Matrix factorization for Recommender Systems

Supervisor:
Dr. Tsui-Wei Weng

Posted:
Dec 6, 2022

Course:
DSC 210 FA'22 Numerical Linear Algebra

Team:

- Suhas Hebbur Eshwar
  PID: A59017426
  CSE
- Gagan Gopalaiah
  PID: A59013265
  CSE
- Dhruthick Mohan
  PID: A59012297
  CSE

1. Introduction

Recommender Systems can be described as algorithms which suggest relevant items to users. At their core, recommender systems are concerned with understanding interactions between users and items. Recommender systems operate by finding common patterns and relationships among users and items, so

that recommendations for a user can be harvested from others who have similar interaction patterns

Recommender systems also help in extracting limited number of items from a huge corpus that best match the user's need. This is especially advantageous if there is an overwhelming amount of data and only a minute subset of data can be displayed to the user. For example, in case of YouTube (a popular video consumption platform), it has a catalog of about 800 million videos and when a user opens YouTube, only a handful of videos can be recommended.

Music subscription services use recommendation systems to shape individual tracks into comprehensive playlists by a particular criterion. By learning more about the nature of playlists, better relevant tracks can be suggested to the user thereby improving the overall experience on music streaming applications.

1.1 History/Background:

Playlists are at the core of music streaming apps such as Spotify, YouTube or Pandora. Playlists like Today's Top Hits and RapCaviar have millions of loyal followers, while Discover Weekly and Daily Mix are just a couple of personalized playlists made specially to match user's unique musical tastes.

Users love playlists too. In fact, the Digital Music Alliance [1], in their 2018 Annual Music Report [2], state that 54% of Spotify's consumers say that playlists are replacing albums in their listening habits. Users not only love playlists, but also love creating them.

The Spotify Million Playlist Dataset Challenge consists of a dataset and evaluation to enable research in music recommendations. It is a continuation of the RecSys Challenge 2018[4], which ran from January to July 2018. To date, over 4 billion playlists have been created and shared by Spotify users. People create playlists for all sorts of reasons: some playlists group together music categorically (e.g., by genre, artist, year, or city), by mood, theme, or occasion (e.g., romantic, sad), or for a particular purpose (e.g., focus, workout) [3].

1.2 Applications:

With the development of recommendation algorithms, more and more practical recommender systems have been developed that provides personalized recommendations to both users and groups. From streaming services to advertisement, they are widespread and essential in many domains.

The importance of good recommendations cannot be overstated since it is pivotal in deciding a company's success or failure. This has been the story of TikTok, which is a short form video hosting service. Despite well-established social media giants, TikTok has had an exponential growth. This success has been attributed to its state-of-the-art recommendation system which has resulted in a phenomenal user engagement of over 1 billion monthly active users.[5]

Netflix also uses a state-of-the-art recommendation system that creates a personalized recommendation to each user. It considers various factors such as interactions of the user with the service (such as viewing history or how the user rated the titles), other users who may have similar preferences in movies. Some other things that are captured to make the recommendations are time of the day, the device being used, how long user is watching, etc.

1.3 State of the art:

Latent factor models are one of the state-of-the art models that are widely in use today. The model incorporates matrix factorization along with bias terms that help make better predictions. Deep Learning based recommenders are also popular as they are capable of learning nonlinear relationships present in the data. Outside of these, other popular approaches being used in recommendation engines today are factorization machines and auto encoders. In our work we will experiment with latent factor models and one popular deep learning approach - neural collaborative filtering.

---

2. Problem Formulation

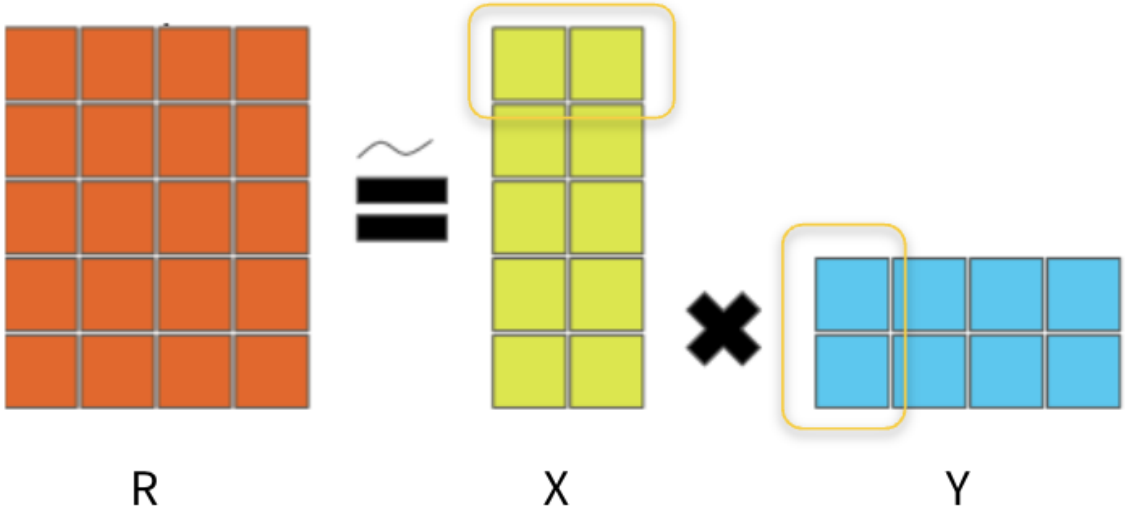
Problem Statement - Automatic Playlist Continuation using Matrix Factorization

