Individual Research Project 2
Project Report
Music Recommendation System

Yogesh Kumar (SJQ3QS)
Sadig Amanli (K5JES7)

June 29, 2023

1 Introduction

Systems that allow users to search for their favorite goods are urgently required due to the vast amount of items that are available on the Internet. The recommendation system is one of the most crucial features for customers to avoid this information overload issue. Based on customers' predefined preferences or access histories, the recommendation service will suggest products they might find interesting. The recommendation relates to a variety of decision-making processes, including choosing what to buy, what music to listen to, or what online news to read. "Item" is the general term used to denote what the system recommends to a user. It could be a song, a book, a movie, etc. Generally speaking, recommendation systems are algorithms that make relevant product recommendations to users. They are really critical in some industries as they can generate a huge amount of income when they are efficient or also be a way to stand out significantly from competitors.

In the music industry, recommendation systems are part of a big engine of streaming apps like Spotify, YouTube Music, Deezer, Tidal, and the like. It is a core of huge engines that work by certain recommender algorithms and suggest a single item or a set of items to users based on such predictions. In the digital era, discovering new music that aligns with our preferences and tastes has become increasingly complex. Music recommendation systems have emerged as valuable tools, leveraging advanced algorithms and data analysis to provide personalized recommendations.

The report explores fundamental principles driving music recommendation systems by using content based approach. It also discusses the role of data collection and preprocessing, capturing user behavior and preferences into meaningful recommendations. The content-based approach emphasizes the inherent qualities of the objects. It creates item profiles using attributes of the item like genre, keywords, or metadata. Content-based filtering suggests products that share qualities with the ones the user has expressed interest in by analyzing their preferences based on their interactions with previously consumed items. This method is very helpful when there is little to no user interaction data available.

2 Review Of The Related Works

Several methods have been investigated in recommendation systems to increase the efficacy and accuracy of recommendations. Collaborative filtering, content-based filtering, and hybrid techniques are some of these techniques. In addition, although the content based approach is the primary focus of this project, it is important to note that collaborative filtering and hybrid methods have their own benefits and have been thoroughly researched in the field of recommendation systems.

Collaborative filtering techniques search similarities among users or items, however only archives of users behavior are analyzed. As an example, similar users have mostly the same products in their baskets and similar items are bought by the same customers. This is the most preferred technique in recommendation systems. They based on the assumption, that if two users have the same opinion on the particular item, it is very likely they like similarly other items. The most important advantages of this kind of systems are: high precision, simple implementation, no additional knowledge about a domain or objects. The long list of advantages is supplemented with the following disadvantages: a problem of "cold start" for users and objects and poor scalability.

In recent years, hybrid methods have gained vital attention due to its ability to combine multiple recommendation techniques to improve recommendation accuracy.. Collaboration-based filtering and content-based filtering are used in hybrid systems to address the shortcomings of each technique separately. Hybrid methods seek to offer more precise and varied recommendations by utilizing the complimentary characteristics of both approaches. By taking into account both user behavior and item features in the recommendation process, these strategies frequently outperform single-method approaches.

3 Description Of The Dataset

The dataset "Music Data: 1950 to 2019" includes details about 28,372 songs from 1950 to 2019. It offers a rich variety of thirty-one features, including both categorical and numerical data, to capture the essence of each song in the dataset. Numerical parameters, including song duration, danceability, loudness, acousticness, valence, and others, provide more insight into the musical traits and characteristics of the songs. The dataset also includes categorical features about the genre and the topic of each song. Each song in the dataset may be seen in its entirety thanks to the intersection of these several properties. The set also includes categorical variables that describe the genre and topic of each song, giving important details about the song's musical style and content. Overall, this dataset contains a wealth of details on songs. It provides the basis for constructing a system for making personalized song recommendations for users based on a variety of factors. Before using the dataset for modeling and recommendation purposes, several preparation processes may be necessary to manage missing values, normalize the numerical features, and encode the categorical characteristics.

