Introduction

MUBI is a self-described database and streaming service for auteur cinema. It's currently hand-curated, but as its user base grows, the task of hand curating films for hundreds of thousands of users starts to become tenuous. While hand curation is a feature of MUBI that sets it apart from other streaming services (in addition to its quirky range of films in its library), there might be a way to assist the hand curation.

What if there was an automated recommendation system that could narrow the pool of potential movies the user might want to watch? This way, the system could predict, for instance, 5 movies, and a hand curator could choose to recommend up to 4 of those 5, having more confidence in the recommendation with less upfront work.

Data Collection

Data was from the MUBI user review database on Kaggle:
https://www.kaggle.com/clementmsika/mubi-sqlite-database-for-movie-lovers

Data cleaning and exploration

There were 5 different tables in the dataset with varying amounts of information. I ended up only using the data from the ratings table.

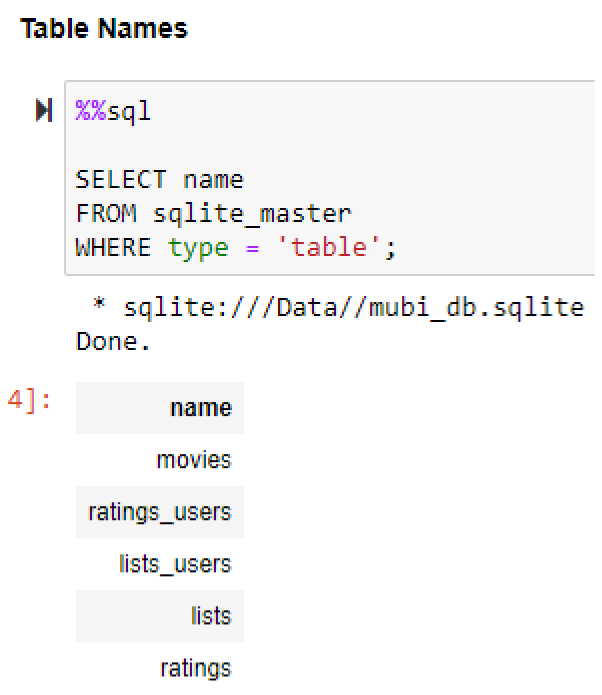


Table Names

```
%%sql

SELECT name
FROM sqlite_master
WHERE type = 'table';
```

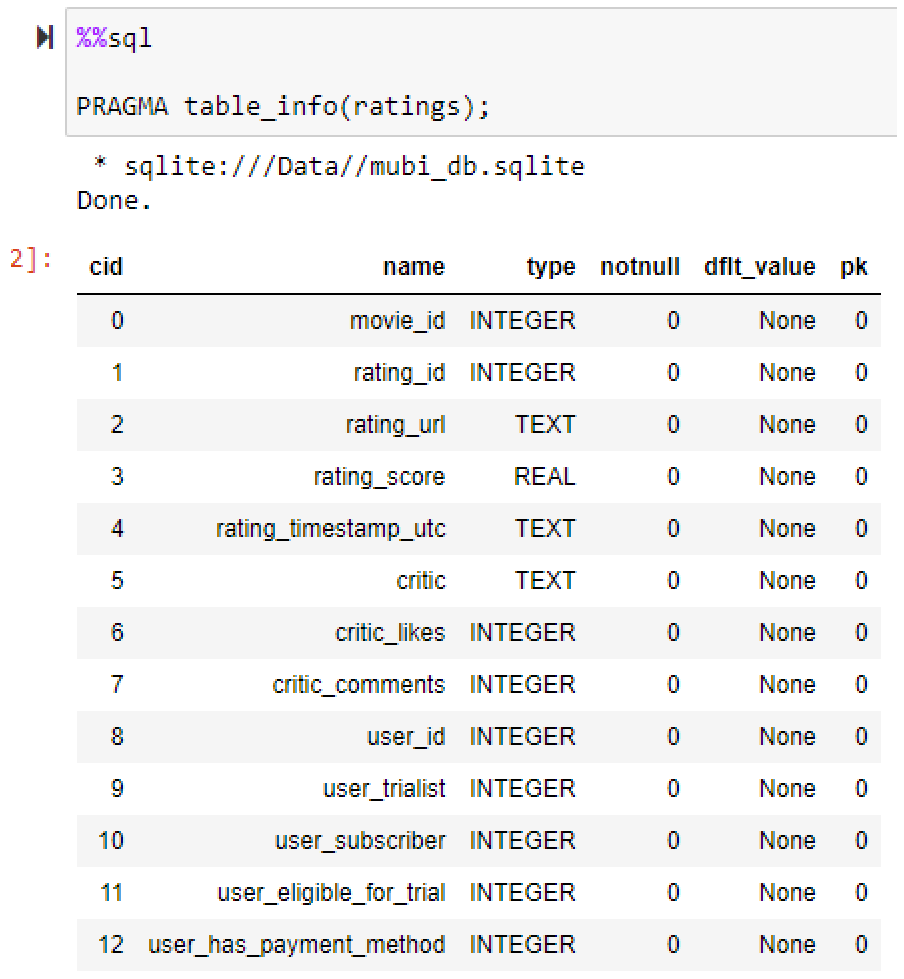
```
 * sqlite:///Data//mubi_db.sqlite
Done.
```