Given a seed playlist title and/or initial set of tracks in a playlist, predict the subsequent tracks in that playlist. Matrix factorization is an integral part of the collaborative-filtering based recommender systems.

We attempt the solve the above problem with Matrix Factorization using 2 approaches -

- Numerical Linear Algebra method using Singular Value Decomposition (SVD)
- Neural Collaborative Filtering (SOTA)

A popular approach to build collaborative-filtering based recommender systems is latent factor models. In these models, instead of defining the features explicitly, we let the model implicitly extract the important user and item features. The basic assumption is that there exists an unknown low-dimensional representation of users and items. Thus, the features are ‘latent’, or implicit. Put differently, latent factor models-based recommender systems are essentially a form of dimensionality reduction based on the principles of Matrix Factorization.



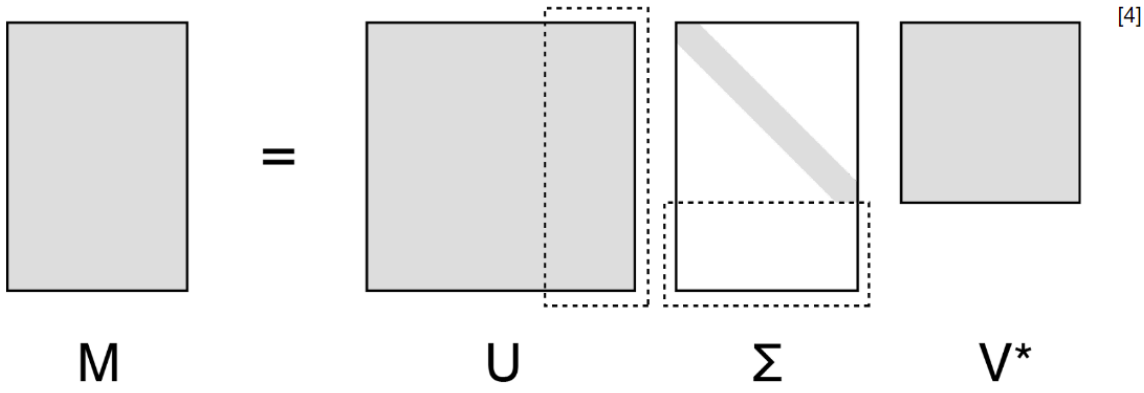
Section 2.1 below describes a linear algebraic approach to factorize matrices.

2.1 Relation to Numerical Linear Algebra:

An intuitive approach to Matrix Factorization is to use SVD or Singular Value Decomposition which decomposes a matrix M into two rotational matrices U and $V^*$and one diagonal matrix composed of singular values.

2.2 NLA Approach Description:

Singular Value Decomposition of a matrix M can be expressed as -



[4]

$$M = U\Sigma V^*$$

where,

- U and V^* are left and right singular values of M (eigenvectors of MM^T and M^TM), and
- Σ is a diagonal matrix of eigenvectors of MM^T

Critically, the best possible rank K approximation of M (in terms of MSE) is found by taking the top K eigenvectors/eigenvalues in $U, \Sigma \; and \; V$ (Eckhart Young theorem).

While SVD appears to be good choice for finding the best possible latent features, it is limited to fully observed matrices, which are extremely rare in datasets used for recommender systems.