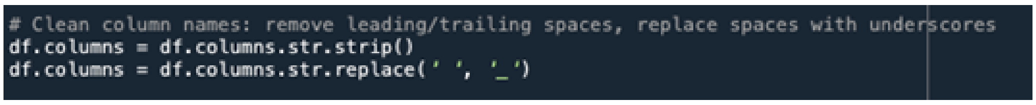
4 Data Preparation Steps

Data preparation is a crucial step in machine learning as the quantity and quality of the data used to train a model directly influence the results. In our case, data preparation steps involve handling missing values, feature scaling, encoding categorical variables. However, some common steps like handling outliers and data splitting might not be relevant because outliers can occasionally be important and useful, and we might want to use all available data to make the best possible predictions. For instance, a song that differs much from others could be quite popular due to its originality; hence, eliminating such outliers would not be advantageous. The step-by-step preparation of our data includes:

1. Importing necessary libraries. For example, Pandas. Pandas is a popular open-source Python library for data analysis and manipulation. "Python data analysis" is what the word "pandas" stands for. It offers efficient, flexible, and expressive data structures that make it simple to manipulate relational or labeled data. The main features of Pandas are reading and writing data in various formats, handling missing data, flexibility in reshaping and pivoting data sets, time series functionality, and high performance for large data sets, etc. In our script, Pandas is used for data preprocessing and manipulation.

2. Loading data using the `pandas.read_csv` function. This function is used to read the '.csv' file and convert it into a Pandas DataFrame.

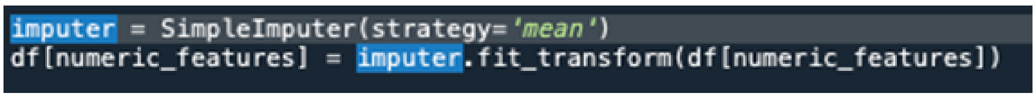
3. Cleaning column names. The next two lines involve removing any leading or trailing spaces from the column names and replacing any remaining spaces with underscores.



```python
# Clean column names: remove leading/trailing spaces, replace spaces with underscores
df.columns = df.columns.str.strip()
df.columns = df.columns.str.replace(' ', '_')
```

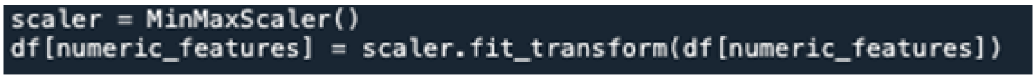
4. Selecting features for the recommendation. We define the list of numerical parameters that will be used later for data imputation and normalization. We handle categorical features, such as genre and topic, using One-Hot Encoding via the `pandas.get_dummies()` function. This function, which is a part of the Pandas library, converts categorical variables into dummy/indicator variables. In our case, it creates a new column for each category value and assigns a 1 or 0 (`True`/`False`) value to the column.

5. Imputing and normalizing numeric features. Firstly, for imputing, we need to import `SimpleImputer` from `sklearn.impute`, which is a class that provides basic strategies for imputing missing values. Imputation is a process of substituting missing data with some statistical measure. In our case, we initialize the `SimpleImputer` with the strategy as `mean`, which means that missing values in each column will be replaced with the mean of the available numbers in that column. After that, we use the `fit_transform()` function to fit the imputer to the data and then transform the data.



```python
imputer = SimpleImputer(strategy='mean')
df[numeric_features] = imputer.fit_transform(df[numeric_features])
```

For normalizing, we first import `MinMaxScaler` from the `sklearn.preprocessing` module. Normalization is a process of scaling numerical data to a range of values, often between 0 and 1, to ensure that no variable has more influence because of its scale. `MinMaxScaler` accomplishes this by subtracting the minimum value in the feature and then dividing by the range of the feature, which is the difference between the maximum and minimum values. Similar to the `SimpleImputer`, the `fit_transform()` function in the `MinMaxScaler` class fits the data and then transforms it in one step. This calculates the minimum and maximum values of each feature, and then applies the transformation to scale the features to the range [0, 1].



```
scaler = MinMaxScaler()
df[numeric_features] = scaler.fit_transform(df[numeric_features])
```

5 Applied Models And Their Hyperparameter Optimization

Content-based techniques apply additional information about users and/or items, as opposed to collaborative methods that rely solely on user-item interactions. In our case, this additional information can include factors such as length, age, genre, or any other relevant information about the music. We will apply content-based methods to create music recommendation systems because user-item interactions are not explicitly specified in our data. The characteristics of the items are often used to determine the level of similarity between them. In simple terms, the content-based strategy utilizes multiple sources of data to provide suggestions.

