Music Recommender System

Jason Ingram
CSUN
Computer Science Department
jason.ingram.806@my.csun.edu

Isaiah Martinez
CSUN
Computer Science Department
isaiah.martinez.891@my.csun.edu

Introduction **2**
Background 2
Project Description 2
Dataset Overview and Preparation 2
Web Crawlers 3
Outline 3
Related Work **4**
Collaborative Filtering 4
Neural Collaborative Filtering 4
K-Nearest Neighbors 5
Methods **5**
Collaborative Filtering 5
Neural Collaborative Filtering 6
K-Nearest Neighbors 7
Evaluation **7**
Performance Metrics 7
Collaborative Filtering 8
Neural Collaborative Filtering 8
K-Nearest Neighbors 8
Conclusion **8**
Future Work **9**
References **10**

Introduction

Background

Music is a major component of social culture across the world [1]. It influences a lot of people's lives and is even a career path for many people [2, 3]. With such a broad range of influence, many music enthusiasts and enjoyers try to compartmentalize the different styles of music to better describe the melodies, artists, and ideas behind these songs. Additionally, many people like to listen to new music and expand their music knowledge. 20 years ago, this was done by listening to the radio station, or going to local music shows. New technological advances have allowed for popular apps such as Pandora, Spotify, and Apple Music to recommend new music to users of such services [4]. In an effort to try making a music recommendation service of our own, we utilized popular methods for music recommenders.

Project Description

Our project utilizes 3 algorithms: Collaborative Filtering (CF), Neural Collaborative Filtering (NCF), and K-Nearest Neighbors (KNN). These 3 different approaches were used on the same dataset which will be described in the next section of this paper. Our collective goal was to use this data to feed into the models created using the aforementioned algorithms in order to predict whether a user would listen to a particular song or not. Thus, this is a binary classification project. We utilized R for obtaining the data and Python to train and evaluate the model.

Dataset Overview and Preparation

Initially, the intention was to utilize Web Crawlers to obtain Music Data from a variety of sources. However, after successfully obtaining data from LastFM, MusicMap, and MusicBrainz, we quickly realized that the data obtained was unfit for training any models. There was a lot of data cleaning that would have to be done manually. Alternatively, deleting such data would mean that we would have lost about half of the data amassed, from 200,000 to just under 80,000 entries. Additionally, there was no user data to accompany the song data. This proved to be the final nail in the coffin for this approach for data collection. For further information towards the Web Crawlers, see the following section.

As a result, we looked into other sources for information containing both music and user data, eventually stumbling upon the servers utilized by MusicBrainz, called ListenBrainz. This database contains both music data and user data for all entries. This publicly available data is available for download [5]. They only keep a few entries of data, presumably for storage cost purposes. Although the data we obtained is no longer available, a similar dataset may be obtained that is more recent.

The files downloaded are contained in a unique tabular format, '.parquet', where the data is stored vertically. This requires the use of a 3rd party library to extract the data called Apache Arrow, as we would be using R for this process [6]. Once this was extracted, we converted the Tabular data into a dataframe, and exported as a CSV to be utilized without the

need for external libraries. The resulting CSV had the following columns: **user_id**, **artist_name**, **release_name**, **recording_name**, **date**, **time**. Each parquet file contains over thousands of rows. After cleaning each parquet file's data, we combined all the CSV's for training.

For using the data in the CF and NCF algorithms, we created User Item Interaction Matrices for processing the interactions between the Users and the Artists of a particular song. This allowed for a straightforward understanding of input and output of these algorithms.

USERS \ ARTISTS	Artist 1	Artist 2	Artist 3
User 1	1	0	1
User 2	0	1	0
User 3	0	1	1

Figure 1: Example of a User Interaction Matrix

Web Crawlers

The design of the first Web Crawler was to utilize a technique called snowballing. It would begin with an artist name, then obtain a list of related artists from MusicMap [7]. This would be stored into a text file, with each line representing an artist. Each artist would then be queried on LastFM to obtain related song data. This included Albums, Tags, Songs, Song Length, etc. These would be saved into a CSV. This 2 step process would allow for a very rapidly growing and interconnected set of artists, which although biased, may have yielded consistent results.