Advantages of SVD -

- Efficient in time
- Computationally less expensive
- Simple and intuitive to understand

Disadvantages of SVD -

- Performs badly on sparse data.
- Is not capable of incorporating metadata

2.3 SOTA Approach Description:

Today, in practice, gradient descent is used to choose the latent user and item factors instead of Singular Value Decomposition. This works by minimizing a loss function that accounts for the mean squared error between the predictions and the actual rating. A learning rate is set and at each iteration γ_u and γ_i are adjusted such

that the resulting loss takes a step closer to its minima. By doing so, γ_u and γ_i that best fit the user-item interaction data are calculated.

$$Loss = arg\ min \frac{1}{|R|} \sum_{(u,i) \in R} (\gamma_u . \gamma_i - R_{u,i})^2$$

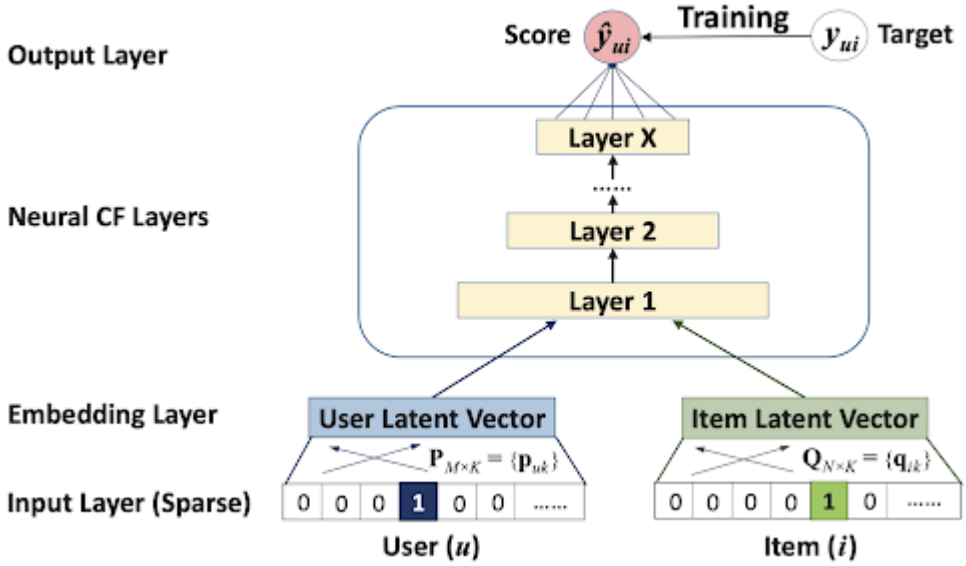
The interaction function, $f(u,i) = \gamma_u . \gamma_i$, can be further modified by incorporating user and item bias terms (β_u, β_i) along with an offset term, α[5].

$$f(u,i) = \alpha + \beta_u + \beta_i + \gamma_u . \gamma_i$$

This form of interaction function is widely used in latent factor based models today as it gives a better performance when regularization is involved to prevent overfitting.

Among other popular methods of implementing recommender systems are deep learning approaches. Increasingly, state-of-the-art recommendation models are based on deep learning methods. These methods are preferred as they are capable of learning complex, non-linear relationships between user and item representations. Deep Learning methods such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) can also incorporate auxiliary information in the form structured data (images, sequences, text, etc.) that can help improve recommendation.

When it comes to matrix factorization for collaborative filtering, one well-known deep learning approach is to replace the inner product between the latent factors (γ_u and γ_i) with a neural architecture which can be as simple as a multi-layer perceptron (MLP). This approach is called Neural Collaborative Filtering (NCF)[6]. The basic structure of this model is depicted below.



The latent vectors are combined by multiple layers of the network to produce the output. The vectors are learned as embeddings for each user and for each item. Both, the MLP's weights and the latent factor are learned simultaneously through back propagation. Another advantage of using neural architectures such as these is that features from metadata can be easily incorporated into the model. This is not possible with SVD models.

2.4 Evaluation:

We plan to use the following 2 metrics for evaluating our results -

1. **Hit Ratio @ K**

   Fraction of playlists for which the correct track is present in the top K recommendations of the playlist. It is defined by the formula below

$$HR = \frac{|U_{hit}^{K}|}{|U_{all}|}$$

2. **NDCG @ K**

   NDCG which stands for Normalized Discounted Cumulative Gain is a measure of ranking quality.

   We need to understand a few other terms before we can formally define NDCG.

   Gain: Gain is the relevance score for each item recommended.

Cumulative gain (CG): Cumulative gain at K is the sum of gains of the first K items recommended.

$$CG_{@K} = \sum_{i=1}^{K} G_i$$

Cumulative gain doesn't take into account the ordering of the items.

