Top-N Movie Recommendations using Machine Learning

Navya Terapalli [*]

March 22, 2023

Abstract

In this paper we explore recommendation algorithms using machine learning. Specifically, our goal is to predict top-N movie recommendations using different models to give us predicted ratings for a movie. As recommendation research has shown there are several metrics to measure when evaluating top-N recommendations such as accuracy (RMSE/MAE), hit rate, coverage, and diversity. In this research, we are focusing on rating ranking and movie genre coverage. We utilize collaborative filtering, content filtering, hybrid recommenders, and finally include neural nets to generate predictions.

1 Introduction

A strong paper that explores balanced recommendations is titled Calibrated Recommendations published by Netflix [Ste18]. It highlights various calibration metrics, approaches, diversity, and fairness to fight the problem of echo chambers and filter bubbles. However, we tackle the issue of balanced recommendations with a much simpler approach and focus on just two metrics, namely rating values and genre coverage, and evaluate them in a simple manner. We focus on RMSE, the feasibility of using the predicted ratings to rank top-N recommendations, and balancing the percentages of genres for a given model.

We use the MovieLens dataset which contains explicit movie ratings across a number of movies and users. We also query the TMDB API to fetch additional data per movie so that we have a more complete dataset so that we can utilize different approaches.

We have a total of five different models. The first is collaborative filtering based on matrix factorization. The second is content based filtering. The third is a hybrid approach where we combine the first and second. The fourth is collaborative filtering based on matrix factorization using Keras. And the fifth is a neural net enhanced version of the fourth model. Use evaluate and measure our predictions and models with RMSE, cosine similarity, loss/epoch, genre

[*]Advised by: Amandalynne G Paullada

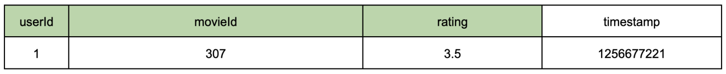
percentage where appropriate. The hypothesis is that hybrid recommendation systems form the most balanced models.

2 Models

In this section we explore our models.

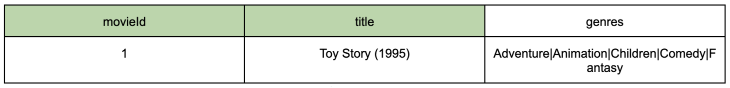
2.1 Collaborative Filtering: Apache PySpark

Using the Apache PySpark library we calculate movie recommendations based on collaborative filtering. We created a subset of the MovieLens 25M dataset to easily work with the data and process it. I chose to use all the 58,000 movies but restricted the model to approximately 1 million ratings given by over 10,000 users. The two main files for my model are ratings-modified.csv and movies.csv. Highlighted below in green are the columns we used for the recommendation algorithm.



userId	movieId	rating	timestamp
1	307	3.5	1256677221

Figure 1: *1 row out of 1M rows in ratings-modified.csv*



movieId	title	genres
1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy

Figure 2: *1 row out of 58K rows in movies.csv*

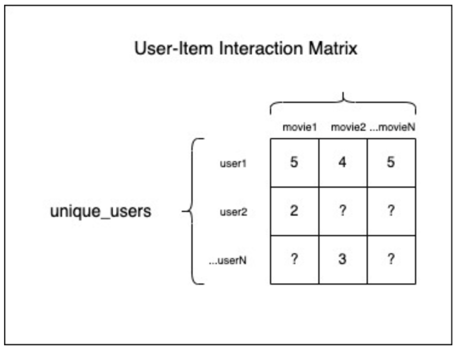


Figure 3: The user–item matrix will be sparse

Each user can give a rating from 1-5 for a given title. The csv files are used to create a user-item interaction matrix like above.

This model's code in this section is based on [M H17] and [Aut]. We utilize the concept of matrix factorization on our ratings matrix. This will decompose the user-item interaction matrix into 2 separate matrices. One is the user matrix and the other is the item matrix. The matrix factorization algorithm we use is ALS, alternating least squares. The most important hyperparameters used are maxIter, rank, and regParam. I've set the maxIter to 10, regParam to 0.01, and rank as the default 1. maxIter iterates the model 10 times during training, the regParam represents the L2 regularization, and the rank represents the latent factors. To test the accuracy of the model we do a 80/20 train-test-split and use a regression evaluator to predict the accuracy of our model. The RMSE (root mean squared error) ranges between 0.84547-0.84921. Which is relatively high given that the ratings range from 1-5. After the model was trained, we combined the data using data structures to easily manipulate the data and present the final recommendation set.

