Tutorial Report

Runpeng Li
rli3@oxy.edu
Occidental College

1 Introduction

For my Senior comps project, I am trying to develop a music recommendation system (MRS) similar to those used by prominent streaming services like Spotify and Apple Music. This system aims to enhance the listener experience by suggesting music and artists based on users' historical listening habits, effectively personalizing content delivery. The target audience for this project includes music enthusiasts and casual listeners, providing them the convenience of discovering new music and artists aligned with their tastes. This is particularly beneficial in the vast landscape of available music where new tracks and underground artists are often overlooked. Additionally, I plan to explore the integration of social listening trends to further refine the recommendations.

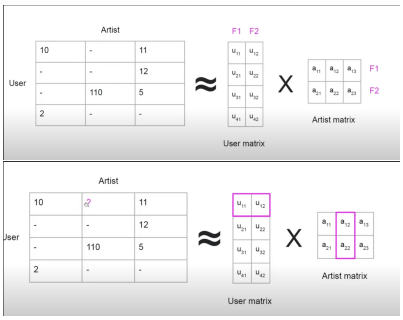
For my tutorial, I have chosen "Build a Spotify-Like Music Recommender System in Python" by Valerio Velardo [1] that uses collaborative filtering techniques, similar to those employed by Spotify, to recommend music to its users. This tutorial provides a practical framework for understanding and implementing collaborative filtering using a dataset from Last.fm, which tracks user listening habits. This approach is relevant to my comps project because it addresses the complexities of building scalable, efficient systems capable of handling vast amounts of data. The goal of this tutorial is not only to build a system that recommends music based on user preferences but also to deepen my understanding of underlying algorithms like matrix factorization and their application in real-world scenarios.

2 Method

2.1 Main Algorithmic Detail

The first step in the tutorial is explaining the Alternating Least Squares (ALS) method for collaborative filtering. ALS is particularly suitable for scenarios where there are many missing entries in the data, which is common in user-item interaction datasets. It excels at handling large datasets and sparse matrices, focusing solely on user behavior. This approach is ideal for our goal of user-based recommendations. It alternates between fixing user features to solve for item features, efficiently minimizing prediction errors. Additionally, ALS maintains its effectiveness as more data becomes available, showcasing its scalability.

2.2 Understanding Alternating Least Squares (ALS) for Collaborative Filtering



In terms of this project the ALS algorithm is utilized to decompose a complex user-artist interaction matrix into two simpler matrices: the user matrix and the artist matrix. This method of decomposition helps in identifying latent factors that may not be immediately apparent in the original matrix. Each entry in the original user-artist matrix represents the interaction such as play counts between a user and an artist. However, not all interactions are known, leading to missing values within the matrix.

Decomposition Process: The process of ALS involves breaking down the user-artist matrix into two lower-dimensional matrices whose product approximates the original one. These matrices are known as the user matrix and the artist matrix. As depicted in the screenshots, the user matrix (U) contains user-specific factors in its rows, while the artist matrix (A) contains artist-specific factors in its columns. Each factor might represent underlying characteristics such as musical genre preference or listening habits.

Matrix Multiplication and Reconstruction: The fundamental premise of ALS is that by multiplying the user matrix and the artist matrix, we can reconstruct the original matrix, filling in the missing entries effectively. This reconstruction is shown in the first screenshot where the multiplication of U and A approximates the original matrix. For instance, if the original matrix shows that a user has listened to certain artists but not others, ALS helps predict how much the user might like the unlistened artists based

on similar user and artist profiles.

Optimizing the Model: ALS alternates between fixing the user matrix and optimizing the artist matrix, and then fixing the artist matrix to optimize the user matrix, hence the name Alternating Least Squares. This iterative process minimizes the prediction error between the actual data and the model's predictions, which is quantified using a loss function typically based on least squares.

Predicting Missing Values: As shown in the second screenshot, to predict a missing value for a user's interaction with an artist, we perform a dot product between the corresponding user vector and artist vector. This step essentially computes the interaction strength or preference score that the user might assign to the artist, based on the learned latent factors.

Utilizing Predictions for Recommendations: Once the model is trained and the missing values are predicted, the next step is straightforward—recommend artists to users based on the highest scores in the reconstructed matrix. This approach ensures that users are recommended artists that align with their discovered preferences, enhancing the personalization of recommendations.

Adaptive Factor Selection: The flexibility of ALS allows for adjusting the number of factors (dimensions of the user and artist matrices), enabling a balance between the accuracy of the reconstructed matrix and computational efficiency. This adaptability is crucial for handling different scales of data and varying complexities in user preferences.

2.3 Setup and Data Import

Then the tutorial move from the theoretical parts to the implementation part of building a music artist recommender using ALS or alternating list squares. To start we need some data. The process begins by setting up the necessary computational environment and importing libraries. For directory management and path operations, the Path module from pathlib is utilized. Data manipulation and analysis are handled by pandas, a Python library that offers extensive functionalities for data processing. The scipy library is integrated for scientific and technical computing, essential for handling sparse matrices commonly used in machine learning algorithms for efficiency in storage and computation. Importantly, the library is employed to facilitate collaborative filtering, particularly using the Alternating Least Squares (ALS) method, well-suited for handling user-item interactions in recommendation systems.