4]:

name
movies
ratings_users
lists_users
lists
ratings

The ratings table had more than just the numeric ratings. Users have the option of also leaving a text review for movies. One future direction for this project is to include more features, either about the movies or the users, into the dataset that the model is trained on.

ratings table



```
%%sql

PRAGMA table_info(ratings);
```

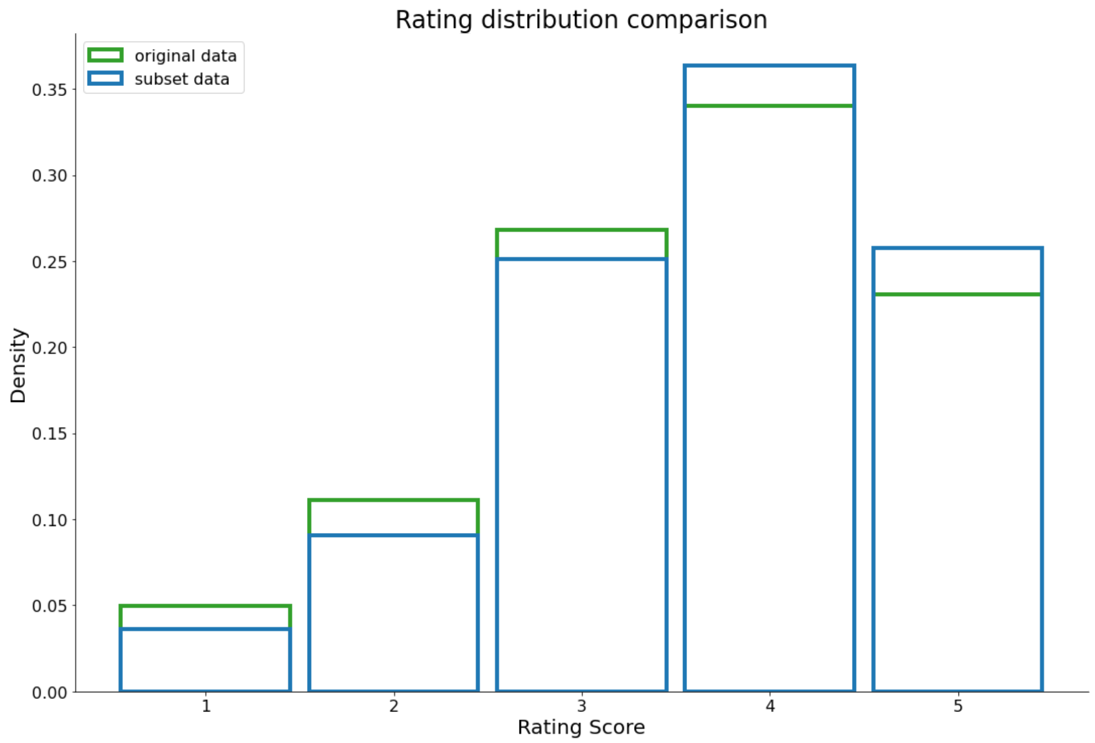
```
 * sqlite:///Data//mubi_db.sqlite
Done.
```