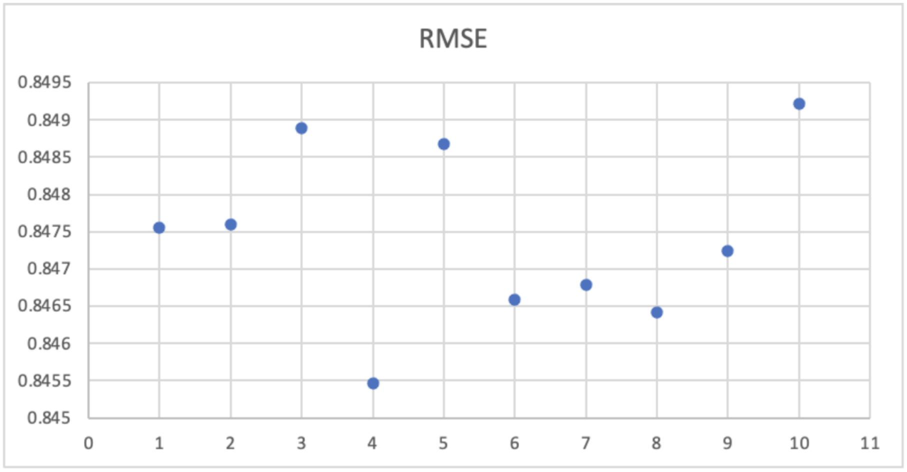
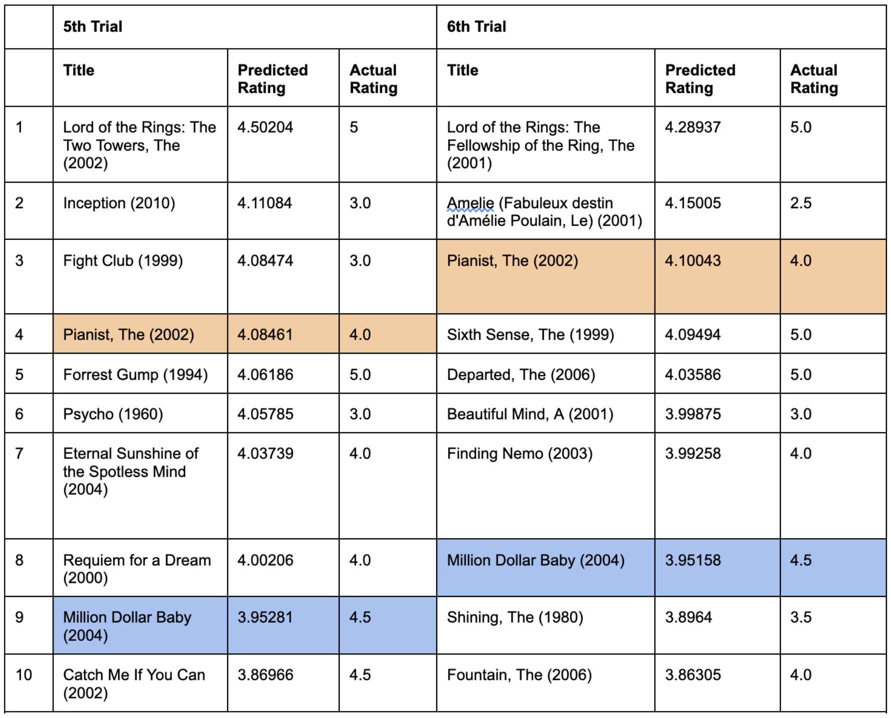


Figure 4: *10 RMSE data points*

One observation of using this ALS model to recommend movies is that the actual movie recommendations are slightly different each time the algorithm is run. Yet, we do find an overlap of recommendations. When the model was run 10 times for a user with userId 99. The following two top-10 recommendations compare the fifth and six trials.



	5th Trial			6th Trial		
	Title	**Predicted Rating**	**Actual Rating**	**Title**	**Predicted Rating**	**Actual Rating**
1	Lord of the Rings: The Two Towers, The (2002)	4.50204	5	Lord of the Rings: The Fellowship of the Ring, The (2001)	4.28937	5.0
2	Inception (2010)	4.11084	3.0	Amelie (Fabuleux destin d'Amélie Poulain, Le) (2001)	4.15005	2.5
3	Fight Club (1999)	4.08474	3.0	Pianist, The (2002)	4.10043	4.0
4	Pianist, The (2002)	4.08461	4.0	Sixth Sense, The (1999)	4.09494	5.0
5	Forrest Gump (1994)	4.06186	5.0	Departed, The (2006)	4.03586	5.0
6	Psycho (1960)	4.05785	3.0	Beautiful Mind, A (2001)	3.99875	3.0
7	Eternal Sunshine of the Spotless Mind (2004)	4.03739	4.0	Finding Nemo (2003)	3.99258	4.0
8	Requiem for a Dream (2000)	4.00206	4.0	Million Dollar Baby (2004)	3.95158	4.5
9	Million Dollar Baby (2004)	3.95281	4.5	Shining, The (1980)	3.8964	3.5
10	Catch Me If You Can (2002)	3.86966	4.5	Fountain, The (2006)	3.86305	4.0

Figure 5: *Two Top-10 recommendations for userId 99 trained on the same model using the same dataset. The left table predicts recommendations when the model has an RMSE of 0.84867. The right table predicts recommendations when the model has an RMSE of 0.84658. Note that the overlapping movies are highlighted.*

We used collaborative filtering to predict the ratings and compare them with the actual score that the user had given the film. Upon inspecting the variance in recommendations due to variance in RMSE. We trained the model ten separate times, each outputting a top-10 for a given user with userId 99, to aggregate a total of 100. Next we used that data to create a histogram to easily visualize how 55 unique movies were recommended- some more frequently than others.