Discounted Cumulative Gain (DCG): DCG takes into account the ordering of items be penalizing the highly relevant items placed at the bottom.

$$DCG_{@K} = \sum_{i=1}^{K} \frac{G_i}{log_2(i+1)}$$

There is one disadvantage with DCG where it is dependent on the number of items being recommended and is not a good metric to compare models that are recommending different number of items.

Normalized Discounted Cumulative Gain (NDCG):

The normalized discounted cumulative gain is the DCG with a normalization factor in the denominator.

The denominator is the ideal DCG score when we recommend the most relevant items first.

$$NDCG_{@K} = \frac{DCG_{@K}}{IDCG_{@K}}$$

$$IDCG_{@K} = \sum_{i=1}^{K^{ideal}} \frac{G_i^{ideal}}{log_2(i+1)}$$

---

3. Experiments

3.1 Setup and logistics:

Programming Language - Python

Data Visualization and Exploratory Data Analysis - matplotlib

Vector-Matrix operations and Sparse matrix representations - NumPy

Model implementations - Surprise and TensorFlow

Logistic Tools - GitHub, Google Collaboratory, Notion

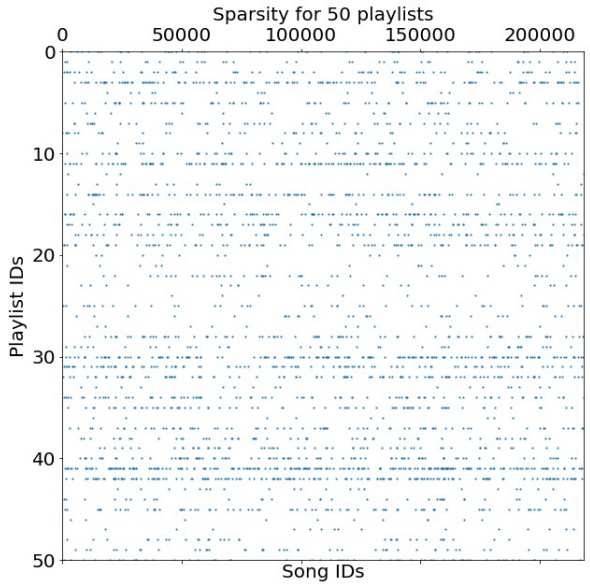
3.2 Dataset and Preprocessing:

We used the Spotify Million Playlist Dataset (SMPD) which has 1,000,000 playlists, including playlist titles and track titles (over 2 million unique tracks by nearly 300,000 artists). Each interaction can be defined as (playlist *p*, track *t*), where *t* is a track present in playlist *p.*

After fetching the interactions between playlists and tracks, we had approximately 66 million interactions. Due to lack of computation power, we decided to work on 2% of the dataset (approximately 1.3 million interactions). Some statistics for the chosen subset of data is described below.

Number of tracks	262338
Number of playlists	19904
Density (Number of non-zero interactions / Number of all possible interactions)	0.003
Sparsity	0.997

We also observed that the dataset is very sparse as depicted by the plot below.



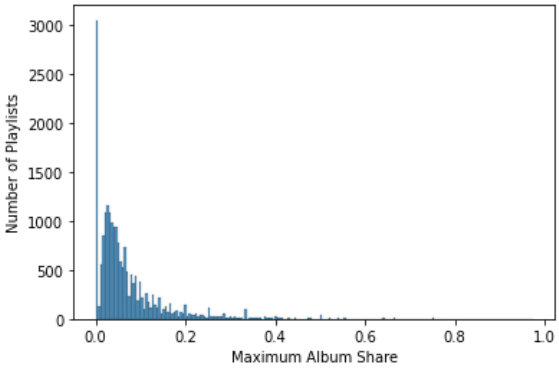
Visualizing Sparsity in the Dataset

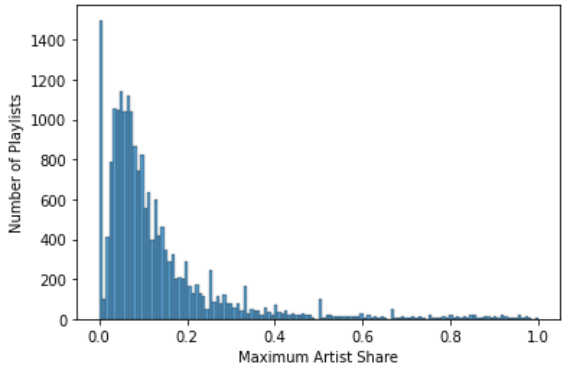
This dataset was then split into train, validation and test sets in the ratio 8:1:1. After which for each interaction in the train set, we sampled a negative interaction by pairing the playlist with a randomly sampled track that was not already in the playlist.

