Man vs. Machine – Comparing Song Recommendation Systems

Lily Logan, Eliza Black, Kylie Griffiths

March 2025

https://github.com/lilylogan/cs453_project/tree/main

Contents

1 Introduction and Background

1.1 General Background

Recommendation systems are personalized information filtration systems designed to tailor suggestions to each user [2]. These can be constructed in various ways, including content-based filtering [6], which compares liked items with other items in a database according to their attributes, and collaborative filtering [8], which takes into account user preference to identify similar interests. The title of the paper, "Man vs. Machine", pertains to the comparison of a user-driven method versus an attribute-driven method. Both methods yield results unique to each input. When the same input is passed through each method, however, the results may differ based on the approach used.

1.2 Specific Problem

Our study investigates the similarity of recommendations from content-based filtering and collaborative filtering in music recommendations. In the context of our study, content-based filtering analyzes song features, while collaborative filtering recommends songs based on how users with similar song interests have interacted with those songs. We hypothesized that the two models would generate statistically similar recommendations.

2 Motivation and Objective

2.1 Problem Statement

Current implementations of musical recommendation systems often utilize both filtering forms and have years of data developed on their songs and users [10]. Despite this, music-sharing platforms like Spotify and Apple Music often face criticism for their recommendation algorithm. Some of this is simply due to a large population of users, making criticism an inevitability. The general concern of many complaints is that the recommendation systems do not recommend songs appealing to the user.

Herein lies the complicated nature of a recommendation system: the representation of quantitative data. There is no correct answer for a recommended song, given the variability of music taste and the wide range of options. Users can like and dislike songs, but that is not always indicative of their feelings about related songs. Other users may never interact with song options but will listen to certain songs on repeat and skip others. These recommendation systems need to blend the algorithms for content-based and collaborative filtering without losing the information from each.

2.2 Contribution and Novelty

Given the limitations of previous research and our dataset, we decided to separate the filtration systems into two models and compare their results. This is the initial stages of a large-scale recommendation system, where input is captured and used to determine a final recommendation. By separating the two systems, we get a deeper insight into the role each algorithm plays in identifying the music to recommend, as we will be able to directly compare how similar the recommended songs are.

3 Data Collection and Analysis

3.1 Dataset

Our study uses two Spotify datasets: Spotify Songs and Spotify Playlists. Spotify Songs is a dataset that contains nearly 30,000 songs with 23 attributes (including topics like danceability and energy), as well as a dataset of user-created playlists, where each entry includes four attributes: `user id`, `artist name`, `track name`, and `playlist name` [1]. For collaborative filtering, the Songify Playlists database was used. Each row consisted of a user id, playlist, and added song. This database maps users to the songs they added to a playlist [9], and contains of over 10 million user interactions.

3.2 Dataset Introduction and Analysis

Both datasets were downloaded into Python pandas DataFrames (an open source data analysis Python library). Text-based attributes were standardized by stripping spaces and converting them to lowercase. All attribute names were standardized by removing leading and trailing spaces and special characters.

Before diving into making sure that the two datasets have the same songs, a union between the playlist dataset and the song attribute dataset, we conducted a data exploration of the song attribute dataset. We analyzed the numerical attributes by plotting their values to derive more information about their spread, which can be seen in Figure 1. The distributions of `track_popularity`, `danceability`, and `valence` were relatively normal. The distributions of `energy`, `loudness`, `speechiness`, `acousticness`, and `liveness` were relatively skewed left or right. The attribute `key` didn't have a clear distribution while `mode` was bimodal. Keeping these in mind, we moved on to ensuring a union between the two datasets.