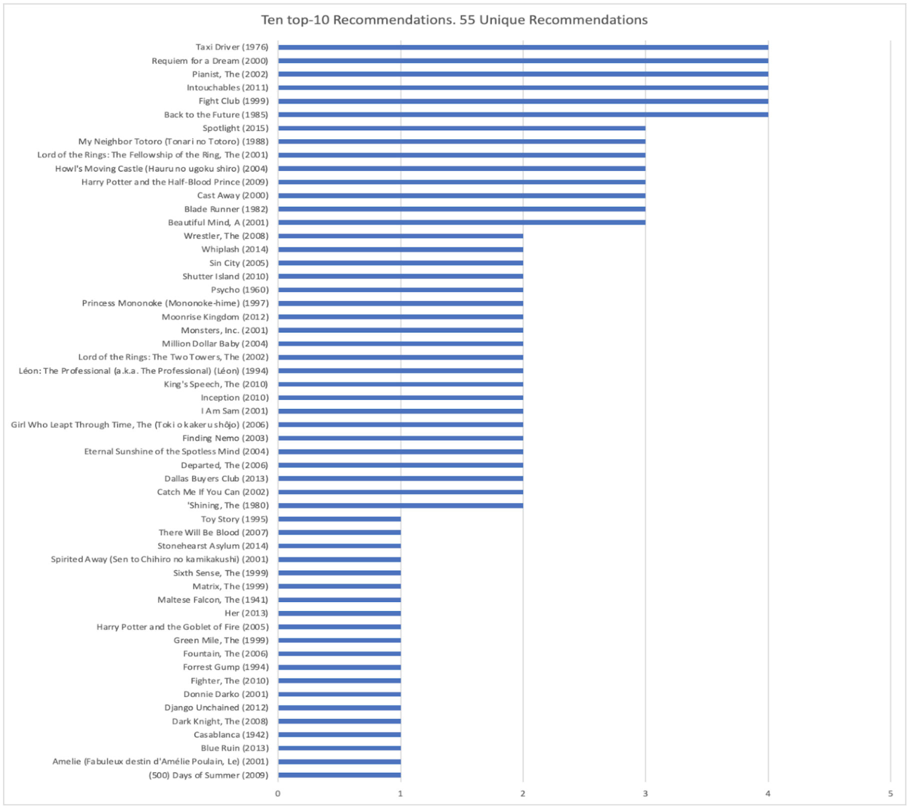


Figure 6: *55 unique recommendations*

The above recommendations span a wide variety of genres such as Drama, Thriller, Crime etc. This model might be suitable for accurately ranking movies by predicted scores and there is an element of randomness that increases the coverage and novelty which shows the users new movies in each set of top-10 recommendations. Let us take a look at how the percentage of genres are shown on a pie chart which evaluates genre coverage. We will aim to improve this distribution in future models as we will see.

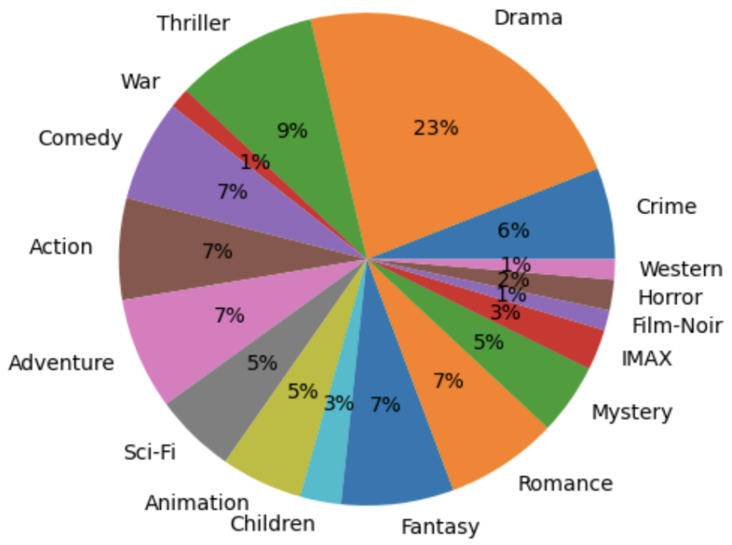
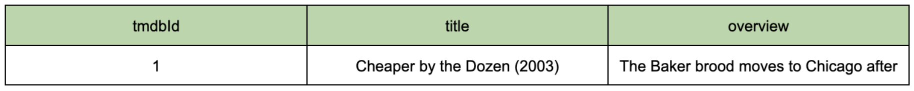


Figure 7: *Each of the 55 recommendations can have 1 or more genres. Pie chart shows the distribution. We see Drama accounts for 23 percent or nearly 1 out of 4*

2.2 Content-Based Recommendation: Scikit-learn

One of the most popular recommendation techniques is content-based filtering. It focuses on item-item similarity based on its properties. We used the sklearn library on the MovieLens dataset and generated our own modified dataset by querying 56,000 movie summaries from TMDB's REST API. After cleaning and processing the data, we had a custom generated csv file. The code used for this content-recommendation model is based on [sen22] and [Gar20].



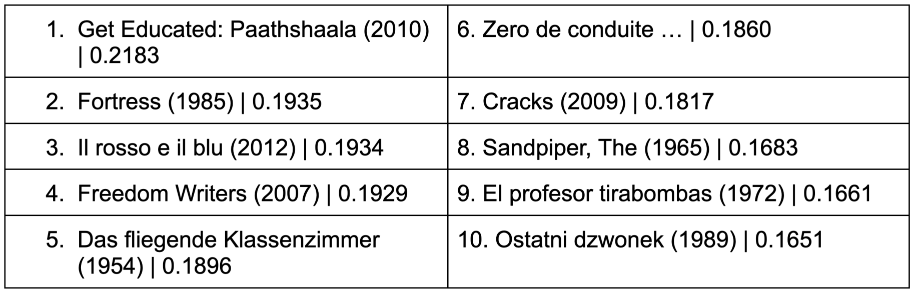
tmdbId	title	overview
1	Cheaper by the Dozen (2003)	The Baker brood moves to Chicago after

Figure 8: *1 row out of 56K rows in custom data*

During the exploratory data analysis we saw that the manually selected 'overview' feature had to be cleaned before being used for the model. The sklean library provides TfidfVectorizer as part of its library which can be used to create a tfidf matrix that can be configured to drop English stop words. This tfidf matrix in turn is used to create a cosine similarity matrix with the help of a linear kernel. The cosine similarity is calculated as follows where each vector (A,B) is the tfidf matrix.

$$\text{cosine similarity} = S_C(A,B) := \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum\limits_{i=1}^{n} A_i B_i}{\sqrt{\sum\limits_{i=1}^{n} A_i^2} \sqrt{\sum\limits_{i=1}^{n} B_i^2}},$$

If the input to our system is the movie ‘Dead Poets Society (1989)’ we can use our cosine similarity matrix to get the top 10 similar movies. Note that although the movies are in a foreign language, the overview is in English.