To apply content-based systems, we need a reliable source of data that contains item characteristics such as length, loudness, etc. Additionally, we require some form of user feedback for the items we are providing recommendations for. In our data frame, we also have ratings.

Firstly, we will use Truncated Singular Value Decomposition (SVD), a dimensionality reduction technique, to convert the data into a format that facilitates the calculation of cosine similarity. SVD can be explained in the context of an $m \times n$ matrix A, where it decomposes the matrix into three other matrices:

$$A = U\Sigma V^T$$

Here, U and V are orthogonal matrices representing the left singular vectors (corresponding to the row space) and the right singular vectors (corresponding to the column space) of A, respectively. Σ is a diagonal matrix containing the singular values. SVD finds applications in fields such as signal processing, statistics, and machine learning. It is frequently used for dimensionality reduction and feature extraction. The decomposition reveals the underlying structure of the data, enabling us to summarize and visualize it.

Truncated SVD is a variant of the SVD method that calculates and returns a specified number of the largest singular values. Unlike regular SVD, Truncated SVD does not require the input matrix to be square, and it produces a low-rank approximation of the matrix. In our case, we choose a value of k such that $k < \min(m, n)$, where k represents the desired number of singular values. By selecting k singular values, we create a rank-k approximation of the original matrix. This reduces the dimensionality of the data, speeds up subsequent computations, and helps remove noise and outliers. Specifically, Truncated SVD transforms the data into a lower-dimensional space of size k. The k new features are uncorrelated and retain as much of the variance in the data as possible. The transformed data can be used for various purposes such as clustering, anomaly detection, and visualization.

After dimensionality reduction, we use cosine similarity to identify the most similar songs for recommendation. Cosine similarity is a metric used to measure the cosine of the angle between two non-zero vectors in a multidimensional space. It provides a measure of how similar the vectors are. Cosine similarity is computed using the following formula:

$$\cos(\alpha) = \frac{A \cdot B}{\|A\| \cdot \|B\|}$$

Where A and B are two vectors, and $\|A\|$ and $\|B\|$ represent the norms of vectors A and B. The resulting value will be between -1 and 1, where 1 indicates identical vectors, 0 indicates orthogonal vectors (i.e., not similar), and -1 indicates diametrically opposed vectors.

In our case, we used cosine similarity as a measure of similarity between different songs. We treated each song as a high-dimensional vector, with its dimensionality defined by the number of features in our dataset (after dimensionality reduction through Truncated SVD). After transforming the dataset using Truncated SVD, we computed a cosine similarity matrix that represented the similarity between every pair of songs in the transformed feature space. Each entry in the matrix, cosine_sim_matrix$[i][j]$, contains the cosine similarity between the i-th song and the j-th song.

When recommending songs, we leveraged this cosine similarity matrix. Given a particular song, you retrieved the cosine similarities between that song and all others, sorted these values in descending order, and picked the top n songs (those with the highest cosine similarity values). This approach ensures that the recommended songs are the most similar to the given song in terms of the selected features.

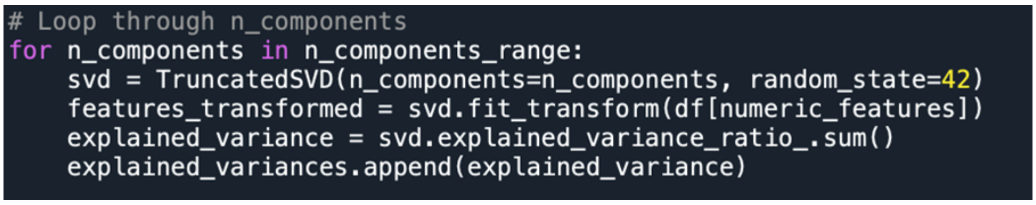
A hyperparameter is a parameter in machine learning that is external to the model and whose value cannot be inferred from the data. They must be predefined, in contrast to model parameters. Hyperparameters may be compared to an algorithm's settings that can be adjusted to improve performance. In our code, the hyperparameter is the *n_components* in the Truncated SVD. *n_components* represents the number of features to keep after the dimensionality reduction process.

