**Colab:**

<https://colab.research.google.com/drive/1d7Aq-m_v2fw9rS9m6vJzX9rq67cgar5t?usp=sharing>

**Report:**

Most of my knowledge for part 1 came from these twos sources:

<https://towardsdatascience.com/introduction-to-recommender-systems-6c66cf15ada>

<https://developers.google.com/machine-learning/recommendation>

1. **Recommender Systems Overview:**

Over the past few years, recommendation systems have been very popular and many companies, such as Youtube, Netflix, Amazon, and many more use them on a daily basis. In simple terms, recommendation systems are machine learning algorithms that recommend things that the algorithm will think people will want/enjoy. There are two main approaches to building a recommender system. The first is content-based filtering, which uses the similarity between items to recommend items similar to what the user likes. The second is collaborative filtering, which uses similarities between items and queries to provide recommendations. Both content-based and collaborative filtering use embedding vectors and in order to measure how similar two items are, both filtering methods use the idea of closeness defined by a similarity measure. A similarity measure is a function that takes a pair of embeddings and returns a scalar measuring their similarity. To calculate that degree of similarity, most recommender systems use one of the following: cosine (cosine of the angle between two vectors), dot product (dot product of the two vectors), or euclidean distance. Unlike the cosine similarity measure, the dot product is sensitive to the norm of the embedding.

So what are content-based and collaborative filtering? Content-based filtering uses item features to recommend more similar items to what the user likes based on the user’s previous history. The main idea of content-based filtering is to use the available features that explain the user-item interactions and build a model based on that. The advantage of content-based filtering is that the model does not need any information about other users and is, therefore, easier to scale to a huge amount of users. Moreover, the model can recommend unique items which few other users are interested in as the model captures the specific interests of the specific user. However, one drawback of content-based filtering is that the model can make recommendations solely based on the existing interests of the user. The model cannot expand on the user’s existing interests to show the user something the user has not seen before.

To deal with that issue of content-based filtering, collaborative filtering takes into account similarities between users and items at the same time to provide recommendations. Because of this, if there are two users with similarities, then the models using collaborative filtering are able to recommend something to user X based on the interests of a similar user B. Many use matrix factorization for a collaborative filtering model. There are a few advantages to using collaborative filtering models. The first is that they do not need any information about users or items as the embeddings are automatically learned. Additionally, the more users interact with the items, the more often new recommendations become accurate. Lastly, the model can recommend new content to the users, and this way users might discover new interests. However, one disadvantage of collaborative filtering algorithms is that the algorithm can only take into consideration past interactions to make recommendations. The algorithm cannot recommend any new items and it cannot recommend anything to new users.

The last type of recommender system I will discuss is the hybrid model. The hybrid model takes into account both content-based and collaborative filtering techniques. Therefore, hybrid models are able to make recommendations that draw from the advantages of both content-based and collaborative filtering methods.

1. **Reasoning for choice of algorithm:**

For the baseline model, the algorithm I decided to go with was the average of ratings. I decided to do this as this is what the requirement document said to do.

For the simple model, I tried three different model-based approaches to do collaborative filtering (the requirement document said to use collaborative filtering for the simple model). The reason I chose model-based approaches is that these approaches use machine learning techniques, such as SVD (Singular Value Decomposition) and matrix factorization and I wanted a model that took user and movie information and the relationship between the two into consideration. These models basically learn the embeddings and then use collaborative filtering.

The first model I used was an SVD model. My SVD model decomposed the matrix into singular vectors and singular values. SVD has a variable called n\_factors, which is how many abstract features are gonna be in the sub-matrices. In the colab notebook you can see that with the validation data, my RMSE was best when I set n\_factors to 2. With the SVD model, I was getting an RMSE of around .96 on the test dataset.

The second model I used for the simple model was the turicreate model. According to the turicreate API, “Based on simple characteristics of the data, a type of model is selected and trained.” I decided to try the turicreate model as it was very easy to implement and calculate the rmse. With the turicreate model, the rmse using the test dataset was 1.04.

The third and last simple model I made used the matrix\_factorization class. According to the GitHub page of matrix\_factorization, kernel matrix factorization “finds the thin matrices P and Q such that P \* Q^T give a good low-rank approximation to the user-item rating matrix A based on RMSE. This is different from SVD since in SVD there is a constraint for matrices P and Q to have mutually orthogonal columns, but in kernel matrix factorization this is not true.” The matrix\_factorization class has two main variables, epochs, and n\_factors (same variable as for SVD). I set epochs to 75 and n\_factors to 30 as that seemed to be getting the best RMSE with the test dataset and the model was not overfitting (you will see this in the colab notebook). With the matrixx\_factorization model, I was getting an RMSE of around .96 with the test dataset.