We also extracted some more features from the metadata available on the tracks.

- Artist share: For a given interaction (playlist *p*, track *t*), it is the fraction of tracks in *p* which are by the same artist as *t*. Intuitively, a given track is more likely to be added to a playlist if the playlist already contains tracks by the artist behind the given track.
- Album share: For a given interaction (playlist *p*, track *t*), it is the fraction of tracks in *p* which are in the same album as *t*. Again intuitively, a given track is more likely to be added to a playlist if the playlist already contains tracks from the same album as the given track.

The following plots visualize a histogram for the maximum artist share and album share across playlists. Since the non-zero values of these features are quite normally distributed, they can be used to learn a better model. Our experiment with Neural Collaborative Filtering uses these features as explained in the later section.





3.3. Implementation of NLA Approach

The following steps were followed to implement SVD using Surprise library -

1. Initialize Reader object and load data from the dataframe.
2. Build the train set using build_full_trainset() function.
3. Initialize SVD with required parameters. n_factors defines the number of latent features per playlist and per track. biased determines whether model has to use playlist and track bias terms or not.

4. Fit the model on the train set.
5. Evaluate the performance on test set.

```
# Pre-process and load the data
reader = Reader(rating_scale=(0, 1))
dataSVD = Dataset.load_from_df(train_wnegatives[['pid','tid','label']].sort_values(by
='pid'), reader)

# Build train dataset
trainSVD = dataSVD.build_full_trainset()

# Initialize SVD model
svd_model = SVD(n_factors=4, n_epochs=10, biased=False)
svd_model.fit(trainSVD)

# Evalute performance of SVD model
svd_hrs_dict,svd_ndcgs_dict=evaluate_SVD_model(svd_model,valid_wnegatives)
```

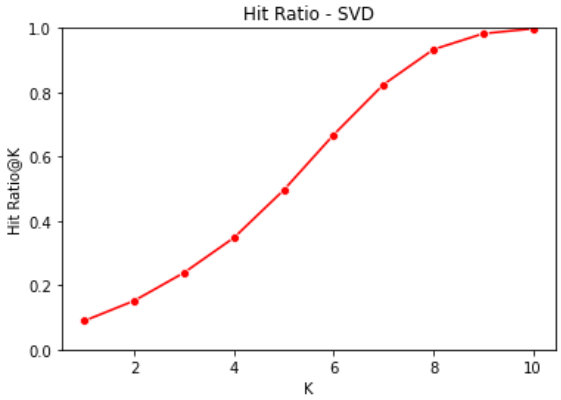
3.4 Results of NLA approach:

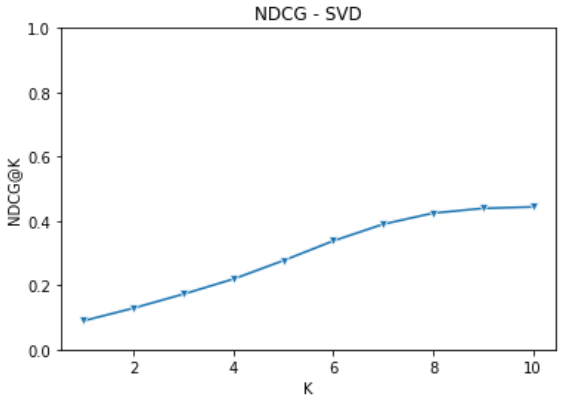
The evaluation metrics, Hit Ratio and NDCG, are implemented as shown:

```
# Hit Ratio
def getHitRatio(ranklist, gtItem):
    for item in ranklist:
        if item == gtItem:
            return 1
    return 0

# NDCG
def getNDCG(ranklist, gtItem):
    for i in range(len(ranklist)):
        item = ranklist[i]
        if item == gtItem:
            return math.log(2) / math.log(i+2)
    return 0
```

On the validation set we observed a Hit Ratio@10 of 0.998 and a NDCG@10 score of 0.443. The following plots depict how the Hit Ratio@K and NDCG@K scores varied over different K.