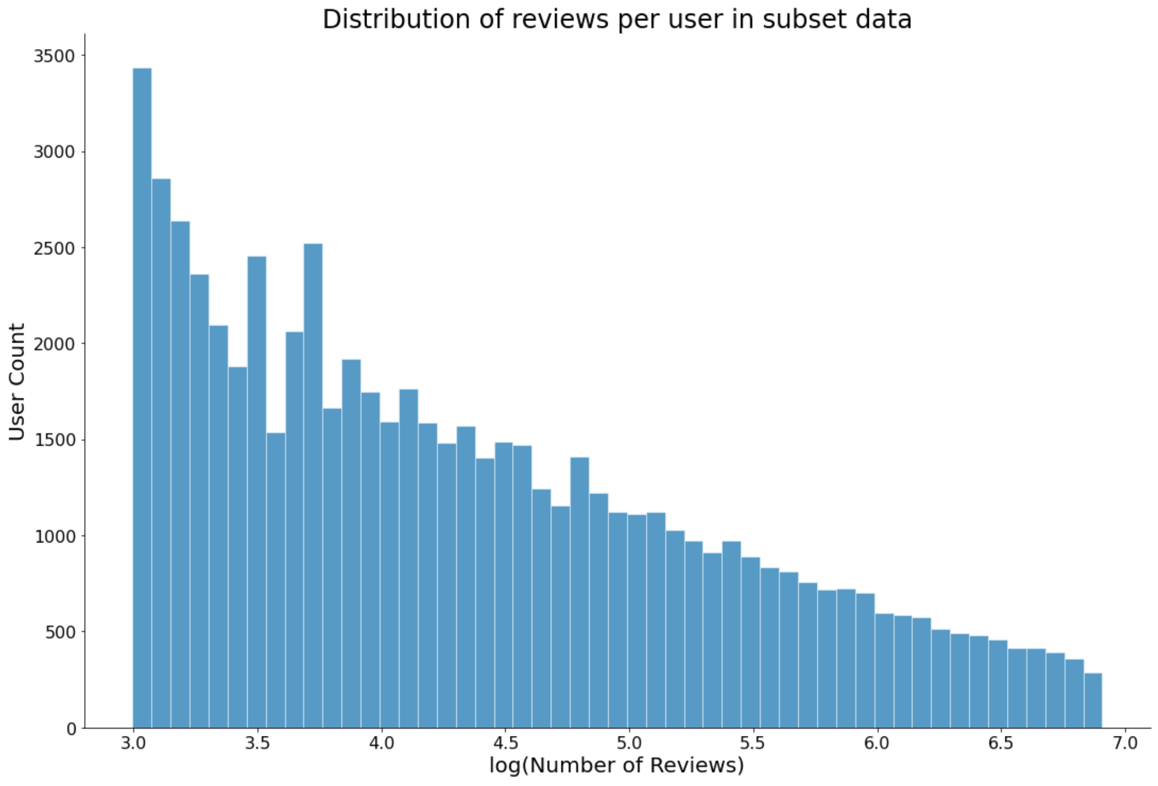
2]:

cid	name	type	notnull	dflt_value	pk
0	movie_id	INTEGER	0	None	0
1	rating_id	INTEGER	0	None	0
2	rating_url	TEXT	0	None	0
3	rating_score	REAL	0	None	0
4	rating_timestamp_utc	TEXT	0	None	0
5	critic	TEXT	0	None	0
6	critic_likes	INTEGER	0	None	0
7	critic_comments	INTEGER	0	None	0
8	user_id	INTEGER	0	None	0
9	user_trialist	INTEGER	0	None	0
10	user_subscriber	INTEGER	0	None	0
11	user_eligible_for_trial	INTEGER	0	None	0
12	user_has_payment_method	INTEGER	0	None	0

The data was already very clean. I accessed it as a SQL database and loaded user IDs, movie IDs, and ratings into a pandas dataframe. There were some null reviews because users have the option of not leaving a numeric rating, but only writing a review. After removing these null entries, I looked at the basic distribution of the data.