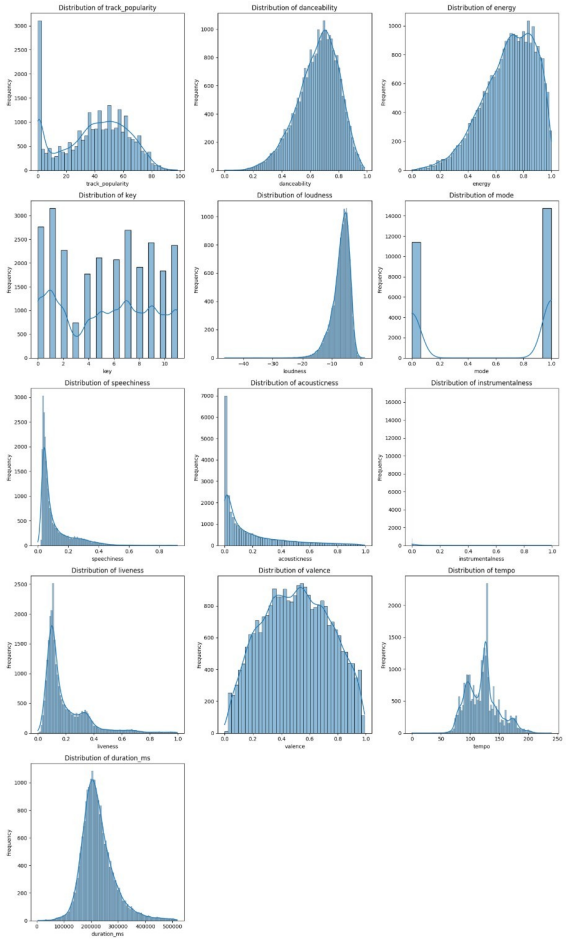

Figure 1: The spread of the song dataset for each of the 13 numerical attributes.

Testing for duplicates in the datasets revealed that the given `track_id` column was not unique and therefore not a reliable input feature for our recommendation system. Therefore, this attribute was removed and a new unique key was created by concatenating the track name and artist name for each song, which we called `track_id`. Duplicate entries based on `track_id` were then dropped, as well as rows with missing data. The modified datasets were matched up, removing songs not present in both datasets. The final datasets contained a total of 8,516 unique songs.

The User-Created Playlists dataset was then prepared for matrix factorization. As discussed in Sec. 4.1, the dataset needed to be converted into a modified binary document-term matrix. In this case, each row value was a unique `user_id`, and each column value was a unique `track_id`. Each entry in the matrix was either a 0, meaning the user had not added that song to a playlist, or a 1, indicating they had.

4 Method

Two primary machine learning algorithms were used to develop the matrix recommendation systems. For the collaborative filtering, this was the matrix factorization and singular value decomposition. For the content-based filtering, the K-Nearest Neighbors algorithm was used.

The outputted playlists from the two recommendation systems were compared using three distinct similarity measures: cosine similarity, Euclidean distance, and Jaccard similarity.

4.1 Matrix Factorization and Singular Value Decomposition

Matrix factorization (or decomposition) is used to decompose matrices into a product of matrices, allowing for more efficient matrix algorithms. Singular value decomposition (SVD) is an eigenvalue-based decomposition that approximates a matrix into three lower-dimensional matrices while saving the maximum variance [3]. Specifically, a truncated SVD was used, which performs more effectively when using sparse matrices.

The User-Created Playlists matrix was decomposed into smaller matrices using a truncated SVD, preserving core components and reducing the dimensionality. An approximate matrix was reconstructed of the original data, using the compressed matrices and reduced components. During experiment and analysis, a new user's playlist would be transformed into the same low-dimensional space using the compressed original matrix and preserved components. We then computed the dot product of this matrix with the core components found in the original to get a list of new song recommendations.

4.2 KNN

K-Nearest Neighbors (KNN) is a method that classifies data by mapping the point using its attributes and classifying the data based on what is nearby. We chose this method because we wanted a way to find data points similar to the input data, which would then be recommendations. Using a KNN allowed us to execute this task.

In order to use this method, the dataset needed to be scaled and encoded as the attributes of the data were both numerical and categorical. Using a standard scaler and one-hot encoder, the data was preprocessed and fitted to the KNN utilizing the `scikit-learn` library. We used Euclidean as the metric of similarity between data points with this KNN.

When getting recommendations for a given playlist, it includes preprocessing the playlist's songs, averaging their attributes, and obtaining the closest k songs [4, 5]. In the preprocessing step, we use the same preprocessor to scale and encode the original dataset. After transforming the data, we average the songs' values to obtain a single aggregated vector. This vector is evaluated by the model which allows us to calculate the nearest neighbors. To ensure that we do not recommend the same song given in the playlist, we calculate `2r` songs, where `r = number of recommendations`. With the

distances and indexes outputted by the `scikit-learn`, we retrieve the recommendations found by using our KNN.

4.3 Comparing Outputs: Distance Metrics

Our two models' outputs were directly compared against each other, so multiple similarity measures were used to identify the potential closeness of these recommendations. Cosine similarity measures the angle between two vectors, meaning that similarity is based on the similarity between the vector orientations rather than their magnitude. Euclidean distance finds the straight-line distance between points and calculates similarity from magnitude. Jaccard similarity, meanwhile, more simply calculated the overlap between sets of elements; if two playlists do not have any overlapping songs, for example, the Jaccard similarity is zero.

