**Neural Collaborative Filtering for a Music Recommendation Engine**

Bratislav Petkovic

College of Information Technology, WGU

Dr. William Sewell

March 25th, 2023

**Research Question**

Companies with product and service oriented business models rely on customer consumption to earn profit. In an effort to increase consumption, online platforms have developed recommendation systems to provide their users with purchase suggestions. The purpose of this study is to generate a recommendation model capable of producing suggestions for music platform users by using data of real Spotify listeners to answer the following research question:

**Can a Neural Collaborative Filtering model be used for a music**

**recommendation system?**

The null and alternate hypotheses are defined as :  **H0 : An NCF recommendation system cannot be designed with an accuracy of > 90%**

**H1 : An NCF recommendation system can be designed with an accuracy of > 90%**

Advancements in the field of artificial intelligence have empowered e-commerce platforms to design increasingly more powerful recommendation systems. The alternative hypothesis is supported by the work of He et al. (2017) in which NCFs showed empirical evidence of displaying more generalized yet accurate results over other recommendation system approaches since they can learn complex non-linear data.

**Data Collection**

 The data was collected from real Spotify users whose identities have been anonymized. The dataset has been published publicly and is available for free on [Kaggle](https://www.kaggle.com/datasets/andrewmvd/spotify-playlists) which is an online data science community made up of machine learning competitions and diverse datasets. The collected data comprises of more than +15,000 users, +289,000 artists and +2,000,000 songs. It should be noted that recommendation engines leverage implicit data which comes by measuring number of user clicks, purchases, or views. In the case of the Spotify dataset implicit data came in the form of user plays.

**Table 1**

*Spotify Dataset from Kaggle*

|  |  |
| --- | --- |
| Field | Type |
| UserID | Categorical |
| Artist | Categorical |
| track | Categorical |
| playlist | Categorical |

The data was publicly published and the real user-identities that the data was collected from are protected. A disadvantage of the dataset is that certain rows are badly formatted or are missing key information. A key advantage of the dataset, negating the presence of unusable rows is that there is an ample quantity of correctly formatted records to perform the data analysis. One challenge which arose during the data collection phase was the lack of a label that the recommendation system would be able to classify. To overcome the challenge, a “leave-one-out” cross validation technique was used to split the training and validation data (Wang, 1970). For each user, an artist that the said user listened to will be removed from the training data and placed in the testing data. This removed item is known as the holdout item.

**Data Extraction and Preparation**

To extract and prepare data for analysis, Python was used. First, the Pandas package was used to load the data from the comma separated values (.csv) file into a data frame. Next, the NumPy package was used to perform vector and matrix operations such as sorting, aggregation, and random sampling. The playlist feature was removed and UserID and Artist columns were cast into the category type. New columns called “user\_id” and “artist\_id” were created by tokenizing columns UserID and Artist, respectively. Furthermore, the data was split into training and testing using a helper function implementing the leave-one-out cross validation method (Figure 1). The training data was used to fit the model while the testing data was used to evaluate the model’s recommendation capabilities. The advantage of using the leave-one-out train-test-split technique was that it allowed to directly check if the model would predict the holdout item without actually being explicitly told that a given user has already consumed the holdout item. The disadvantage of it was that the selection of the holdout item was arbitrary – because dates were not present it was not possible to make the most recently consumed item the holdout item.

**Figure 1**

*Code to Read & Prepare Spotify Data*

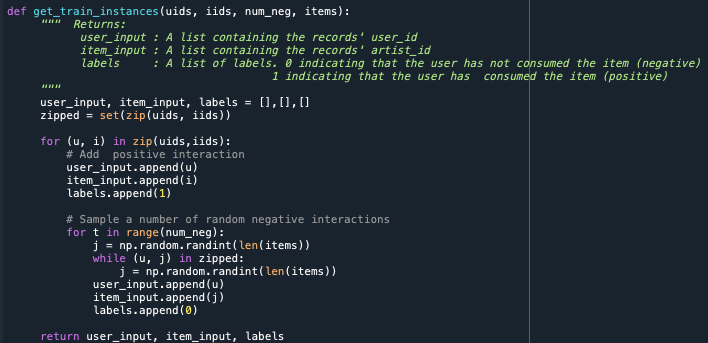
Text

Description automatically generated

Additionally, since the data was implicit, it only displayed the artists a particular user consumed. Therefore, negative labels had to be artificially created. To produce negative labels a helper function called “get\_train\_instances” was constructed (Figure 2). The function randomly sampled *N* artists that the user did not consume for each unique user-artist combination. The function returns 3 lists : a list of user\_ids, a list of artist\_ids and a list of labels indicating if user\_id has listened to artist\_id. In the study, it was decided that parameter “num\_neg” or *N* was optimal at 4 – for each positive label there were 4 negative labels. The advantage of artificially creating negative labels was that it enabled the recommendation system model to differentiate between negative and positive labels and therefore capture the underlying relationship between users and the items they consume.