2.4 Data Loading

Accurately loading and processing of user-artist interaction data is very import for this project. This step involves reading a tab-separated values (TSV) file containing user IDs, artist IDs, and their corresponding interaction weights, which represent the strength or frequency of user interactions with specific artists.

```
def load_user_artists(user_artists_file: Path) -> scipy.sparse.csr_matrix:
    user_artists = pd.read_csv(user_artists_file, sep="\t")
    user_artists.set_index(["userID", "artistID"], inplace=True)
    coo = scipy.sparse.coo_matrix(
        (user_artists.weight.astype(float),
         (user_artists.index.get_level_values(0),
          user_artists.index.get_level_values(1)))
    )
    return coo.tocsr()
```

Here, the interaction data is first loaded into a DataFrame and then transformed into a COO (Coordinate List) format. This format is particularly useful for constructing sparse matrices where the matrix elements are predominantly zeros. Subsequently, the matrix is converted into a CSR (Compressed Sparse Row) format, optimizing row access and matrix operations, which are critical for the performance of the ALS algorithm.

2.5 Artist Information Retrieval

Simultaneously, artist information is managed by the ArtistRetriever class, which is designed to fetch artist names based on their unique identifiers. This functionality enhances the interpretability of the recommendation system by converting artist IDs into human-readable names, thus making the output of the system user-friendly.

```
class ArtistRetriever:
    def __init__(self):
        self._artists_df = None

    def load_artists(self, artists_file: Path) -> None:
        artists_df = pd.read_csv(artists_file, sep="\t")
        self._artists_df = artists_df.set_index("id")

    def get_artist_name_from_id(self, artist_id: int) -> str:
        return self._artists_df.loc[artist_id, "name"]
```

Here, the ArtistRetriever class initializes an empty DataFrame that is later populated with artist data loaded from a TSV file. The DataFrame is indexed by artist IDs to facilitate quick retrieval of artist names, critical for displaying results to end-users.

2.6 Model Fitting and Recommendations

With the necessary data structures in place, the next step involves configuring and employing the ALS model for collaborative filtering. The ImplicitRecommender class contains the ALS model provided by the implicit library. This class is responsible for fitting the model to the user-artist matrix and generating artist recommendations for individual users based on their historical interactions.

```python
class ImplicitRecommender:
    """The ImplicitRecommender class computes recommendations for a given user
    using the implicit library.
    Attributes:
        - artist_retriever: an ArtistRetriever instance
        - implicit_model: an implicit model
    """
    def __init__(
        self,
        artist_retriever: ArtistRetriever,
        implicit_model: implicit.recommender_base.RecommenderBase,
    ):
        self.artist_retriever = artist_retriever
        self.implicit_model = implicit_model

    2 usages (1 dynamic)
    def fit(self, user_artists_matrix: scipy.sparse.csr_matrix) -> None:
        """Fit the model to the user artists matrix."""
        self.implicit_model.fit(user_artists_matrix)

    2 usages (1 dynamic)
    def recommend(
        self,
        user_id: int,
        user_artists_matrix: scipy.sparse.csr_matrix,
        n: int = 10,
    ) -> Tuple[List[str], List[float]]:
        """Return the top n recommendations for the given user."""
        artist_ids, scores = self.implicit_model.recommend(
            user_id, user_artists_matrix[n], N=n
        )
        artists = [
            self.artist_retriever.get_artist_name_from_id(artist_id)
            for artist_id in artist_ids
        ]
```

In this section, the ALS model is trained (fitted) with the sparse user-artist matrix to learn the latent features of users and artists. After training, the model can predict the likelihood of a user appreciating a given artist, enabling personalized artist recommendations. This mechanism improves user experience by tailoring music recommendations to individual tastes, leveraging both explicit interaction data and inferred user preferences.

These steps collectively establish a robust framework for a music recommendation system, using machine learning techniques to deliver personalized content to users effectively.

3 Matrics and Results

In the tutorial, the evaluation of the music recommendation system was demonstrated by successfully printing out the top five artists along with their corresponding scores. The result showcased artists such as Fiona Apple with a score of 1.4759, Pet Shop Boys at 1.4608, The Chemical Brothers at 1.2308, Zero 7 at 1.1960, and Robyn at 1.1701. These numbers are really high. However the tutorial did not explain why and the purpose. But it did successfully print out the top five artists and scores. Therefore, I can conclude that I accomplished the tutorial.

```
100%|██████████| 10/10 [00:06<00:00,  1.60it/s]
Fiona Apple: 1.4759516716003418
Pet Shop Boys: 1.4607843160629272
The Chemical Brothers: 1.23080313205719
Zero 7: 1.19602370262146
Robyn: 1.17010498046875

Process finished with exit code 0
```

4 Reflection

After completing this tutorial on collaborative filtering for music recommendation using ALS, I recognize the need for further exploration and development in this area. The tutorial effectively demonstrated the basic setup of a recommendation system but also exposed the complexities of achieving high accuracy and relevance in the recommendations. Looking ahead, it's essential to delve deeper into various recommendation algorithms and consider hybrid models that could offer more personalized suggestions to users. Handling sparse datasets and translating algorithmic predictions into meaningful musical recommendations remains a significant challenge. Additionally, the unclear methodology behind the scoring system highlights a gap in my understanding that needs addressing. My concern lies in whether the current approach can genuinely capture the diverse and dynamic tastes of users, or if more advanced techniques are necessary to improve the system's predictive accuracy. Moving forward, I am both cautious and curious about the necessary steps, recognizing the substantial learning and experimentation required.