Movie Recommendations Time Series

Project Report: Group 7

Group Members

John Rutledge

Brian Gilmore

Chiamaka Aghaizu

William Baker

Introduction

Thanks to technological developments in digital information storage during the information age and the vastness of data available in the era of big data, companies across many industries spend considerable resources trying to improve customer experiences (while also leveraging profit) by utilizing historical user data and machine learning algorithms. The Recommendation algorithms used by Netflix to anticipate movie likes and dislikes are the interest of this research on the effectiveness of various recurrent architectures.

This project aims to build a recommender system using the Netflix Prize dataset and to compare the accuracy of our model to that of a model from 2009 that won $1,000,000 in the Netflix Prize competition [1][2]. The implementation of recommendation algorithms requires historical data across a large number of users in order to accurately predict movie preferences. For example, online retailers like Amazon might use data such as time spent looking at specific products, numbers of clicks on products, and purchase history to recommend similar products to a user that he/she would be more inclined to purchase. Similarly, in order to improve the experience of its customers, Netflix uses a variety of metrics including historical data, movies rated, and movies in queue to recommend other movies and television shows the customer would likely enjoy. The appropriate application of robust recommender systems can increase customer retention, and is arguably the cornerstone of effective entertainment sites.

Related Works

One common approach to recommender systems is the use of matrix factorization. This involves using matrix decomposition algorithms to create a user feature vector and an item feature vector from a user-item matrix. While this method has proven reliable in the past, as it was used by the winners of the Netflix Prize competition, it has several inherent drawbacks - the foremost of these is the size of the matrix itself (and the related overhead, computationally speaking).[1]

Both the Netflix Prize winners and Netflix data scientists have noted the importance of including time when developing a recommender system as it allows a model to account for changes to a user's preferences over time..[1][2] This incorporation of time points to the possibility of using a recurrent neural network, such as an LSTM, to leverage the power of deep learning with time series data.

1 Methods

The modeling process was split in parallel between multiple team members who came up with various different architectures and methodologies. The first hurdle was figuring out how to handle the data. The Netflix Prize dataset consists of approximately 100 million ratings from about 480,000 users on 17,770 movies/shows [3]. This creates an enormous and incredibly sparse user-item matrix with over 8.5 billion cells, but with only 1% of those cells containing non-zero entries. In addition, the process of filling this matrix using user-movie lookup is a single item, non-parallelizable process, leading to impractical processing time.

In order to create a manageable dataset, we reduced the data to the top "u" rating users and "m" rated movies. With u = 10000 and m = 2000, this equated to just under 10% of all ratings (approx. 10 million). Previous work has shown clustering of similar movies using matrix factorization on user-movie matrices.[1] This led us to apply principal component analysis (PCA) and non-negative matrix factorization (NMF) to our top rated user-movie matrix to look for specific clusters. Unfortunately, clusters were only apparent (visually) when using a small subsample of the data (20,000 ratings, Figure 1).