3.5 SOTA Implementation:

3.5.1 Implementation of SVD with bias terms

The steps to implement SVD with bias terms is the almost same as that of regular SVD as described in Section 3.3 except the model is initialized with one small change. The *biased* parameter is set to true.

```
svdb_model = SVD(n_factors=4, n_epochs=10, biased=True)
svdb_model.fit(trainSVD)
```

3.5.2 Implementation of Neural Collaborative Filtering

The NCF model is implemented with the TensorFlow library. The model takes in 3 types of input - playlist ID, track ID, and metadata features (album share and artist share). To accommodate these inputs the following Input layers are built.

```
playlist_input = Input(shape=(1,), dtype='int64', name = 'playlist_input')
track_input = Input(shape=(1,), dtype='int64', name = 'track_input')
meta_input = Input(shape=(2,),name = 'meta_input')
```

For learning the latent factors, two Embedding layers are initialized. Each of which learn factors with 4 latent features for each playlist and each track. The output of these layers are then converted into one dimensional vectors using Flatten.

```
playlist_input = Input(shape=(1,), dtype='int64', name = 'playlist_input')
track_input = Input(shape=(1,), dtype='int64', name = 'track_input')
meta_input = Input(shape=(2,),name = 'meta_input')

# Embedding layers for playlist latent factors
```

```
playlist_embedding=Embedding(input_dim=num_playlists,
                             output_dim=num_factors,
                             name='playlist_embedding',
                             embeddings_initializer=GlorotNormal(),
                             embeddings_regularizer=L2(),
                             input_length=1
                             )
# Embedding layers for track latent factors
track_embedding=Embedding(input_dim=num_tracks,
                          output_dim=num_factors,
                          name='track_embedding',
                          embeddings_initializer=GlorotNormal(),
                          embeddings_regularizer=L2(),
                          input_length=1
                          )

# Flatten the Embeddings
playlist_factors=Flatten(name='flatten_playlist_embedding')(playlist_embedding(playlis
t_input))
track_factors=Flatten(name='flatten_track_embedding')(track_embedding(track_input))
```

The factors are then concatenated with the metadata features (album and artist share).

```
vector=Concatenate(name='concat_inputs')([playlist_factors,track_factors,meta_input])
```

The concatenated vector is then passed into a multilayer perceptron with 3 hidden layers of 16, 8, and 4 units respectively. Each of these layers involve L2 regularization and use the ReLU activation function.

```
layer1=Dense(16,
             kernel_regularizer=L2(),
             activation='relu',
             name='layer1'
             )(vector)
layer2=Dense(8,
             kernel_regularizer=L2(),
             activation='relu',
             name='layer2'
             )(layer1)
layer3=Dense(4,
             kernel_regularizer=L2(),
             activation='relu',
             name='layer3'
             )(layer2)
```

The final layer provides the output. The weights for this layer are randomly initialized from a uniform distribution. A sigmoid activation function converts the output into a

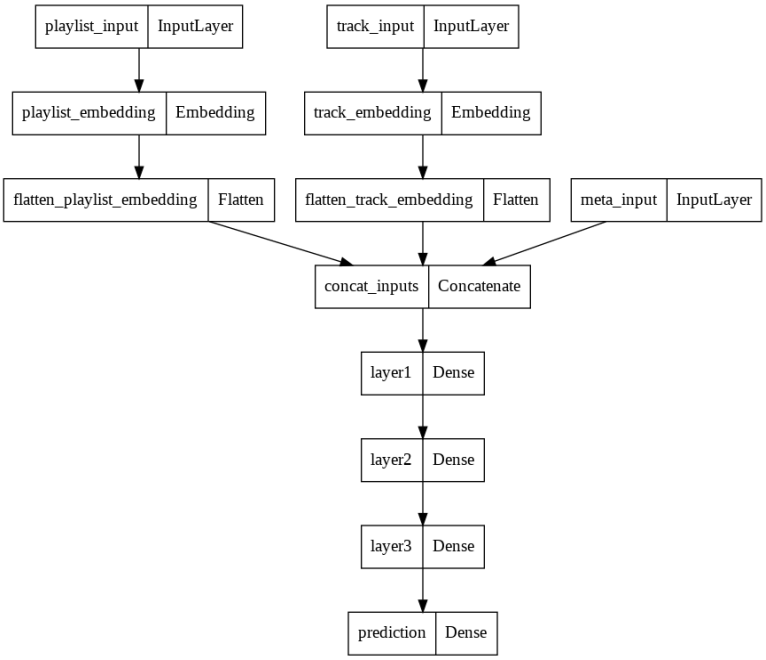
probability score.

```
prediction=Dense(1,
                 kernel_initializer=LecunUniform(),
                 activation='sigmoid',
                 name='prediction'
                 )(layer3)
```

The layers are then bought together to form a model as follows.

```
Model(inputs=[playlist_input,track_input,meta_input],
              outputs=prediction)
```

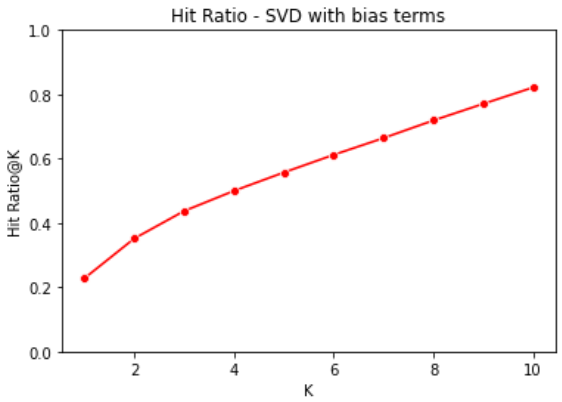
The complete architecture is visualized below.