We set out to create a song recommender system that suggests tracks users will love, based on their past music choices. Our initial challenge was dealing with large datasets and ensuring our system could handle real-world data effectively. I started working on a scraper for MusicBrainz, which, while rich in song metadata, lacked the user interaction metrics essential for personalized recommendations. We transformed Spotify's data into user-item interaction matrices. This step was key to preparing our data for the recommendation algorithms.

Outline

The outline for the following sections of the paper are as follows: First, we will discuss the Related Works in regards to music recommendation. Second, we will describe the Methods of implementing the 3 algorithms of interest: CF, NCF, KNN. Afterwards, we will briefly describe the evaluation methods we utilized, as well as the Performance Evaluation for each model. This will allow us to culminate the results of the performance analysis in the Conclusion section. These results will allow us to discuss the areas of improvement and Future Work, the final section of this document.

Related Work

Because our dataset lacks a definitive user rating, this means that our algorithms need to be able to handle implicit user data [8, 9].

Collaborative Filtering

CF is the first algorithm that would be utilized. The particular instance of CF utilized is Alternative Least Squares (ALS) [10, 11]. This algorithm is widely used to handle implicit data. It is available via a github repository that was installed via pip [12]. To fine-tune the performance of the model, we utilized Grid Search to find optimal hyperparameters for the best performance using AUC. Grid Search is a brute-force algorithm to test all given parameters and parameter values to see which performs better.

Neural Collaborative Filtering

NCF is a recent innovation within the DL discipline where the creators of the algorithm sought to utilize DL methods of Neural Networks (NN), in tandem with the Collaborative Filtering method [13]. In order to improve the speed of the algorithm, I utilized the GPU instead of the CPU for processing. This sped up the process from hours to minutes, at the cost of the complexity of the model structure.

The general model structure can be seen in the below picture from a publicly available github repository discussing different types of recommender systems [14]:

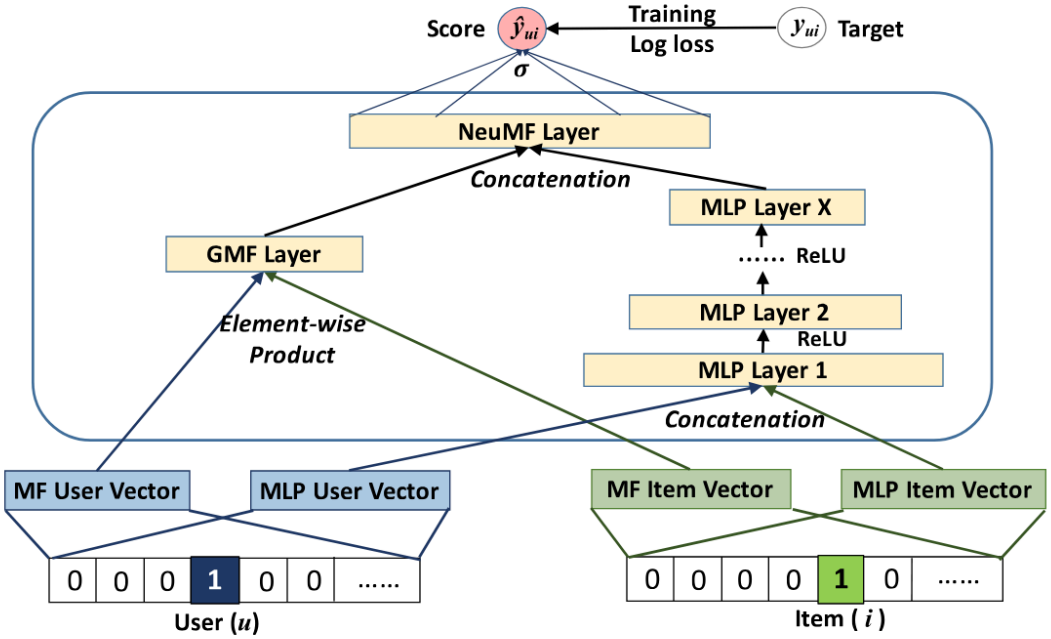

Figure 2: NCF model summary

It operates by taking 2 inputs of the User and Item Vectors, then performs the Dot Product to obtain the User Item Matrix. This gets fed into subsequent hidden layers (in the above image as Multi-Layer Perceptrons (MLP) layers). At the final hidden layer, the output gets sent to the binary classifier which in our case utilizes the sigmoid activation function (with 1 representing an interaction and 0 meaning no interaction).