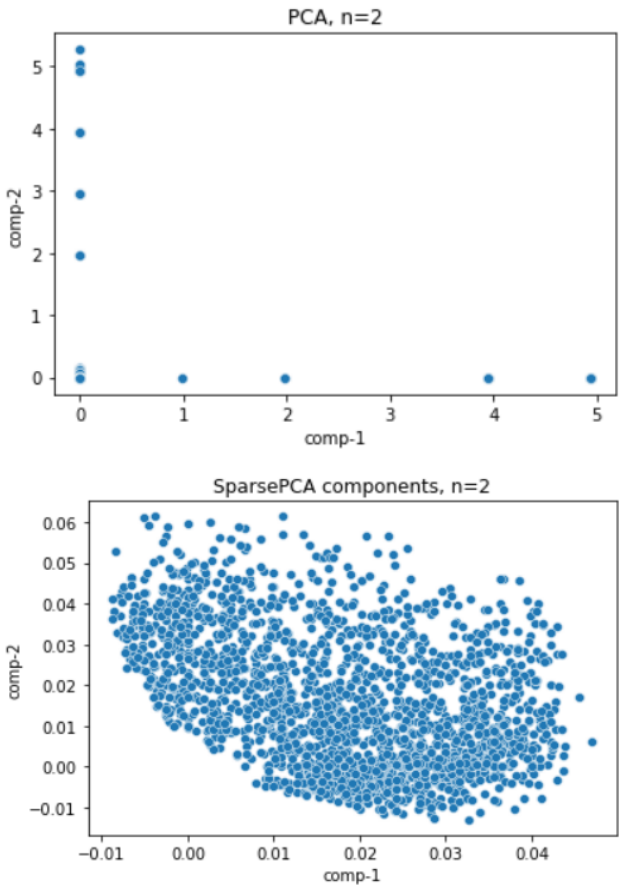


Figure 1: a) PCA on sub-sampled data (n=20,000), b) SparsePCA on top-rated datas

Undeterred (and since K-Means clustering is an incredibly fast computation), we proceeded to compare results of K-Means clustering from 7 clusters to 30, using inertia and silhouette as evaluation metrics (Figure 2). Inertia indicates the coherency of clusters (sum of squared distance of samples to assigned centroid) and the silhouette score is a metric of the similarity of a sample to its assigned cluster (distance to nearest same cluster neighbor vs distance to nearest non-cluster neighbor, averaged across all samples). Based on these metrics and the size of our top-rated dataset, a clustering of 20 was decided as the best selection for dimensionality reduction.

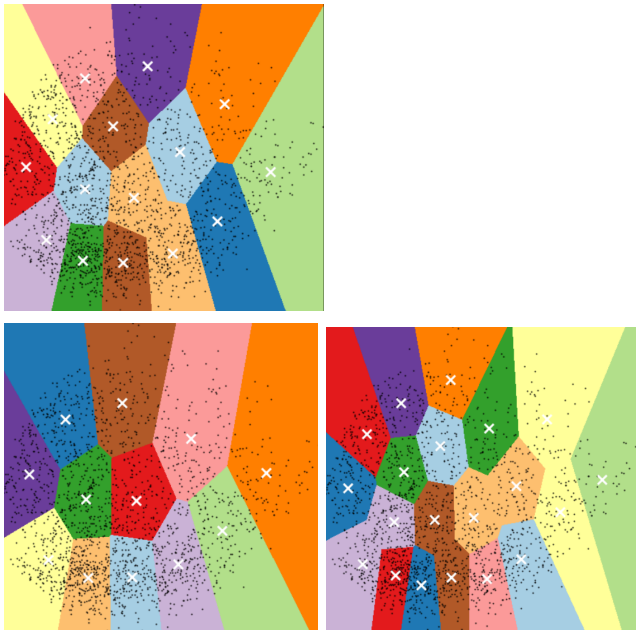
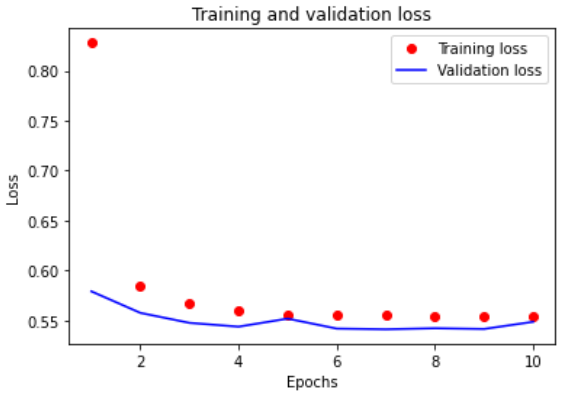


Figure 2: K-Means clustering using 12, 15, and 20 clusters.

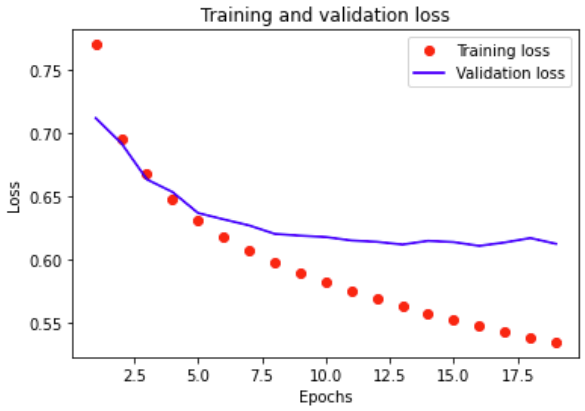
These results were then ordered into a time series with a sliding time window of 10, with the final rating being removed as the label and then split into training, validation, and testing sets. After this, we started with an initial tensorflow recurrent model with 1 layer, 228 LSTM hidden neurons, a normalization layer after the input, and a dense activation using swish, adam optimizer, and early stopping. Finally, this was trained on as many epochs as the validation data kept improving.



In an attempt to improve our model, keras-tuner was then employed using Hyperband tuner and was then used to find an optimized set of hyperparameters for this model. This final best performing model (after one and a half iterations

through the algorithm with 615 configuration trials), produced a model that performed slightly better on the validation dataset. The gelu activation function was notably preferred over sigmoid (with added scale conversion back to the label's 0-5 range), swish, or relu (among others). Gelu uses a standard Gaussian cumulative distribution function to provide a much more linear function than relu, which might explain the preference.