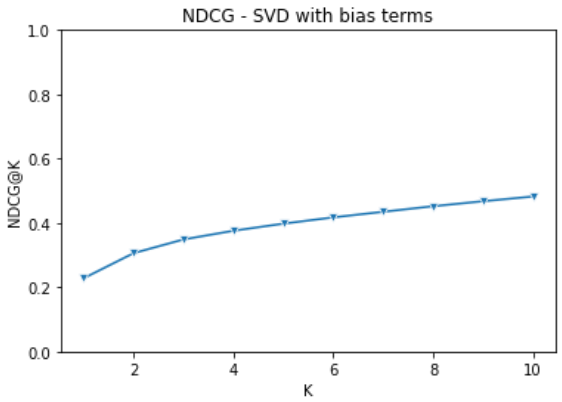


3.6 SOTA Results:

3.6.1. Results of SVD with bias terms

On the validation set we observed a Hit Ratio@10 of 0.821 and a NDCG@10 score of 0.482 for this variation of SVD. The following plots depict how the Hit Ratio@K and NDCG@K scores varied over different K.

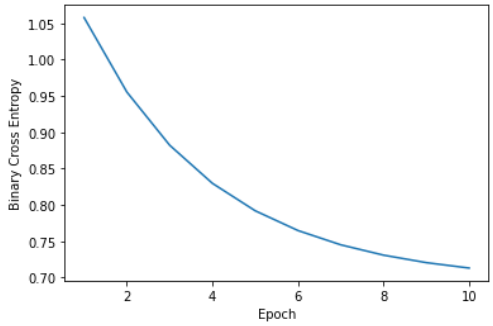




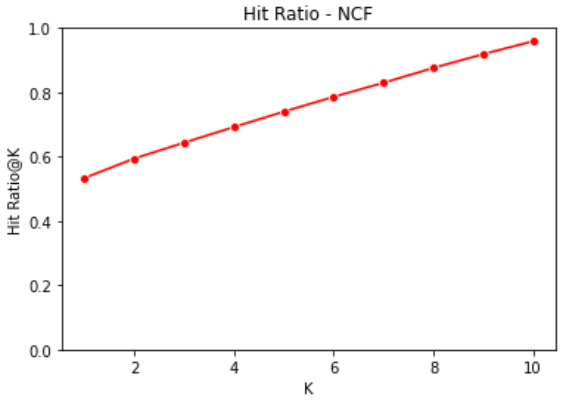
3.6.1. Results of Neural Collaborative Filtering

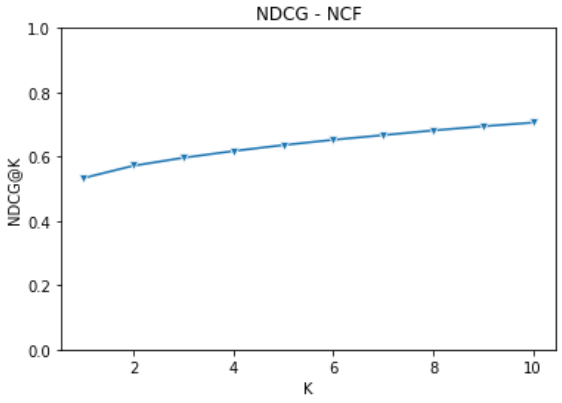
The best results for NCF were obtained with the following parameters.