Another approach towards utilizing NCF from a different package may be found here [15]. This source utilizes simplistic code, so it was used only as a reference for knowledge acquisition.

K-Nearest Neighbors

[16, 17]

Methods

Below we will describe the algorithms of interest in detail.

Collaborative Filtering

Collaborative Filtering operates by looking at a single vector (Users or Items) and optimizes the results between them. This then occurs for the non-chosen vector. The results for each optimization are then compared until they reach convergence [10, 11, 12]. When convergence is achieved, the algorithm is complete and is ready to be evaluated. This process begins on smaller k-sized matrices. In terms of hyperparameters, k = factors.

```
import implicit
#model fitting
model = implicit.als.AlternatingLeastSquares(factors = 150, regularization = 0.001, iterations = 100)
model.fit(user_artist_train_matrix)
```

Figure 3: CF model training

There were 4 hyperparameters that would be tuned for optimal performance: factors, regularization (to prevent overfitting), iterations (number of times to train), and alpha (amount of weight attributed to predictions) (not pictured in Figure 3). To achieve optimal results, we utilized Grid Search for testing a wide range of values for each hyperparameter. Figure 4 demonstrates one of the parameter grids used for Grid Search.

Utilizing this technique we found the best hyperparameters to run with the given dataset: a factor of 8, regularizer of 0.2, number of iterations of 75, and an alpha of 10.

```
#define param grid to search
param_grid = {
    'factors' : [8, 10, 15],
    'regularization' : [0.15, 0.18, 0.2, 0.23, 0.27, 0.3],
    'iterations' : [60, 65, 75, 80, 85],
    'alphas' : [10]
}
```

Figure 4: Parameter Grid

Neural Collaborative Filtering

Initially, I utilized the CPU for the NCF algorithm, but this proved to be very time inefficient. Thus, I opted to utilize the GPU for better performance. My PC specs are: i7-10700k CPU and ASUS RTX 3070 GPU for reference.

In both cases, I had created several types of CNF Models with varying amounts of layers. Below is the final model created with a total of 2 input layers, 2 embedding layers, 3 hidden layers, and output layer with sigmoid activation function.

Layer (type)	Output Shape	Param #	Connected to
user_input (InputLayer)	[(None, 1)]	0	[]
item_input (InputLayer)	[(None, 1)]	0	[]
user_embedding (Embedding)	(None, 1, 5)	77220560	['user_input[0][0]']
item_embedding (Embedding)	(None, 1, 5)	77220560	['item_input[0][0]']
flatten (Flatten)	(None, 5)	0	['user_embedding[0][0]']
flatten_1 (Flatten)	(None, 5)	0	['item_embedding[0][0]']
dot (Dot)	(None, 1)	0	['flatten[0][0]', 'flatten_1[0][0]']
dense (Dense)	(None, 2048)	4096	['dot[0][0]']
dropout (Dropout)	(None, 2048)	0	['dense[0][0]']
dense_1 (Dense)	(None, 1024)	2098176	['dropout[0][0]']
dropout_1 (Dropout)	(None, 1024)	0	['dense_1[0][0]']
dense_2 (Dense)	(None, 512)	524800	['dropout_1[0][0]']
dropout_2 (Dropout)	(None, 512)	0	['dense_2[0][0]']
dense_3 (Dense)	(None, 1)	513	['dropout_2[0][0]']

Total params: 157,068,705
Trainable params: 157,068,705
Non-trainable params: 0

Figure 5: NCF Model Structure

K-Nearest Neighbors

I started with K-Nearest Neighbors because it's straightforward and effective at finding user or item similarities. However, KNN struggled with the large dataset due to its memory demands. I considered using more scalable alternatives like SVM, but thought that I was stuck with what I had chosen. In the future, I would like to attempt this project again using different algorithms.

To handle the heavy lifting of data processing, I used Google Cloud Services. This move was a game changer, allowing me to manage and process our data at scale.
Google Cloud introduced an entirely new set of problems that I was wholly unaware of. Implementing a fully functional application on google cloud is an intense undertaking. Ultimately, I was unable to obtain a working version of my application.

