when going from size 10 to 100; however, we notice large increases when increasing the size to 500 and 1000. This is most likely due to the amount of time needed to read through the incoming recommendation lists for each user and then calculating Mixed-scores for all those movies. Because of the amount of extra reading and calculations needed for bigger sizes, we notice larger increases in time as the size of the incoming recommendation lists increases.

6 CONCLUSIONS AND FUTURE WORK

Conclusions:

Latent Factor Model: Based off the experiments done on the Latent Factor Model, we came to a few conclusions. The first of which is that the number of factors is important when building the Latent Factor Model. As observed in our results, using too little factors will not give a good model as the model won't be able to get more information on the user and movie. Using too many factors has the problem of over fitting as the model will be over fitted to the data and will be less effective for predicting ratings outside of the training data.

The number of rounds through the training data is also important for similar reasons as the number of factors. Using too little rounds will make the model less effective as our p and q matrices wouldn't have gotten updated enough to be optimized, and using too many rounds has the problem of over fitting as the p and q matrices will have been fitted too much to the training data and is no longer as effective for data outside of the training data.

Lastly, the initialization strategy is also important. It appears from our results that a good initialization strategy will try to initialize values that are estimated to be closer to the actual values of p and q. Initialization values that are too far may suffer from not being able to converge to the actual p and q values after several rounds through the training data. On the other hand, initialization values that are closer to the actual values will converge better. In our experiments, it seems initializing random values in a range will typically give the best results.

Content Based Model: The content based model only utilizes information about the movies – this implies that there is no dependence on the user data. Therefore, we don't need a lot of users or even a lot of data about the users in order to generate results. One of the other benefits of the content based model is that we are able to recommend users with specific movie tastes. This also means we can recommend movies that are new or not very popular, but fit the user's profile – thus, there is no cold start item problem. However, one of the cons to the content based model is that it is hard to recommend movies to new users or movies outside a user's profile which means it is hard to expand a user's movie tastes.

Hybrid (Mixed List) Model: Overall, the Hybrid Model is able to mix the lists produced by the Latent Factor Model and the Content Based Model to produce a top 10 recommendations list that is better than the Latent Factor Model and the Content Based Model. We see this as the Hybrid Model outperforms in precision, recall, F-measure, and NDCG. The reason this model works well is because it takes the user's predicted rating and preferences into account. With this model, movies that score relatively high in both previous models are given preference to movies that scored high in one model, but not the other model. This helps improve the recommendations list as movies that score relatively high in both previous models are likely to be better recommendations since they have a high predicted rating and matches the user's preferences.

We also came to the conclusion that the size of the incoming top recommendation lists from the Latent Factor Model and the Content Based Model is also important in the effectiveness of the Hybrid Model's mixed list. In general, we want a large enough size so we can take into account movies that scored relatively high in both lists but not high enough to make it to the top 10 lists for either model. This way, we can possibly recommend these movies as they are likely better recommendations than movies that scored high enough to be in the top 10 list of one model but not high in the other model. However, we don't want to use a too large size as the time is takes to mix the lists will increase, and there is a chance that the overall quality of the list may decrease as movies that scored low or medium in both lists may make it to the final top 10 recommendation list in the Hybrid Model. Larger sizes also appear to decrease the NDCG, which decreases the quality of the ordering of the top 10 list.

Future Work: There are a few ideas for further explorations of the models presented. One of the ideas that was not entirely explored in this project was the content based model. There are numerous other methods to construct the movie profiles and user profiles. One future work would be to see how different constructions of the movie and user profiles improve or hurt the performance of predictions and recommendations. Additionally, we only tested the cosine similarity measure in our content based model. However, there are several other metrics that could be tested like euclidean distance and it would be interesting to see how the choice of similarity measure impacts predictions and recommendations. It would also be important to see how these changes to the content based model would impact the performance of the hybrid model.

Another idea that could be further explored would be to add other recommendation models such as user or item based collaborative filtering. We could then compare the performance of the new models with the latent factor model and the content based model. It would also be interesting to see if different combinations of hybrid models perform better than others or whether a combination of all models results in the best performance overall.

ACKNOWLEDGEMENT

In the early stages of our project, we took inspiration from the guidance of Professor Yongfeng Zhang for ideas on where to go with our project. We also utilized Professor Zhang's lecture presentation slides to gain more background knowledge on the models and how to implement them. [2] [3]

REFERENCES

[1] Jure Leskovec, Anand Rajaraman, and Jeffrey D. Ullman. *Mining of Massive Datasets.* Cambridge University Press.
[2] Yongfeng Zhang. 2022. Recommender Systems: Content-based Systems and Collaborative Filtering. University Lecture. (2022).
[3] Yongfeng Zhang. 2022. Recommender Systems: Latent Factor Models. University Lecture. (2022).