To optimize the hyperparameter *n_components*, we used the explained variance as a method. Explained variance is a concept used in Principal Component Analysis (PCA) and Singular Value Decomposition (SVD) to measure how much information (variance) can be attributed to each of the principal components. It is used to understand the distribution of the dataset's original variance across the principal components. In simple terms, explained variance tells us how much a particular principal component contributes to the variance in the data. It's a ratio between the variance of that principal component and the total variance.

When we use PCA or SVD, it's common to sum the explained variances of each component, so we get the total explained variance for n components. This total explained variance indicates how well these n components represent the original data, with higher values indicating better representation. If a small number of components explain most of the variance, this is an indication that the data may be highly dimensional but occupy a smaller subspace. It's often desirable to keep components that explain the majority of the variance while reducing the overall dimensionality.

6 Evaluation Of The Model

Truncated SVD is applied to the processed data, reducing its dimensionality. This technique is efficient and effective in handling large datasets, providing a viable solution for the "curse of dimensionality." The main hyperparameter in this method is *n_components*, indicating the number of dimensions to which the data should be reduced. Selecting the optimal *n_components* is an essential part of the process, which is achieved through a loop iterating over a range of possible values (*n_components_range* = range(5,23)), fitting the TSVD model, and computing the total explained variance for each number of components:

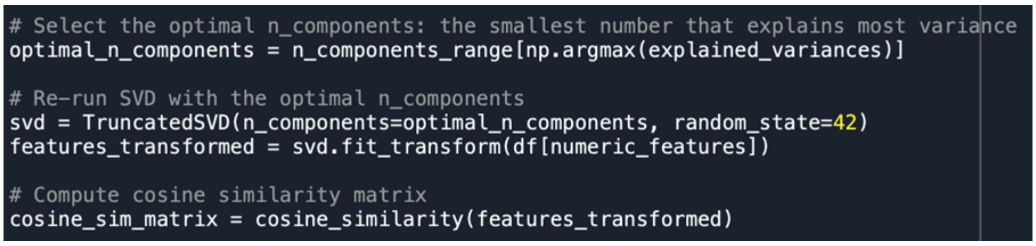


```
# Loop through n_components
for n_components in n_components_range:
    svd = TruncatedSVD(n_components=n_components, random_state=42)
    features_transformed = svd.fit_transform(df[numeric_features])
    explained_variance = svd.explained_variance_ratio_.sum()
    explained_variances.append(explained_variance)
```

The explained variance represents the amount of information retained from the original data in the transformed data. Ideally, it should be as high as possible, so the `'n_components'` value that gives the highest total explained variance is chosen as optimal:

optimal_n_components = n_components_range[np.argmax(explained_variances)].

Once the optimal number of components is determined, the SVD is re-run with this value, and the transformed features are computed. Subsequently, the cosine similarity matrix of the transformed features is calculated using cosine_similarity(features_transformed). This matrix forms the basis for the recommendation process, encapsulating the similarity between every pair of songs in the dataset.