1. Get Educated: Paathshaala (2010) | 0.2183	6. Zero de conduite … | 0.1860
2. Fortress (1985) | 0.1935	7. Cracks (2009) | 0.1817
3. Il rosso e il blu (2012) | 0.1934	8. Sandpiper, The (1965) | 0.1683
4. Freedom Writers (2007) | 0.1929	9. El profesor tirabombas (1972) | 0.1661
5. Das fliegende Klassenzimmer (1954) | 0.1896	10. Ostatni dzwonek (1989) | 0.1651

Figure 9: *The top 10 similar movies to Dead Poets Society as ranked in descending order by their cosine similarity*

This model is suitable as a general recommender system given a title but due to the static nature of the movie overview the same recommendations will be returned for a given input.

2.3 Creating a Simple Hybrid Recommender: Combining Our Collaborative and Content Approaches

We can combine both of the approaches to create a simple hybrid recommendation system. Our goal is to create more robust recommendations by first selecting a user from our dataset and feeding the corresponding userId into our collaborative recommendation system. This gives up a top-10 recommendation with a predicted rating per movie. Using each of those movies, we can then create a content based recommendation to get similar movies by using the movie plot summary, otherwise labeled the overview. However, our content based recommendations simply give us movie recommendations without a predicted score. So we then take those content based recommendations and feed them back once more into our collaborative system to give us a large set of recommendations each with a given predicted rating. Below is a high level architecture diagram of our hybrid recommendation pipeline. The code used for the hybrid model is a combination of models 2.1 and 2.2 plus our own data processing business logic.

We took the 55 unique movies that I described in the Collaborative Systems and fed it through the pipeline stages 1, 2, 3, 4, and 5 as described above in the picture. The novel set of recommendations gave an output of movie ratings of which I sampled 225 unique ratings and displayed them on the graph below. We can assume that a baseline rating of “3” means the movie is average. By looking at the line of best fit we can see that our hybrid recommendation set has mostly average or above average movies. To be specific, 132 movies with predicted ratings greater or equal to 3 and 93 movies with predicted ratings below 3.

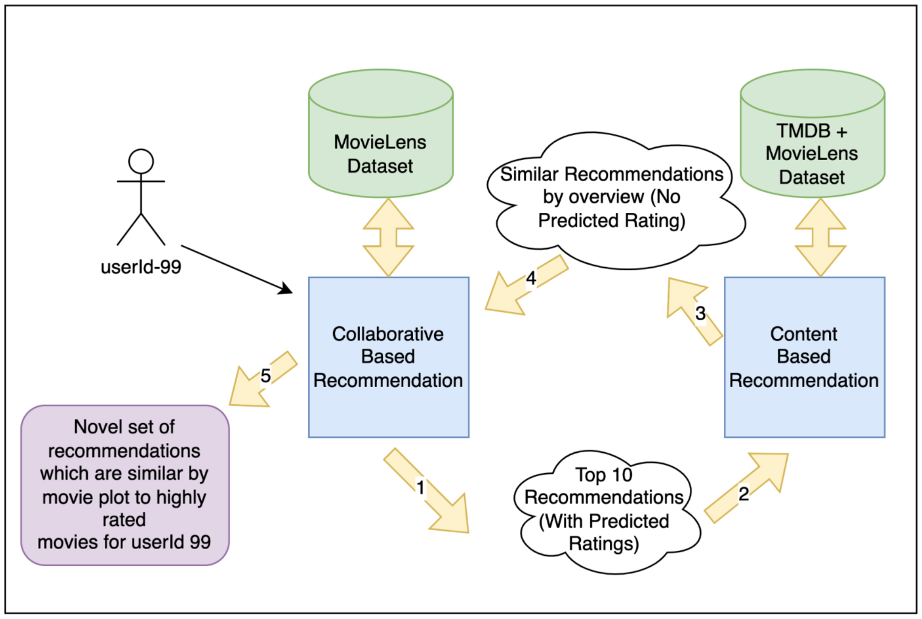

userId-99
MovieLens Dataset
Similar Recommendations by overview (No Predicted Rating)
TMDB + MovieLens Dataset
4
3
Collaborative Based Recommendation
Content Based Recommendation
5
Novel set of recommendations which are similar by movie plot to highly rated movies for userId 99
1
Top 10 Recommendations (With Predicted Ratings)
2

Figure 10: *Hybrid Recommender system takes userId, past userId interactions and outputs a novel set of recommendations*

To balance between the metrics of coverage and a good rated movie we can suggest movies from the above average movie subset of size 132. Given that a movie from the output of our hybrid recommender has a 58 percent of being rated a 3 or above, our system helps us solve the problem of balancing movies that the user will enjoy and improving coverage of a wide variety of movies.

Accuracy with RMSE isn’t the only metric that matters. Other metrics such as coverage are also solved with this. Let’s say a new movie is introduced into our system. Because no one might initially see that movie we won’t have user-item interactions. However, if the overview summary of that new movie is similar to something that is highly rated for a given there is a chance this new movie gets displayed as a novel item to someone. Let us now recreate another