In order to make sure each user had enough ratings for the model to learn properly, I only included users with at least 20 ratings. I also excluded users who were outliers on the high end (greater than the 99th percentile of number of reviews).

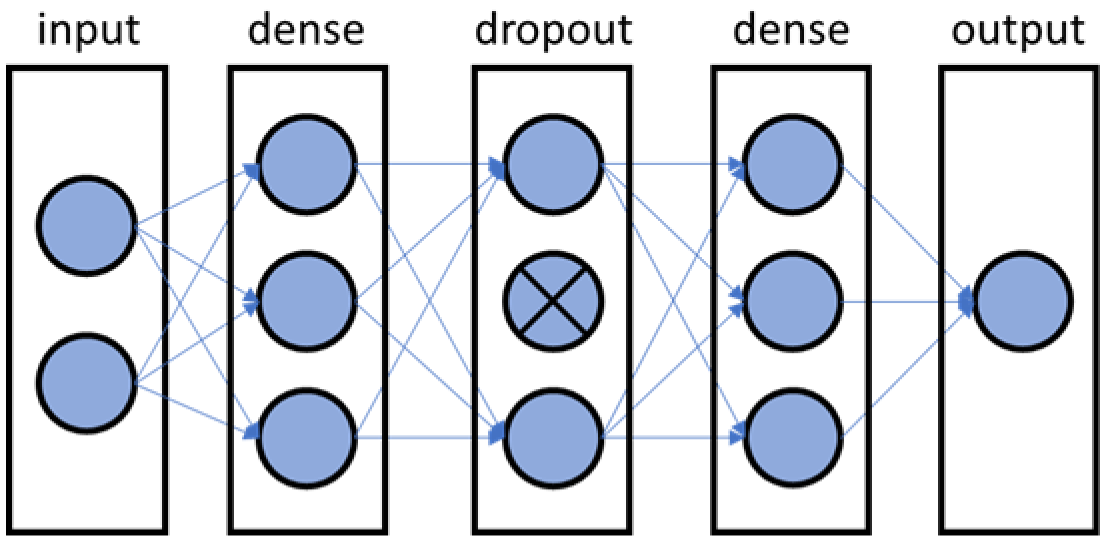

Rating distribution comparison
original data
subset data
Density
Rating Score
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
1
2
3
4
5


Distribution of reviews per user in subset data
User Count
log(Number of Reviews)
0
500
1000
1500
2000
2500
3000
3500
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0

Data pre-processing and Training Data Development

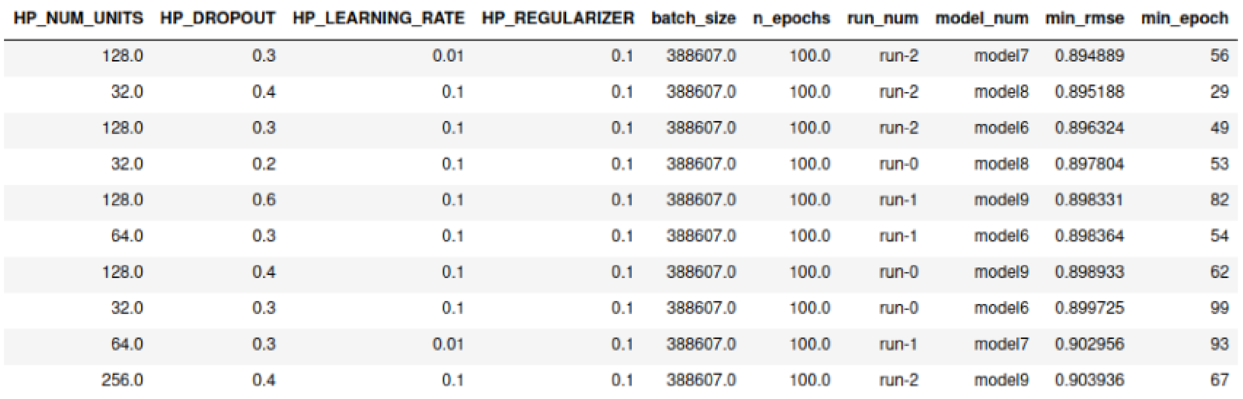
Hyperparameters are the most important part of a neural network. I began by testing various numbers of nodes in dense layers. My initial models had a large overfitting problem, so I introduced a dropout layer and tuned the dropout rate for it. I also tested regularization parameters and learning rate.