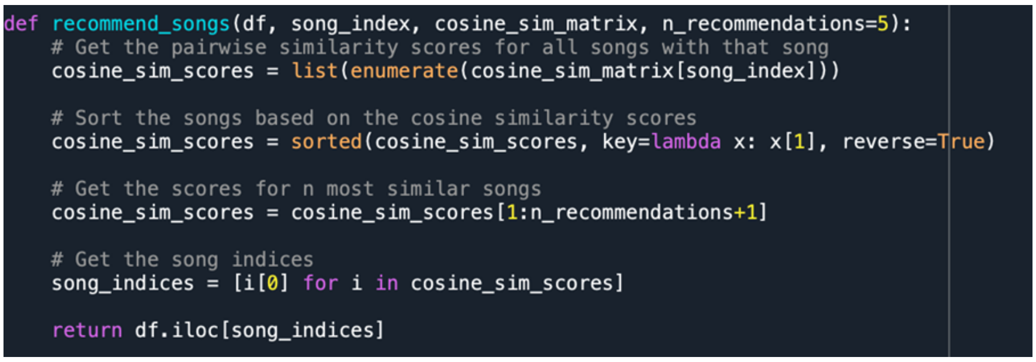


```python
# Select the optimal n_components: the smallest number that explains most variance
optimal_n_components = n_components_range[np.argmax(explained_variances)]

# Re-run SVD with the optimal n_components
svd = TruncatedSVD(n_components=optimal_n_components, random_state=42)
features_transformed = svd.fit_transform(df[numeric_features])

# Compute cosine similarity matrix
cosine_sim_matrix = cosine_similarity(features_transformed)
```

The song recommendation function (‘recommend_songs’) uses this matrix to suggest songs similar to a given input song. It retrieves the cosine similarities between the input song and all others, ranks them in descending order, and selects the top ’n’ songs, where ’n’ is another hyperparameter indicating the number of recommendations to make.



```python
def recommend_songs(df, song_index, cosine_sim_matrix, n_recommendations=5):
    # Get the pairwise similarity scores for all songs with that song
    cosine_sim_scores = list(enumerate(cosine_sim_matrix[song_index]))

    # Sort the songs based on the cosine similarity scores
    cosine_sim_scores = sorted(cosine_sim_scores, key=lambda x: x[1], reverse=True)

    # Get the scores for n most similar songs
    cosine_sim_scores = cosine_sim_scores[1:n_recommendations+1]

    # Get the song indices
    song_indices = [i[0] for i in cosine_sim_scores]

    return df.iloc[song_indices]
```

7 Visualization of Results

Visualizing the results of a machine learning model, such as a recommendation system, is a powerful way to demonstrate the functionality of the system. In this particular case, the visualizations help us understand the positioning of songs within a multidimensional feature space and how they relate to each other based on the selected features. The task is tackled with t-Distributed Stochastic Neighbor Embedding (t-SNE) to project the high-dimensional data onto a 2D plane for visual representation.

t-SNE is a technique for dimensionality reduction that is particularly well-suited for visualizing high-dimensional datasets. The method minimizes the divergence between two distributions: a distribution that measures pairwise similarities of the input objects and a distribution that measures pairwise similarities of the corresponding low-dimensional points in the embedding. It is one of the few techniques that allow preserving both the local and the global structure of the data simultaneously.

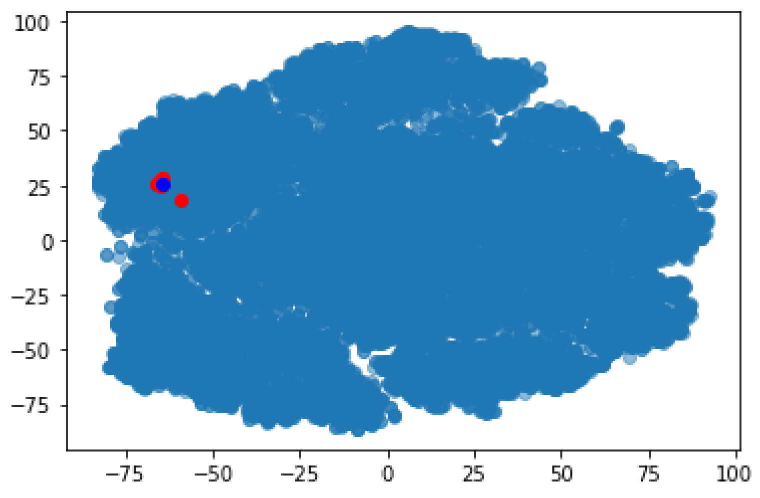
The t-SNE model is created using `tsne = TSNE(n_components=2)`, and it transforms the previously reduced features into a 2D set using `data_2d = tsne.fit_transform(features_transformed)`. The result is a 2D representation of the songs, where each point corresponds to a song, and its position in the plane reflects its similarities to other songs.