**Figure 2**

*Code to Create Negative & Positive Interactions*

****

**Analysis**

The analysis was performed in 4 steps. First, a model was constructed given a set of parameters. Second, the model was fitted using the training data and the specified parameters. Third, for each user present in the dataset, the model generated a prediction which is treated as a recommendation. Fourth, the model was evaluated by assessing the accuracy of the item predictions. Lastly, the four steps are repeated with a different set of parameters to increase the accuracy.

**Model Architecture**

The architecture of the Neural Collaborative Filtering model was influenced by the work of He et al. (2017) in which General Matrix Factorization (GMF) and a Multilayer Perceptron (MLP) are ensembled in a single model (Figure 3). GMF and MLP each consisted of identical input embeddings layers made up of a user and item latent vectors. Although the input to the user and item embedding layers was identical for the GMF and MLP, the embedding output dimensionality was different – the output size of the MLP embedding layer was one half of the units of the first hidden neural network layer while the output size of the GMF was set to the latent feature parameters. Because matrix factorization techniques – first implemented by Lee and Seung (2000) – are simple and only capture linear relationships in user-item datasets, they were only composed of 2 embedding layers. The MLP section, however, was more complex and required a greater attention to detail. To begin, the default MLP architecture consisted of 4 hidden layers, implemented via the Dense object of the Keras library. The 4 hidden layers were initialized with a tower pattern in which the first layer has the most units and the last layer has the least. The advantage of using a tower structure is that the neural network is capable of learning more abstract features – a Microsoft AI team discovered that deeper neural networks with shrinking layer sizes yield better results for image recognition (He, K. et al, 2016).

Rectified linear unit function was chosen as the activation function for the Dense layers because it is less prone to overfitting – sigmoid and tanh functions are at a higher risk of vanishing gradient issues in which weights incorrectly vanish towards 0 (Thakur, 2022). To prevent any additional overfitting of the model, an L2 regularization technique was implemented for all hidden layer with varying magnitudes. As explained by Payne (2022), the benefit of L2 regularization is that it minimizes the generalization error by preventing overfitting. Once all hidden layers of the MLP were added, the GMF and the MLP models were concatenated together. The output layer of the model was given a sigmoid activation function which is frequently used for classification tasks among neural networks. Finally, the model was compiled using the Adam optimizer and binary cross entropy for the loss function. As explained by Ajegakar (2021), the Adam optimizer consistently outperforms other optimizers and was therefore the first choice in the study.

**Figure 3**

*Model Structure*

**Diagram

Description automatically generated**

**Fitting The Model**

Once the model was constructed, it had to be fitted using the training data which was generated in the data preparation step. The two parameters which impacted the speed and performance of the training were batch size and the number of epochs to train the data on. According to Smith et al (2018), the batch size to learning rate ratio influences the generalization error greatly – the researchers discovered that higher batch size to learning rate ratios are associated with higher generalization error. Upon testing multiple batch sizes and learning rates, it was asserted that a learning rate of 0.01 and a batch size of 5000 maximized the test accuracy.

As Figures 4 and 5 demonstrate, the optimal number of epochs was determined to be X. When the model was trained for less than X epochs, the model was underfit and when it was trained past that the model validation accuracy started decreasing, indicating that overfitting to the training data was occurring.

**Figure 4 Figure 5**

**A picture containing graphical user interface

Description automatically generated A picture containing chart

Description automatically generated**

**Prediction and Evaluation**

After fitting the model, it was capable of generating recommendations when given an input of items that a user has already consumed. Figure 5 depicts the recommendations generated by the model for the user who has already listened to the specified artists in Figure 6. Since a sigmoid activation was used in the output layer, the predictions came in the form of a list of items and an associated probability that the user would consume the item – higher probabilities indicate that the model was more confident that the item would be consumed. For example, for the user with id equivalent to 6 the holdout item which the model never saw during training was the artist named “Benga”. The model gave “Benga” 99% probability of being consumed by the specified user and is therefore consider a highly relevant recommendation.

**Figure 5 Figure 6**

*The artists liked by user 6 Model recommended artists for user 6*

**Graphical user interface

Description automatically generated with low confidence A picture containing text

Description automatically generated**

To evaluate the performance of the whole model, predictions had to computed for each user which turned out to be computationally expensive. Running the predictions for each user took an hour and forty minutes or approximately 0.4 seconds per user on average. Once all predictions were completed, records whose probabilities were greater than 0.5 were labeled as positive and those with less than 0.5 were labeled as negative. A positive label indicates an item that a user is likely to consume. Then, accuracy was calculated by dividing the number of positive labeled holdout items by the number of all holdout items which is synonymous to recall. As Lendave explains in “How to Measure the Success of a Recommendation System” (2021), traditional classification metrics such as precision, recall and F1 are best markers of measuring the relevancy of the generated recommendations. A specific advantage of using recall was that the data processing was kept minimalistic since it required least processing power – precision would have been more computationally expensive and would have decreased the size of the training data. However, using recall also presented a disadvantage since it did not consider the relative rank of the generated recommendation.

**Data Summary and Implications**

By carefully considering all training and architecture parameters, the final model reaches a recall accuracy of 85%. Since the NCF model does not prove capable of reaching an accuracy of 90%, the null hypothesis is accepted. Nonetheless, the model proves useful and is able to generate relevant music artist recommendations for the majority of users.