The majority of the initial models tested had this basic structure: 2 dense layers separated by a dropout layer.



I used mean squared error as my loss function and root mean squared error as my evaluation metric. Both of these are standard for recommender systems, which need to minimize the distance between predictions and actual values as much as possible.

This table shows the top 10 outcomes of hyperparameter tuning. By the end of my testing, I had run and evaluated over 20 different models using various architectures and hyperparameters.

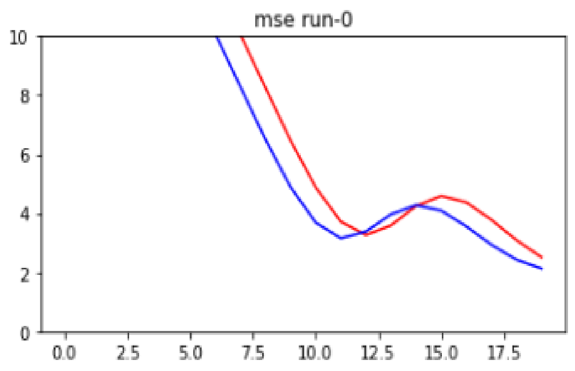


HP_NUM_UNITS	HP_DROPOUT	HP_LEARNING_RATE	HP_REGULARIZER	batch_size	n_epochs	run_num	model_num	min_rmse	min_epoch
128.0	0.3	0.01	0.1	388607.0	100.0	run-2	model7	0.894889	56
32.0	0.4	0.1	0.1	388607.0	100.0	run-2	model8	0.895188	29
128.0	0.3	0.1	0.1	388607.0	100.0	run-2	model6	0.896324	49
32.0	0.2	0.1	0.1	388607.0	100.0	run-0	model8	0.897804	53
128.0	0.6	0.1	0.1	388607.0	100.0	run-1	model9	0.898331	82
64.0	0.3	0.1	0.1	388607.0	100.0	run-1	model6	0.898364	54
128.0	0.4	0.1	0.1	388607.0	100.0	run-0	model9	0.898933	62
32.0	0.3	0.1	0.1	388607.0	100.0	run-0	model6	0.899725	99
64.0	0.3	0.01	0.1	388607.0	100.0	run-1	model7	0.902956	93
256.0	0.4	0.1	0.1	388607.0	100.0	run-2	model9	0.903936	67

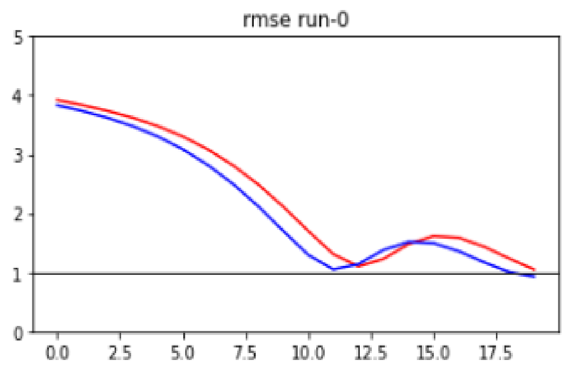
A brief snapshot of my evaluation metrics for the second model I tested versus the final model used to make the predictions.

An early model test:

Loss (MSE)

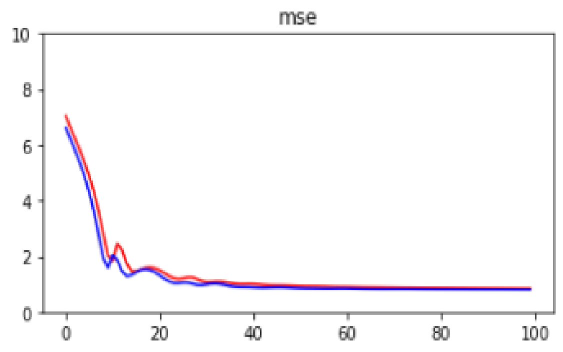


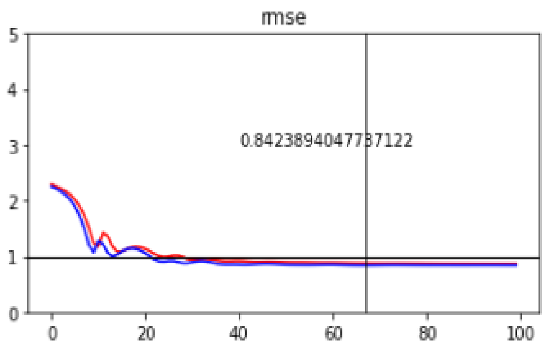
Evaluation (RMSE)