For the hybrid model, I decided to use the userID and movie embeddings from the matrix\_factorization class. This is because the class has very easy-to-use functions to get the embeddings. The matrix\_factorization class has a map called user\_id\_map which maps the UserID to the index in the user\_features array which is the user's embedding. The item\_id map does the same thing for the movie IDs - it maps the movie to its features. I also wanted to incorporate more data into the hybrid model. Therefore, I took the genre of the movies (which was already one-hot encoded) and I took some more information on the users (age, gender, profession). Once I got all this data, I passed it into a neural network so that it can learn new features from the data.

To get the data ready for the neural network, I did the following. For every user, I get the movieID and userID, and then with those, I get the index from the map, and from that index, I get the userID and movieID embeddings. For the user info, I get the user ID and match that userID with the right row in the u.user dataset. For the genre, I take the movieID, find the row in the ml\_movies dataset with that movieID, and then take the genres from that row. Within the user data, I take the user’s age (normalized by dividing by the max-age), and I also take a one-hot encoded version of the user’s gender and profession. I believe those are good characteristics to define the user. I think it was important to have characteristics that define the user and not just the user ID as I think this will make better connections between users.

The user embeddings and movie embeddings each have a length of 30. I multiplied the user\_embedding[0] *\* movie\_embedding[0], and do the same for embedding[1] \** movie\_embedding[1], etc up until 30. Then I add the one-hot encoded genres at the end of the array, normalized age, and one-hot encoded gender and profession. In essence, I will have a 2d array like:

[[embeddings of user1[0] *\* embeddings that movie[0], embeddings of user1[1] \** embeddings that movie[1], etc. until embeddings of user1[30] \* embeddings that movie[30], genres of that movie, normalized age of user, one-hot encoded gender of user, one-hot encoded profession of user],

[embeddings of user2[0] *\* embeddings that movie[0], embeddings of user2[1] \** embeddings that movie[1], etc. until embeddings of user2[30] \* embeddings that movie[30], genres of that movie, normalized age of user, one-hot encoded gender of user, one-hot encoded profession of user],

Etc.]

So I ended up with a 2d matrix with features of (80k, 73)

The reason for multiplying the first feature in the movie by the first feature in the user, etc. and keeping each one and not doing the dot product of the two matrices is because each user feature is co-dependent on the movie but if you add it up all together then you are losing some of the data that could be helpful when combined with the genres.

After constructing the 2d array I pass it into a Neural Network with 4 layers, with the input layer having 128 units and the output layer having 1 unit (a rating between 0 and 5). The input shape is 73 as described above. I decided to use relu as the activation for all the layers and have a dropout of .5 between all the layers. I used adam as the optimizer and MSE as the loss. I then used a batch size of 128 and ran the model for 40 epochs (the loss started to stabilize at around 35/40 epochs).

The reason for all these choices is because this led to a model which did not overfit and led to a good RMSE using the training dataset (1.07) and a pretty good RMSE using the testing dataset (1.3). I played around with the neural network configuration a lot and this was the best result I could get.

1. **Project journaling and analysis on implementation:**

I believe the most difficult part of the project was deciding what data to use and once I decided that to clean all the data and combine the different data sets into one dataset which can be fed into the neural network. Once this was done, because python already has many relevant classes for recommender systems, creating the models was not too difficult.

The only metric I used to see how effective my models were was RMSE, which represents the sample standard deviation of the difference between predicted values and observed values. For example, an RMSE of 1 means that on average, the predicted rating deviates from the actual rating by 1 star. Accuracy is not a good measurement as the outcome of the model is a rating between 0-5.

To summarize my results, this is what I got for the RMSE of the different models using the test dataset:

1. Baseline Model: 1.03
2. Simple Model:
3. SVD: 0.96
4. Turicreate: 1.04
5. Matrix\_factorization: 0.96

3. Hybrid Model: 1.3

As you can see, the RSME for the baseline and the three simple models are all below or close to 1, showing that these models are pretty accurate and good models. However, the RMSE for the hybrid model I built is around 1.3, showing that this model is not bad but could definitely use improvement. At first, I was running into the issue that the model was always predicting a rating that is toward the middle point (my min rating was around 3 and my max was around 4 from my predictions). However, I fixed this issue (which you can see in the colab notebook - min is around .5 and max is around 5) and yet the RMSE was still around 1.3. I think if I had more time to improve the Hybrid model, what I would do is try to get the movie embeddings and user embeddings using the other two models (SVD, turicreate) and see what would happen with those embeddings as opposed to the matrix\_factorization embeddings. Furthermore, I would try to use different user features than the ones I used in my model.