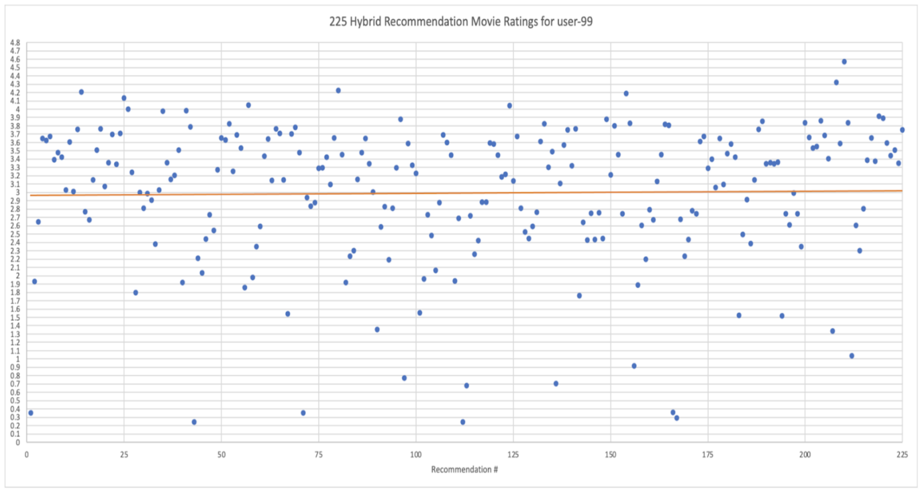
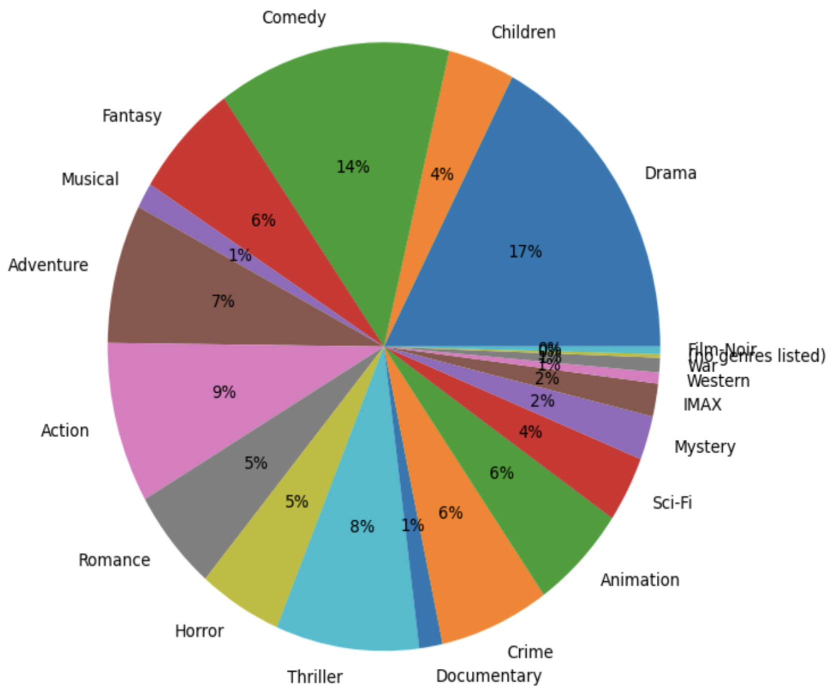


Figure 11: *Graph of 225 movie ratings. 58 percent of recommendations with a rating of 3 or above from our hybrid recommendation model*

pie chart with the 255 movie recommendations from the hybrid approach to see the genre distribution. Although this pie chart has 255 data points, since we are normalizing the data to compare percentages, we can safely say that our hybrid approach provides a more distributed set of genres.

In addition, our simple hybrid approach helps to alleviate the cold start problem. If a new movie is introduced it may be recommended to a user simply because its overview is similar to an existing highly rated movie. This is another reason why it may be okay to recommend movies that are rated below the line of best fit once in a while. Based on the hit rate, the number of movies that a viewer actually selects from the top-10, we can then decide if that recommendation was useful or not. Below we see a more diverse genre representation as compared to Figure 7. Reflecting on our hypothesis, we see that a simple hybrid recommender produces higher genre coverage. Drama genre has reduced below 20 percent and an increase in comedy.



2.4 Keras Collaborative Filtering

We can evolve our recommendation system even further by leveraging the power of Keras. First let's recreate the matrix factorization model we have previously explored using Keras and create a top-n recommendation. The code for this model and the model in section 2.5 comes from [Fen] with some modifications for updated library APIs, minor model architectural changes, and additional code for our recommendations and results analysis. So we reuse the ratings modified.csv we have created earlier to leverage the userId and movieId columns as inputs to our Keras model. First we have to create two Keras Input tensors with a shape of 1 which is what will flow through our model. The shape of 1 lets our model know that we have a one dimensional input with 1 input. This Input is where we supply each column of userId and movieId to each of the tensors. These tensors are then fed into Keras Embedding layers. We try to predict the rating of a user with the concept of the hidden factors that we label as the latent dimension. We model the user-item matrix as the following. We can visualize the input and output dimensions of the Embedding layer. The number of unique users and movies is determined by the size of our dataset. The number of hidden features is configurable and I set it as 20. The number of unique users is the size of the input dimension for the Embedding layer. The number of hidden features is the size of the output dimension for the Embedding layer. Each Embedding layer is then transformed into a Flatten layer which reduces each 2 dimensional matrix with dimensions of (unique users X hidden features) and (hidden features X unique movies) to matrices of size (1 X (unique users X hidden features) and (1 X (hidden features X unique movies).

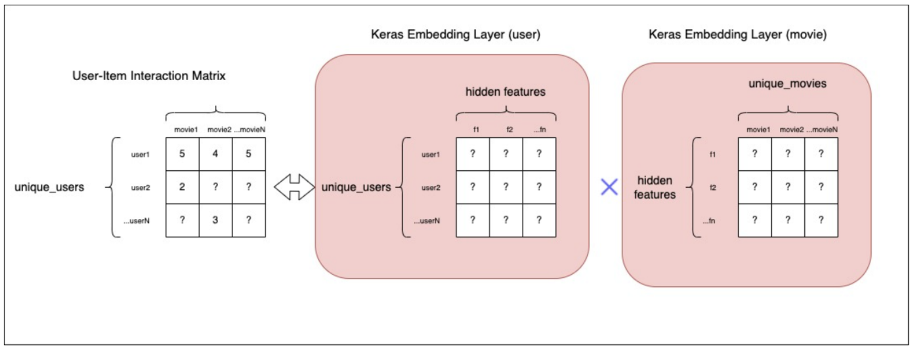


Figure 12: *Both of these Embedding layers will be further processed before taking a dot product to predict a user's rating*