← Epochs →

Final model:



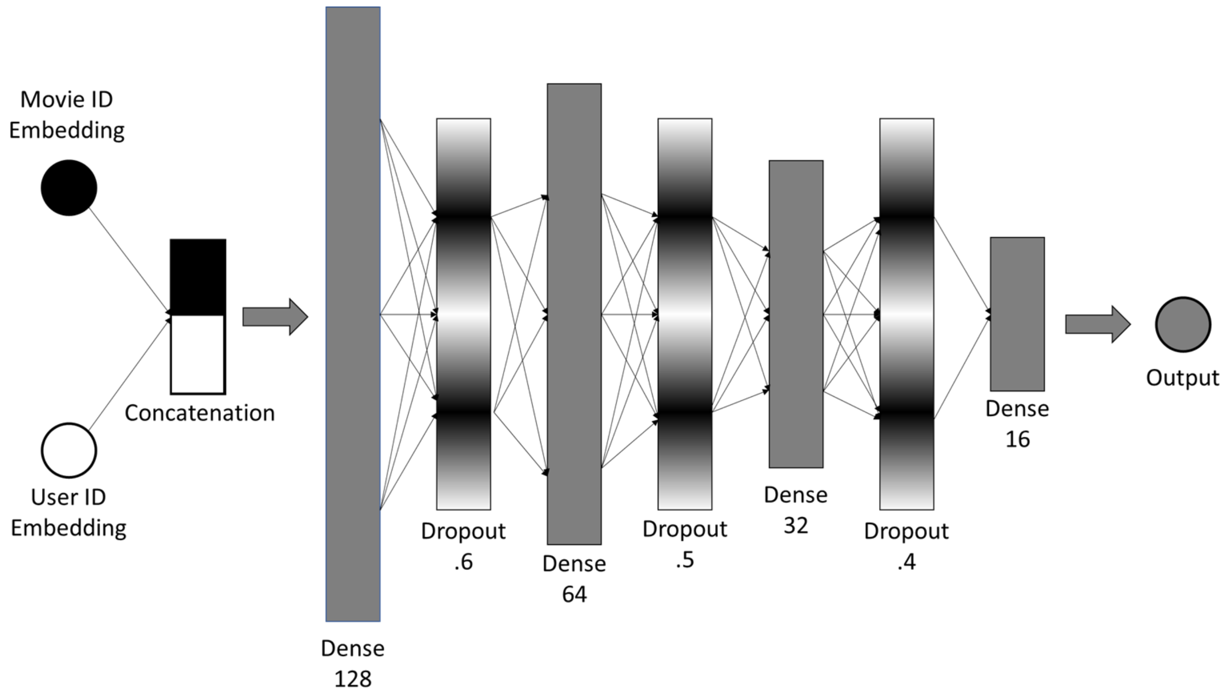


Red: training set

Blue: validation set

Modeling

The final model architecture is shown in the diagram below.