Our second model is a relatively simple tensorflow recurrent model consisting of three dense layers with 256, 128, and 64 nodes with 2 dropout layers positioned between them, followed by a dense output layer. This model used relu activation functions, the adam optimizer, and early stopping based on the validation loss. This model was trained strictly on an embedding of the 'top' users and movies mentioned previously. The image below shows the training and validation loss over time, and demonstrates the importance of validation data when training deep learning models.



After this, all of our models were compared by using the RMSE of the test dataset. This is basically the square root of the loss function: sqrt(mean(square(y_true - y_pred)).

1.1 Results

Due to practical limitations, a direct comparison with the top kaggle performer would not be entirely accurate since our dataset had to be a smaller version taken from the top users and movies, owing to the limitations of our devices and processing times, which for these particular operations do not scale linearly with the data size, but drastically increase in complexity. This is why our team created multiple different models to compare. Additionally, we ran automated tests on the LSTM model architecture to fine-tune elements such as comparing different final activation functions, dropout rates, hidden layers, number of neurons, as well as learning rates. All-in-all, we ran 615 tests, generating over 10 GB of cached model parameters and files before we settled on a promising candidate to add as our third model.

```
Trial 615 Complete [00h 00m 15s]
val_loss: 0.5495085120201111

Best val_loss So Far: 0.541039764881134
Total elapsed time: 00h 02m 43s

Search: Running Trial #616

Value             |Best Value So Far |Hyperparameter
3                 |4                 |layers
0.34145           |0.39063           |dropout_rate
selu              |gelu              |final_activation
0.070598          |0.00097747        |lr
348               |319               |units
19                |7                 |tuner/epochs
0                 |0                 |tuner/initial_epoch
3                 |4                 |tuner/bracket
0                 |0                 |tuner/round
```

Unlike with the Dense model, the LSTM was extremely prone to overfitting; this was to the point where we were unable to create a model with more than one LSTM layer without it performing worse. Here, however, is the example where the Keras-tuner algorithm was able to properly get higher performance from a total of 4 hidden layers, each with 319 units. With our top performing models' RSME test set scores of ~0.737 (Sparse Tensor + Embedding + Dense), the initial LSTM scoring 0.7602, and the keras_tuner model's scoring 0.7602, the deviation of scores was ~2.3%.

Discussion

Comparatively, it took significant processing power and effort to do manual dimension reduction on the movie genres, binning them into 20 categories. However, it also enabled the creation of simplified models requiring far fewer connections (678,874 instead of 1,840,353) and much quicker model prototyping and inference since the work was precomputed. Much of the connections within the dense model were necessitated by the large dimensionality of the movie embeddings. However, compared to our dense embedding model, it performs a bit worse than the more complex and resource demanding model. Thus, to conclude, the manual movie category preprocessing enables the quick and resource-efficient method of handling new movie recommendations, knowing the categories of each (and not the user data on). It reduces the accuracy by a few percentage points compared with embedding. On the

other hand, embedding made out-of-vocabulary movies not usable by the recommendation system and increased resource usage, but it also performed a few percent better and was significantly easier to implement. For future research, the most difficult aspect of this project was the high dimensionality of the data. Since our models reduced the dimensionality (using either embeddings or splitting thousands of movies into 20 categories), getting this section right appears to be the most important component separating the top performers from the lowest performers [1].

REFERENCES

[1] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for Recommender Systems. Computer 42, 8 (2009), 30–37. DOI:http://dx.doi.org/10.1109/mc.2009.263

[2] Harald Steck, Linas Baltrunas, Ehtsham Elahi, Dawen Liang, Yves Raimond, and Justin Basilico. 2021. Deep Learning for Recommender Systems: A Netflix case study. AI Magazine 42, 3 (2021), 7–18. DOI:http://dx.doi.org/10.1609/aimag.v42i3.18140

[3] Netflix. 2019. Netflix prize data. (November 2019). Retrieved May 1, 2022 from https://www.kaggle.com/netflix-inc/netflix-prize-data

[4] Danofer. 2021. Deep Learning for Netflix Prize Challenge. (February 2021). Retrieved May 1, 2022 from https://www.kaggle.com/code/danofer/deep-learning-for-netflix-prize-challenge/notebook

[5] Anon. sklearn.metrics.mean_squared_error. Retrieved May 1, 2022 from https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html

[6] Anon. 2022. Netflix prize. (April 2022). Retrieved May 1, 2022 from https://en.wikipedia.org/wiki/Netflix_Prize