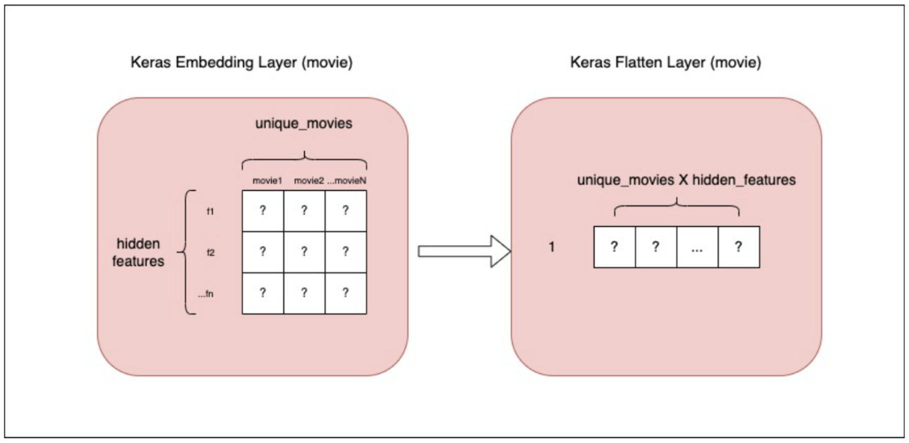
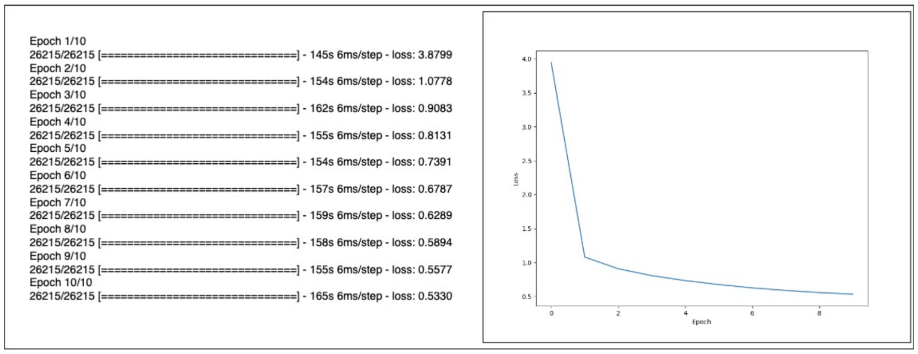

Keras Embedding Layer (movie)
Keras Flatten Layer (movie)
unique_movies
movie1
movie2
...movieN
hidden features
f1
f2
...fn
?
unique_movies X hidden_features
1
...

Figure 13: *The movie Embedding layer is flattened. Similarly the user Embedding layer is also flattened*

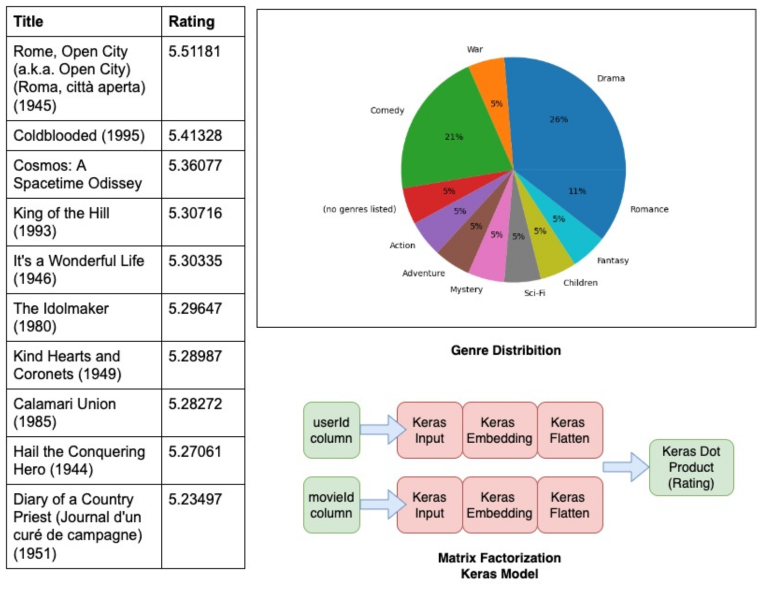
Once we have both the Flatten layers we feed it into a Dot product Keras layer which completes our matrix factorization model. This allows us to predict ratings for a given userId. Let's reuse the same userId 99 to predict top-10 rated movies and train the model for a total of 10 epochs. This time we will predict the rating for userId 99 for every movie in the test set and then get the highest rated top 10. Below we find the architecture of our matrix factorization model and plot the loss over each epoch. Overall we achieve an RMSE of 0.8322399. And the loss decreases sharply between the first and third epoch.



```
Epoch 1/10
26215/26215 [==============================] - 145s 6ms/step - loss: 3.8799
Epoch 2/10
26215/26215 [==============================] - 154s 6ms/step - loss: 1.0778
Epoch 3/10
26215/26215 [==============================] - 162s 6ms/step - loss: 0.9083
Epoch 4/10
26215/26215 [==============================] - 155s 6ms/step - loss: 0.8131
Epoch 5/10
26215/26215 [==============================] - 154s 6ms/step - loss: 0.7391
Epoch 6/10
26215/26215 [==============================] - 157s 6ms/step - loss: 0.6787
Epoch 7/10
26215/26215 [==============================] - 159s 6ms/step - loss: 0.6289
Epoch 8/10
26215/26215 [==============================] - 158s 6ms/step - loss: 0.5894
Epoch 9/10
26215/26215 [==============================] - 155s 6ms/step - loss: 0.5577
Epoch 10/10
26215/26215 [==============================] - 165s 6ms/step - loss: 0.5330
```

Figure 14: *The loss of each epoch for our Keras model*

Next we take the top-N recommendations, check the genre distribution, and display the model. Once the Drama genre seems to be overrepresented. In addition, the top-10 recommendations ratings are slightly higher than the 1-5 ratings, which isn't a good output in terms of accuracy if we want to compare the predictions with the actual ratings a user may give. However, this method still works at least to rank the top recommendations.