Optimizer	Stochastic Gradient Descent
Loss Function	Binary Cross Entropy
Learning Rate	0.001
Batch Size	256
Number of Epochs	10
Number of latent factors	4



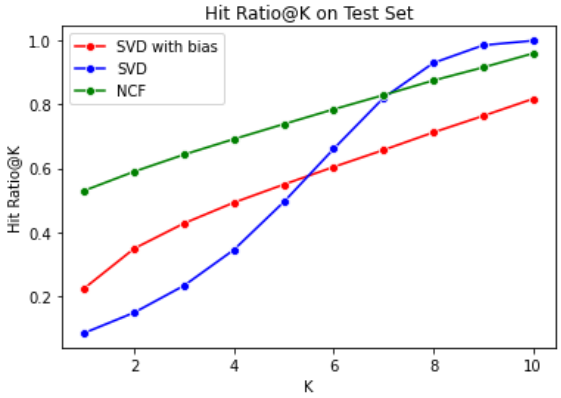
We observed a Hit Ratio@10 of 0.958 and a NDCG@10 score of 0.705 for this variation of SVD. The following plots depict how the Hit Ratio@K and NDCG@K scores varied over different K.

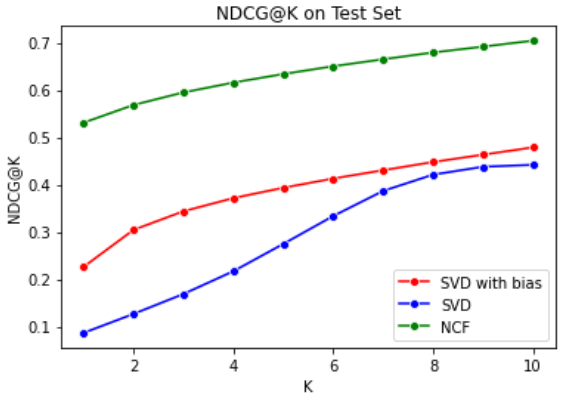




3.7 Compare and contrast:

The performance of all three models - regular SVD, SVD with bias terms, and Neural Collaborative Filtering, on the test set is depicted in the plots below.





We observe that regular SVD has a better hit ratio for higher values of K than other approaches. However the hit ratio drops to its lowest for lower values of K. The NCF model gives a consistently increasing hit ratio over all values of K. When it comes to NDCG, we see that the NCF model significantly performs better than other approaches by a large amount. Also, SVD with bias terms performs better than regular SVD.

4. Conclusion

Since NDCG takes the rank of relevant items in the recommendation list into account during evaluation, it is a better metric when compared to hit ratio. A higher NDCG value of NCF, tells us that the model is better at assigning higher ranking to relevant items in a list of recommendations than the other two approaches. Ranking of recommendation is crucial when these model are used in different domains. The most relevant item to a user should be at the very top of the recommendations rather than at the bottom. Therefore, the NCF model is superior to SVD.

5. Acknowledgements

We would like to express gratitude to professor of the course Dr. Tsui Weng for the guidance and support provided throughout the project. We would also like to thank the TA's for this course - Ali and Manoj, for their help.

6. References

[1] Digital Music Alliance https://dima.org/

[2] 2018 Annual Music Report https://dima.org/wp-content/uploads/2018/04/DiMA-Streaming-Forward-Report.pdf

[3] Spotify Million Playlist Dataset Challenge: https://www.aicrowd.com/challenges/spotify-million-playlist-dataset-challenge

[4] RecSys challenge 2018: https://www.recsyschallenge.com/2018/

[5] Smith, B. (2021) How TikTok reads your mind, The New York Times. The New York
Times. Available at:
https://www.nytimes.com/2021/12/05/business/media/tiktokalgorithm.html
(Accessed: November 9, 2022).

[6] Rocca, B. (2019) Introduction to recommender systems, Towards Data Science. Available at: https://towardsdatascience.com/introduction-to-recommender-systems6c66cf15ada (Accessed: November 8, 2022).

[7] Akalin, A. (2020) Matrix Factorization Methods, 30 September. Available at: https://compgenomr.github.io/book/matrix-factorization-methods-for-unsupervised-multiomics-data-integration.html (Accessed: November 8, 2022).

[8] Gunderson, G. (2018) “Singular Value Decomposition as Simply as Possible,” Gregory Gunderson, 18 December. Available at:
https://gregorygundersen.com/blog/2018/12/10/svd/ (Accessed: November 8, 2022).

[9] McAuley, J. (2022) “Model-based Approaches to Recommendation,” in Personalized
Machine Learning. Cambridge, United Kingdom, Cambridge: Cambridge University Press, pp. 106–131.

[10] He, X. et al. (2017) “Neural collaborative filtering,” Proceedings of the 26th International Conference on World Wide Web [Preprint]. Available at:
https://doi.org/10.1145/3038912.3052569.

[12] C.W. Chen, P. Lamere, M. Schedl, and H. Zamani. Recsys Challenge 2018: Automatic Music Playlist Continuation. In Proceedings of the 12th ACM Conference on
Recommender Systems (RecSys ’18), 2018.