Evaluation

Performance Metrics

To evaluate the different models, we utilized a variety of techniques for the given implicit model recommenders: Area under the ROC Curve (AUC), Loss, Accuracy, Precision, and Recall [18, 19]. AUC is the measure of the area under the ROC curve, which is a graphical metric to compare the rates of False positives (FP) to True positives (TP). In our case, we didn't visualize the AUC, instead we took the raw value.

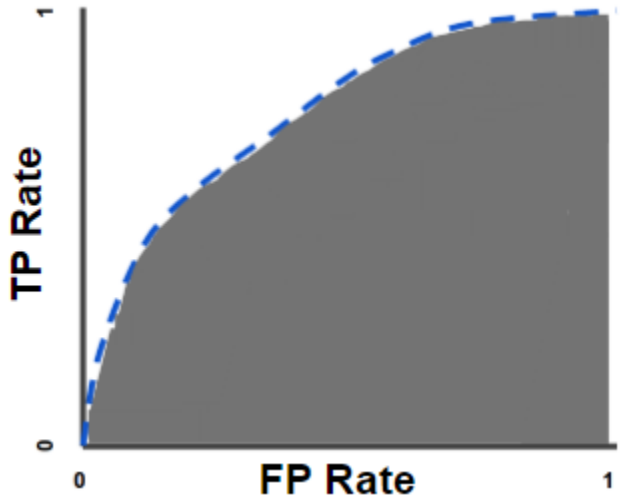

Figure 6: ROC Graph with AUC being the gray region

Loss is a measure of how well a model is predicting values (typically classes) with a loss approaching +0 being desirable [20]. Accuracy is the measure of how well the model predicts True Positives and True Negatives, which for our dataset represents a user liking a particular artist or song. Precision is the measure for the amount of true positives over total positives, i.e. correctly predicted artists/songs that are liked by a user. Recall is the same except instead of

positives, it looks at negatives, therefore it looks at the correctly predicted artists/songs not liked by a user.

For reading the results, we indicate an interaction between a user and artist/song as a positive value of 1, and no interaction as 0.

Collaborative Filtering

For Collaborative Filtering, I utilized the AUC method of evaluation. This resulted in a very poor score of 0.5473. This means that given any random user and artist, the model would do slightly better than a coin toss (50/50). I found this quite perplexing as this algorithm is supposed to be rather powerful and accurate for any sized data. Despite numerous attempts at hyperparameter changes, this value remained the best for any parameters searched with Grid Search.

Neural Collaborative Filtering

Because of the surprisingly low performance of CF, I hypothesized that the NCF model would outperform the CF model because of the innate ability for NN's to analyze complex relationships between data. My suspicions were wrong.

Despite obtaining an extremely high Accuracy and low loss (99.83%, 0.0095 respectively), Precision and Recall were so low that they were less than $1 * 10^{-3}$. This indicates that the model was overfitting (memorizing) the data. Despite numerous changes to the model structure, this yielded the best results in terms of accuracy and loss with Precision and Recall approaching 0.

K-Nearest Neighbors

My first idea was to measure KNN's performance using precision at K and recall at K. Realizing that I did not have a way to create a dataset to test against, I decided that K-fold Cross-validation would be a much better choice. The lack of a working model prevented me from implementing the function that I had designed for that purpose.

My lack of results has stymied my team from being able to compare our results.

Conclusion

I was unhappy with the results I obtained because I was hoping that the NCF model would be extremely good due to the complexity of the data. I had also initially designed a complex, multi-layered Hybrid approach of combining a series of recommender systems to reduce complexity at each stage and improve runtime, but simply had no time to complete it since the initial models were performing so poorly. Perhaps adding this approach would have yielded better results. I think that because of the consistent poor performance from the models, as well as the poor performance from the KNN predictions recommending the same songs, there may have been a problem with the dataset of some kind that we didn't see.

However, I am satisfied with my efforts to make these 2 approaches work. I spent a lot of time researching which algorithms would be best for our data. I also spent a surprising amount of time looking into how to feed the data into them properly, as they frequently gave errors of incorrect format and so forth. I think that this process of trial and error is generally disliked amongst many professionals due to it being so simplistic, but I find it rather refreshing and fun. This is because the basis of Computer Science is logic and problem solving; trying new approaches and tweaks to already existing approaches is the basis of life, research, and learning in general.