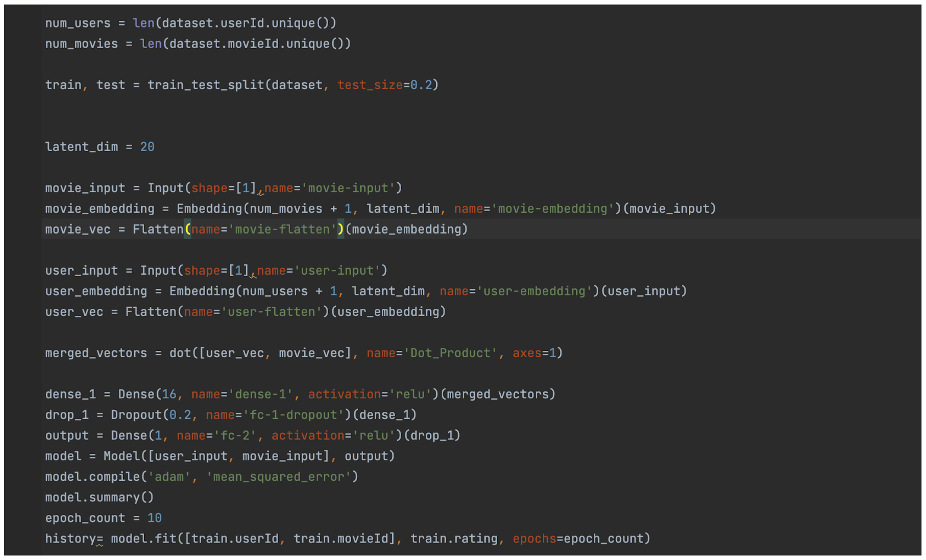
Title	Rating
Rome, Open City (a.k.a. Open City) (Roma, città aperta) (1945)	5.51181
Coldblooded (1995)	5.41328
Cosmos: A Spacetime Odissey	5.36077
King of the Hill (1993)	5.30716
It's a Wonderful Life (1946)	5.30335
The Idolmaker (1980)	5.29647
Kind Hearts and Coronets (1949)	5.28987
Calamari Union (1985)	5.28272
Hail the Conquering Hero (1944)	5.27061
Diary of a Country Priest (Journal d'un curé de campagne) (1951)	5.23497

Figure 15: *The top-10 recommendations (left), genre distribution (top-right), and Keras model architecture (bottom-right)*

Furthermore, the prediction ratings are starkly more precise and close to each other for a given list of movies. This helps to rank movies better that may be more closely rated as opposed to our previous approach using Apache PySpark. Looking at our pie chart we see that our genre distribution is once again concentrated on the drama genre at 26 percent.

2.5 Keras Collaborative Neural Nets

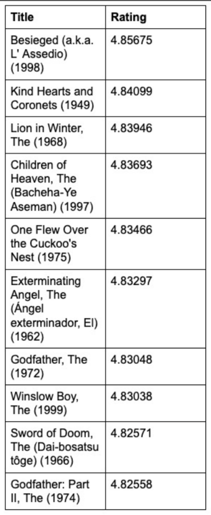
Now let us introduce Keras Dense and Dropout Layers to the previous architecture. My work here is inspired by the paper titled Neural Collaborative Filtering by Xiangnan He et al., where they apply neural networks to collaborative filtering. In their research they conclude that deeper layers offer better recommendation performance [ea17]. My model here in 2.5 is a simple version of that approach which improved rating predictions ranges over the model in 2.4 and is again based on the work from [Fen]. The first Dense layer has 16 neurons using the “ReLu” activation function. Followed by one Dropout layer, and a final Dense layer with 1 neuron.



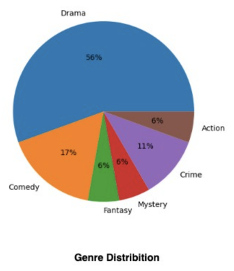
```
num_users = len(dataset.userId.unique())
num_movies = len(dataset.movieId.unique())

train, test = train_test_split(dataset, test_size=0.2)


latent_dim = 20

movie_input = Input(shape=[1],name='movie-input')
movie_embedding = Embedding(num_movies + 1, latent_dim, name='movie-embedding')(movie_input)
movie_vec = Flatten(name='movie-flatten')(movie_embedding)

user_input = Input(shape=[1],name='user-input')
user_embedding = Embedding(num_users + 1, latent_dim, name='user-embedding')(user_input)
user_vec = Flatten(name='user-flatten')(user_embedding)

merged_vectors = dot([user_vec, movie_vec], name='Dot_Product', axes=1)

dense_1 = Dense(16, name='dense-1', activation='relu')(merged_vectors)
drop_1 = Dropout(0.2, name='fc-1-dropout')(dense_1)
output = Dense(1, name='fc-2', activation='relu')(drop_1)
model = Model([user_input, movie_input], output)
model.compile('adam', 'mean_squared_error')
model.summary()
epoch_count = 10
history= model.fit([train.userId, train.movieId], train.rating, epochs=epoch_count)
```

Figure 16: *Python code snippet of building our collaborative neural network recommendation*

Once more we recreate our loss per epoch, top-10 recommendations, genre distribution, and display our Keras model architecture. As we can see the loss reduces much faster than the previous Keras model. The ratings are now bound within the 1-5 rating scale, while still maintaining the precision and closeness of the predicting ratings per movie. Unfortunately, we see that as far as balancing the genres, this model does not present a balanced recommendation. The Drama genre is overrepresented at a whopping 56 percent higher than what we've seen thus far. While this model succeeds at rating predictions, it fails at the second metric of coverage. It achieves an RMSE of 0.82106 and below we see our loss per epoch.