Since the number of items consumed varied significantly by artist, exploratory analysis was performed to determine if accuracy was impacted by the magnitude of item consumerism. A bar plot is created to determine the association between the number of items consumed and the models accuracy in recommending the holdout item (Figure 7). From the analysis, it is evident that recall accuracy is not affected by the number of items that a user consumed – regardless of the number of items consumed, model performance stays relatively equal. Performing the exploratory analysis reveals that the model is generalizing well when faced with fewer items consumed.

**Figure 7**

**Chart, bar chart

Description automatically generated**

One limitation of the analysis was the computational expense that was required. Training the model lasted for more than 2 hours for 8 epochs only. Generating predictions for each user added an extra 2 hours. By decreasing the learning rate or batch size or increasing the model complexity the training and validation computations increased significantly, making it increasingly more time and computation costly to perform the analysis. One way that the analysis limitation could be offset is by using machine learning platforms which provide GPUs for hire such as Amazon Sage Maker.

For the research question to be answered, it is recommended that the current model is expanded further before it could be deployed for the business use case. There are three main approaches which have the potential to improve the model performance.

The first approach requires increasing the dataset size used to train and test the model. Having more users would allow the model to better understand the patterns of music consumption behavior that listeners exhibit. In other words, by having more content to learn from, the model would become smarter.

The second approach in upgrading the recommendation system would be to implement content-based filtering. As Dmitry Pashtukov informs in his breakdown of Spotify’s multifaceted recommendation system (2022), making inferences based on artist and track metadata is incredibly valuable in recognizing similar artists and tracks. The specific music features that mark similarities between artists are genre, primary language and mood, style, and culture tags. By incorporating these features, it is possible to perform a supplementary K-Nearest Neighbor model to derive similar artists to serve as recommendations.

The third method requires implementing Natural Language Processing to analyze lyrics – a huge part of most musicians’ identities. Additionally, NLP techniques would enable dimensionality reduction of the item population. As seen in Figure 8, an artist named Akon is categorized as a separate item for each new featuring artist he has collaborated with. By performing simple NLP methods such as cosine similarity or syntactic analysis, the below artists could all be categorized as Akon. By reducing item dimensionality, the NCF model would be able to make better inferences about the data – lemmatization and stemming are not dissimilar to the proposed NLP dimensionality reduction technique.

**Figure 8**

*Example of High Dimensionality*

**A picture containing graphical user interface

Description automatically generated**

Recommendation systems are continuing to be studied by AI researchers and consumption-dependent businesses. Neural Collaborative Filtering has asserted itself as one of the most powerful standalone recommendation systems to date. Despite the constructed model not reaching the proposed accuracy target, it demonstrates tremendous value to the stakeholders of the proposed business use case. More importantly, the model gives an option of being coupled with the aforementioned techniques. The next steps of the study should be to create an ensemble model by combining the constructed NCF and the proposed NLP model.

**References**

Ajagekar , A. (2021). *Adam*. Adam - Cornell University Computational Optimization Open Textbook - Optimization Wiki. Retrieved April 2, 2023, from <https://optimization.cbe.cornell.edu/index.php?title=Adam>

He, K., Zhang, X., Ren, S., & Sun, J. (2015, December 10). *Deep residual learning for image recognition*. arXiv.org. Retrieved April 2, 2023, from <https://arxiv.org/abs/1512.03385>

He, X., Liao, L., Zhang, H., Nie, L., Hu, X., & Chua, T.-S. (2017, August 26). *Neural collaborative filtering*. arXiv.org. Retrieved April 2, 2023, from <https://arxiv.org/abs/1708.05031>

Lendave, V. (2021, October 23). *How to measure the success of a recommendation system?* Analytics India Magazine. Retrieved April 3, 2023, from <https://analyticsindiamag.com/how-to-measure-the-success-of-a-recommendation-system/>

Pashtukov, D. (2022). *How Spotify's algorithm works? A Complete Guide to spotify recommendation system [2022]: Music tomorrow blog*. How Spotify's Algorithm Works? A Complete Guide to Spotify Recommendation System [2022] | Music Tomorrow Blog. Retrieved April 4, 2023, from <https://www.music-tomorrow.com/blog/how-spotify-recommendation-system-works-a-complete-guide-2022>

Payne, M. (2022). *Neural collaborative filtering for deep learning based recommendation systems: Architecture Breakdown & Business Use Case*. Width.ai. Retrieved April 2, 2023, from <https://www.width.ai/post/neural-collaborative-filtering>

Smith, S. L., Kindermans, P.-J., Ying, C., & Le, Q. V. (2018, February 24). *Don't decay the learning rate, increase the batch size*. arXiv.org. Retrieved April 2, 2023, from <https://arxiv.org/abs/1711.00489>

Thakur, A. (2020, August 19). *Relu vs. sigmoid function in deep neural networks*. W&B. Retrieved April 2, 2023, from <https://wandb.ai/ayush-thakur/dl-question-bank/reports/ReLU-vs-Sigmoid-Function-in-Deep-Neural-Networks--VmlldzoyMDk0MzI>