This transformation is plotted as a scatter plot using matplotlib, with `plt.scatter(data_2d[:, 0], data_2d[:, 1], alpha=0.5)`. This plot serves as a canvas on which the recommendations and their target song are highlighted.

To facilitate a clearer visualization, the function to recommend songs (`recommend_songs`) is leveraged to highlight the recommendations for a given song on the t-SNE plot. The recommendations’ indices are retrieved using `song_indices = recommendations.index.values`.

A second scatter plot is then overlaid onto the original plot, with points representing the recommended songs highlighted in a different color (`plt.scatter(data_2d[song_indices, 0], data_2d[song_indices, 1], color=’red’)`). This visualization approach allows us to see how close (or far) the recommended songs are from the input song within the t-SNE transformed space, providing an intuitive understanding of the recommendations’ quality.

Finally, the song for which recommendations were generated is highlighted using a third scatter plot, with the point representing the song shown in a distinct color (`plt.scatter(data_2d[song_index, 0], data_2d[song_index, 1], color=’blue’)`). This approach serves as a clear marker for the song of interest and its relationship with the recommended songs.



The blue dot stands for the song we chose, the red dots for the songs the model suggests (those songs have traits in common with the song we chose), and the remaining dots for all the other songs in our dataset. In our example, the blue point and the red point are close to each other. So, in the context of a scatter plot, we can say that the points that are close to each other are more similar than those farther apart. Based on the visualization of the example, we can say that our recommendation system finds similarities.

8 Conclusion

As a result, the recommender system created for music recommendation based on the "Music Data: 1950 to 2019" dataset has shown impressive potential in offering pertinent song choices. The recommendation system showcased here demonstrated how machine learning techniques, including Truncated Singular Value Decomposition (SVD) and cosine similarity, could be employed to reduce dimensionality and establish song similarity. These techniques formed the foundation of the recommendation algorithm, which effectively suggested songs similar to a chosen song. A grid search approach was adopted to optimize the hyperparameters, notably the number of components in SVD, thereby fine-tuning the model's performance.

Further, the report discussed data cleaning and preprocessing steps, such as normalization and imputation, which were critical in preparing the dataset for the machine learning pipeline. Categorical variables were handled using one-hot encoding, transforming them into a format suitable for the machine learning model. A substantial portion of the report was dedicated to discussing the visualization of the recommendation system's results. The t-Distributed Stochastic Neighbor Embedding (t-SNE) technique was employed to create a 2D representation of the high-dimensional song dataset. This visualization offered an insightful perspective on the recommendation system's functionality, demonstrating how similar songs cluster together in the transformed space.

In essence, the construction of this recommendation system signifies a significant stride in the field of music informatics. It reaffirms the potential of machine learning in transforming raw data into meaningful insights, in this case, the connection between songs. The system's potential applications extend from music streaming services to radio stations and can be further explored to improve personalization in music delivery.

9 References

1. Baptiste Rocca (2019). Introduction to Recommender Systems. *Towards Data Science*. Retrieved from `https://towardsdatascience.com/introduction-to-recommender-systems-6c66cf15ada`

2. Vatsal (2021). Introduction to Recommender Systems. *Towards Data Science*. Retrieved from `https://towardsdatascience.com/recommendation-systems-explained-a42fc60591ed`

3. GeeksforGeeks. *Singular Value Decomposition (SVD)*. Retrieved from `https://www.geeksforgeeks.org/singular-value-decomposition-svd/`

4. Laurens van der Maaten (2023) .*t-SNE*. Retrieved from `https://lvdmaaten.github.io/tsne/`

5. *Dimensionality Reduction, Cosine Similarity, t-SNE Visualization, and yperparameter Optimization.* Retrieved from `https://scikit-learn.org/stable/`