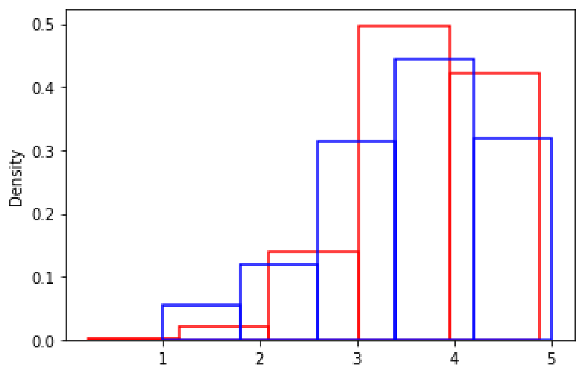


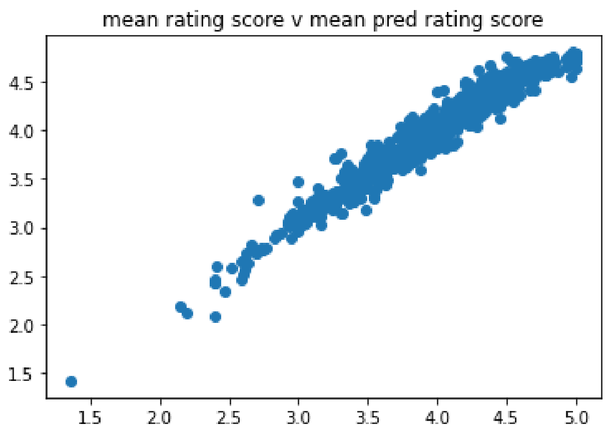
All dense layers used the ReLU activation function and l2 regularization. An initial problem I encountered was a lot of overfitting. Including multiple dropout layers as well as the regularization solved that issue.

Not pictured in the diagram is the initializer bias in the output layer, which I set to the constant 7. An issue I encountered was that the model was minimizing the loss by centering the majority of predictions around the mean of all predictions. By adding bias to the output layer, I was able to shift the predicted values to better reflect the actual distribution of ratings.

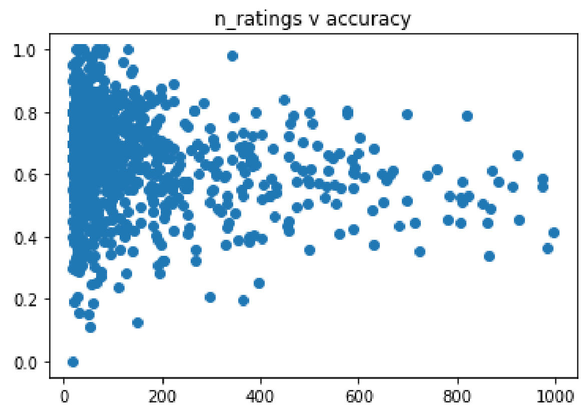
Model evaluation and predictions

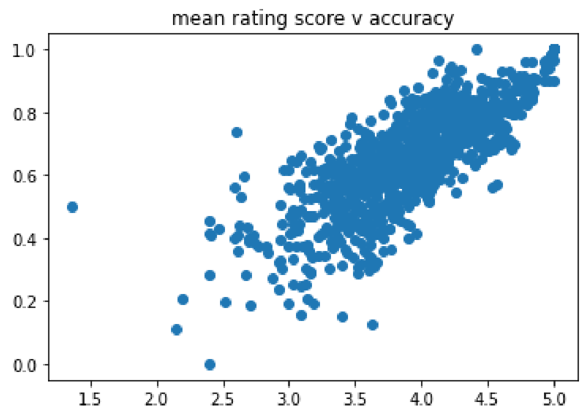
I ran the final model to predict ratings for 1000 randomly picked users for all 80k+ movies in the final dataset. The following plots show some evaluation metrics based on these predictions.
Distribution of actual ratings (red) versus predicted ratings (blue)