My project did not meet the initial expectations set during its conception. The shift to Google Cloud, intended as a solution to our computational hurdles, unfortunately compounded the challenges without delivering the anticipated benefits. The system, as it stands, is unable to scale efficiently and falls short of providing the responsiveness required for a recommender system. The setbacks have provided valuable lessons.

Future Work

I would like to see the results of the hybrid approach that I designed, after further refinement of course. Taking these challenges in mind as well, I have a suspicion that perhaps the dataset might be not fed properly to these models (despite me checking several times). I would revisit the data in order to ascertain its correctness.

I think that I would like to run more tests to improve the performance of the CF algorithm as well. I believe it would be more fruitful, as running the NCF model puts lots of strain on my computer. It would use about 7.4 GB of the 8 GB VRAM available on my GPU. I was seriously worried about the health of my system since it would use such a high amount of resources for many hours at a time, but all PC health tests I've run say the system is fine.

All in all, this was a great experience to work on a project that is an active area of research (Spotify, Apple Music, etc.) with a large number of users. As always, it's a pleasure being in your class Ms. Lord, and allowing us as students to take on projects that will actually be useful in industry and are fun to take part in.

Moving forward, I aim to explore alternative algorithms that are less resource-intensive and more suited to cloud computing. Additionally, a deeper dive into cloud resource management and optimization is necessary to better harness the potential of cloud computing for large-scale data processing.

References

1. A. Huerter, “How Music and Culture Work Together: Science Behind Music,” *www.musichouseschool.com*, Sep. 28, 2020. https://www.musichouseschool.com/how-music-and-culture-work-together-science-behind-music
2. “12 Jobs With Music You Can Pursue With Salary and Duties,” *Indeed Career Guide*. https://ca.indeed.com/career-advice/finding-a-job/jobs-with-music
3. L. Peralta, “Impact of Music on Society - Sociological Effects,” *Save The Music Foundation*, Nov. 03, 2021. https://www.savethemusic.org/blog/how-does-music-affect-society/
4. D. Curry, “Pandora Revenue and Usage Statistics (2020),” *Business of Apps*, Aug. 27, 2020. https://www.businessofapps.com/data/pandora-statistics/
5. https://data.metabrainz.org/pub/musicbrainz/listenbrainz/fullexport/
6. https://arrow.apache.org/
7. https://www.music-map.com/
8. Victor, “ALS Implicit Collaborative Filtering,” *Medium*, Jul. 10, 2018. https://medium.com/radon-dev/als-implicit-collaborative-filtering-5ed653ba39fe
9. https://developers.google.com/machine-learning/recommendation/collaborative/basics
10. https://towardsdatascience.com/alternating-least-square-for-implicit-dataset-with-code-8e7999277f4b
11. https://towardsdatascience.com/recommendation-system-matrix-factorization-d61978660b4b
12. https://github.com/benfred/implicit
13. X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, “Neural Collaborative Filtering,” *Proceedings of the 26th International Conference on World Wide Web - WWW ’17*, 2017, doi: https://doi.org/10.1145/3038912.3052569.
14. https://github.com/recommenders-team/recommenders/blob/main/examples/02_model_collaborative_filtering/ncf_deep_dive.ipynb
15. https://medium.com/data-science-in-your-pocket/recommendation-systems-using-neural-collaborative-filtering-ncf-explained-with-codes-21a97e48a2f7
16. B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, “Item-Based Collaborative Filtering Recommendation Algorithms.” Available: https://www.ra.ethz.ch/cdstore/www10/papers/pdf/p519.pdf
17. R. M. Bell and Y. Koren, "Scalable Collaborative Filtering with Jointly Derived Neighborhood Interpolation Weights," *Seventh IEEE International Conference on Data Mining (ICDM 2007)*, Omaha, NE, USA, 2007, pp. 43-52, doi: 10.1109/ICDM.2007.90.
18. https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc/
19. https://www.evidentlyai.com/classification-metrics/accuracy-precision-recall
20. https://developers.google.com/machine-learning/crash-course/descending-into-ml/training-and-loss