Title	Rating
Besieged (a.k.a. L' Assedio) (1998)	4.85675
Kind Hearts and Coronets (1949)	4.84099
Lion in Winter, The (1968)	4.83946
Children of Heaven, The (Bacheha-Ye Aseman) (1997)	4.83693
One Flew Over the Cuckoo's Nest (1975)	4.83466
Exterminating Angel, The (Ángel exterminador, El) (1962)	4.83297
Godfather, The (1972)	4.83048
Winslow Boy, The (1999)	4.83038
Sword of Doom, The (Dai-bosatsu tôge) (1966)	4.82571
Godfather: Part II, The (1974)	4.82558



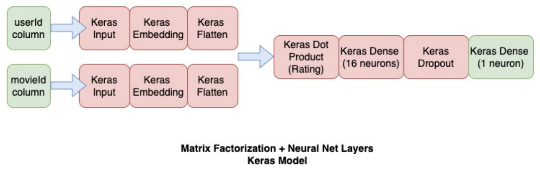
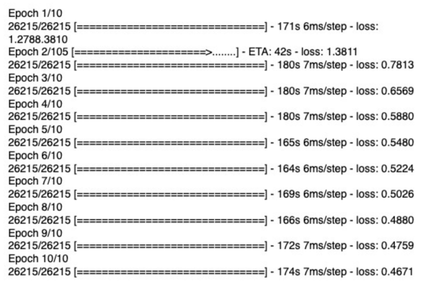


Figure 17: *The top-10 recommendations (left), genre distribution (top-right), and Keras neural net model architecture (bottom-right)*



```
Epoch 1/10
26215/26215 [==============================] - 171s 6ms/step - loss:
1.2788.3810
Epoch 2/105 [====================>........] - ETA: 42s - loss: 1.3811
26215/26215 [==============================] - 180s 7ms/step - loss: 0.7813
Epoch 3/10
26215/26215 [==============================] - 180s 7ms/step - loss: 0.6569
Epoch 4/10
26215/26215 [==============================] - 180s 7ms/step - loss: 0.5880
Epoch 5/10
26215/26215 [==============================] - 165s 6ms/step - loss: 0.5480
Epoch 6/10
26215/26215 [==============================] - 164s 6ms/step - loss: 0.5224
Epoch 7/10
26215/26215 [==============================] - 169s 6ms/step - loss: 0.5026
Epoch 8/10
26215/26215 [==============================] - 166s 6ms/step - loss: 0.4880
Epoch 9/10
26215/26215 [==============================] - 172s 7ms/step - loss: 0.4759
Epoch 10/10
26215/26215 [==============================] - 174s 7ms/step - loss: 0.4671
```

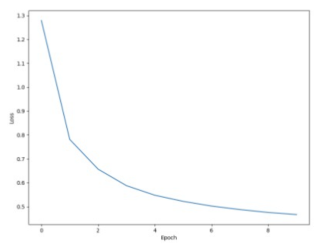


Figure 18: *Loss per epoch for Keras collaborative neural net model*

3 Conclusion and Possible Future Steps

Overall we have enough data to compare our models and see which metrics are useful. As our hypothesis predicted, a hybrid approach balances prediction ratings and genre distribution than either collaborative or content based approaches. Our purely collaborative Keras model was not useful for both prediction rating and genre coverage. Whereas our collaborative neural network Keras model was only useful for prediction rating and was not useful in terms of an equitable genre coverage.

	PySpark Collaborative	**Sci-kit Content**	**Simple Hybrid**	**Keras Collaborative**	**Keras Collaborative + NN**
Prediction Rating	Within scale of 1-5	NA	Within scale of 1-5	Not within scale of 1-5	Within scale of 1-5
Genre Coverage	Genre Concentrated	NA	Genre Diverse	Genre Concentrated	Genre Highly Concentrated

To further continue this research the next logical step would be to utilize metadata that we can gather from the MovieLens and TMDB APIs. In a future research paper, instead of matrix factorization, we could make use of a simple implementation of factorization machines [Lan21].

References

[Aut] No Author. Als - pyspark 3.3.0 documentation. `https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.recommendation.ALS.html`.

[ea17] Xiangnan He et al. Neural collaborative filtering. in proceedings of the 26th international conference on world wide web (www '17). international world wide web conferences steering committee, republic and canton of geneva, che, 173–182., 2017. `https://doi.org/10.1145/3038912.3052569`.

[Fen] C. Feng. Neural collaborative filtering - machine learning notebook. `https://calvinfeng.gitbook.io/machine-learning-notebook/supervised-learning/recommender/neural_collaborative_filtering`.

[Gar20] S. Garodia. Content-based recommender systems in python, 2020. `https://medium.com/analytics-vidhya/content-based-recommender-systems-in-python-2b330e01eb80`.

[Lan21] Fei Lang. Movie recommendation system for educational purposes based on field-aware factorization machine, 2021. `https://doi.org/10.1007/s11036-021-01775-9`.

[M H17] M Hendra Herviawan. Movie recommendation based on alternating least squares (als) with apache spark, 2017. `https://hendra-herviawan.github.io/build-movie-recommendation-with-apache-spark.html`.

[sen22] seniordatascientist. Content-based recommender system with python, 2022. `https://dev.to/seniordatascientist/content-based-recommender-system-with-python-5g85`.

[Ste18] Harald Steck. Calibrated recommendations. *In Proceedings of the 12th ACM Conference on Recommender Systems (RecSys '18). Association for Computing Machinery, New York, NY, USA, 154–162.*, 2018. `https://doi.org/10.1145/3240323.3240372`.

[Ter22] Navya Terapalli. Movie recommendations, 2022. `https://github.com/NavyaTer/Movie-Recommendations`.