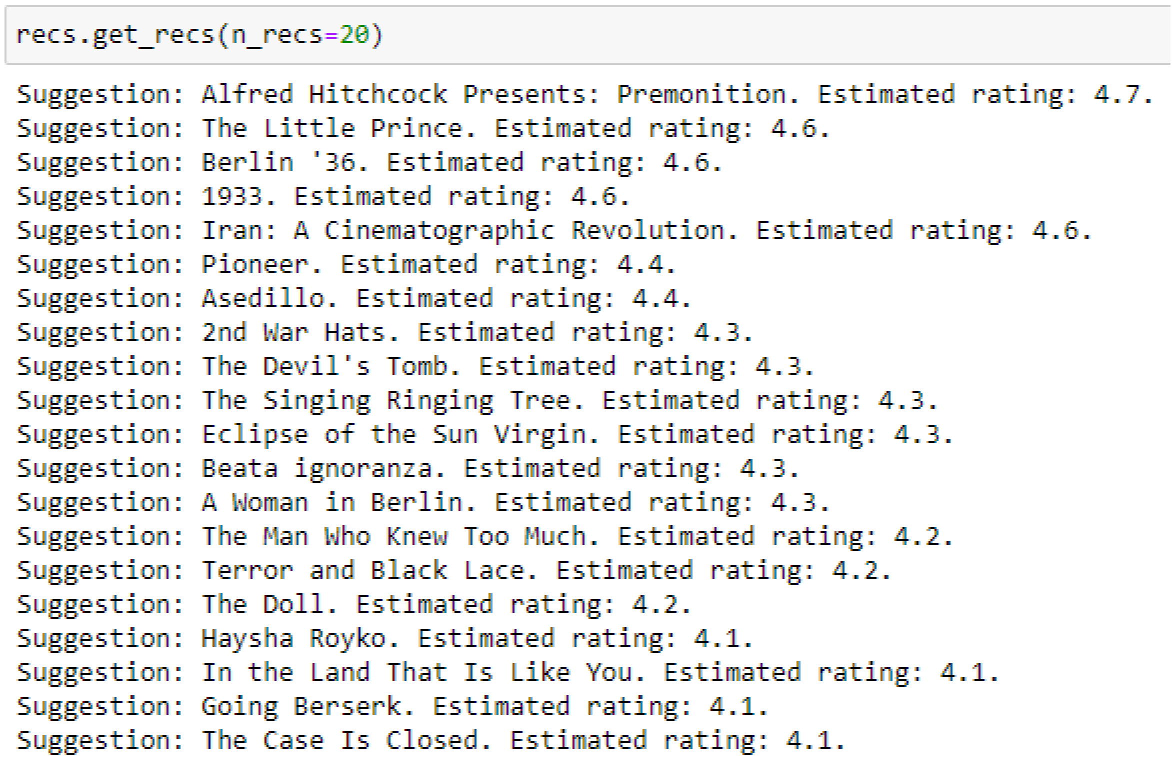


While the number of ratings did not seem to have much of an effect on the accuracy of the model, a user's mean rating did. Part of the reason for this is there are much fewer low ratings than high ratings.In the future it will be good to further train the model with lower ratings.





Finally, I made an entrypoint notebook where people can explore the results of the model and also see predictions for individual users, such as the example below.



```
recs.get_recs(n_recs=20)
```

```
Suggestion: Alfred Hitchcock Presents: Premonition. Estimated rating: 4.7.
Suggestion: The Little Prince. Estimated rating: 4.6.
Suggestion: Berlin '36. Estimated rating: 4.6.
Suggestion: 1933. Estimated rating: 4.6.
Suggestion: Iran: A Cinematographic Revolution. Estimated rating: 4.6.
Suggestion: Pioneer. Estimated rating: 4.4.
Suggestion: Asedillo. Estimated rating: 4.4.
Suggestion: 2nd War Hats. Estimated rating: 4.3.
Suggestion: The Devil's Tomb. Estimated rating: 4.3.
Suggestion: The Singing Ringing Tree. Estimated rating: 4.3.
Suggestion: Eclipse of the Sun Virgin. Estimated rating: 4.3.
Suggestion: Beata ignoranza. Estimated rating: 4.3.
Suggestion: A Woman in Berlin. Estimated rating: 4.3.
Suggestion: The Man Who Knew Too Much. Estimated rating: 4.2.
Suggestion: Terror and Black Lace. Estimated rating: 4.2.
Suggestion: The Doll. Estimated rating: 4.2.
Suggestion: Haysha Royko. Estimated rating: 4.1.
Suggestion: In the Land That Is Like You. Estimated rating: 4.1.
Suggestion: Going Berserk. Estimated rating: 4.1.
Suggestion: The Case Is Closed. Estimated rating: 4.1.
```

Future improvements

To further improve the model, I’d like to incorporate more features beyond just rating scores. These could be facts about the movies (genre, director, etc), facts about the user (what lists they belong to, what paid features they’ve subscribed to), and even facts about ratings (date, did the rating include a text review, how many likes and comments the review got).

I also think data and results such as this would benefit from an interactive dashboard, which could be a future project to allow users to interact with these results.