5 Experiment and Discussion

5.1 Experiment

After configuring the two models, KNN and matrix factorization, we needed to test how similar the recommendations were between both models. We simulated 121 input and output scenarios by selecting a range of the number of songs, `s`, in the playlist given to the models and the number of recommendations, `r`, outputted by the models. It is important to note that the playlists given to each model were identical to each other. We used the ranges {1, 5, 10, ..., 50}, separated by increments of 5, for both parameters.

5.1.1 Similarity Between Recommendation Systems

The outputs from the matrix factorization and k-nearest neighbor recommendation methods were compared to each other using four similarity metrics: cosine, Euclidean, normalized Euclidean, and Jaccard.

Normalized Euclidean distance was calculated using min-max normalization. Min-max normalization is used to scale the Euclidean distance between two feature vectors so that it falls within the range [0, 1]. This specific normalization process assumes the minimum possible Euclidean distance to be 0 and calculates the minimum and maximum distances between feature vector. The raw Euclidean distance is then calculated using this formula:

$$\text{normalized} = \frac{\text{raw} - \text{min dist}}{\text{max dist} - \text{min dist}}$$

After normalization, 0 means the two feature vectors are identical, while 1 means they are as different as possible within the dataset. Normalization ensures comparability across feature scaling and across similarity measures such as cosine similarity, which is bounded between $[-1, 1]$.

5.2 Discussion

Our results confirm the central hypothesis: KNN and MF generate statistically similar recommendations. While the exact songs recommended by each model greatly differ (low Jaccard similarity), their feature-based similarity is extremely high (cosine similarity ≈ 1.0 and normalized Euclidean distance < 0.1). These statistics, illustrated by Figure 1, indicate that both song recommendation methods recommend songs with nearly identical characteristics.

Stat	Euclidean Dist. (Norm)	Cosine Sim.	Jaccard Sim.
count	121	121	121
mean	0.051469	0.999715	0.001269
std	0.045661	0.000502	0.004897
min	0.006610	0.997129	0.000000
max	0.330024	0.999996	0.029412

Table 1: Statistics for similarity metrics

As a secondary result, the researchers were interested in determining the correlation between computed cosine similarity and Euclidean distance measures to decide whether the similarity measures are interchangeable in the context of this research project or whether they must be individually analyzed.

Between all 121 tests, the Pearson correlation coefficient measuring the linear correlation between normalized Euclidean distance and cosine similarity was -0.43 (the correlation between the similarity metrics is negative because a high cosine similarity corresponds to a low Euclidean distance, as both are measures of high similarity). The corresponding p-value to the correlation measure was 0.0000009574; because this value is much smaller than 0.05, the results rejected the null hypothesis (that cosine similarity and normalized Euclidean distance are not correlated) and determined that the correlation is statistically significant.

As Figure 2 illustrates, we determined that the cosine similarity and normalized Euclidean distance similarity metrics are highly correlated. The correlation between Jaccard similarity and the other similarity metrics, however, is not statistically significant. This result reveals how the inclusion of song attributes in the similarity computations (cosine, Euclidean) yields more nuanced results than a simple set union (Jaccard).

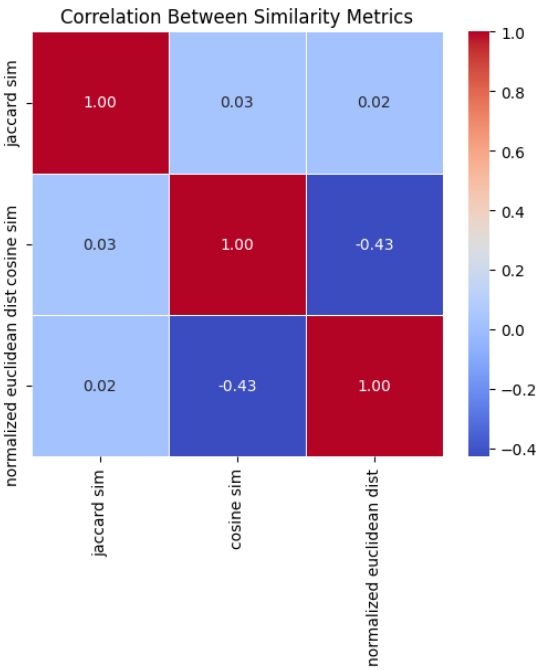

Figure 2: Heatmap of the correlation between similarity metrics

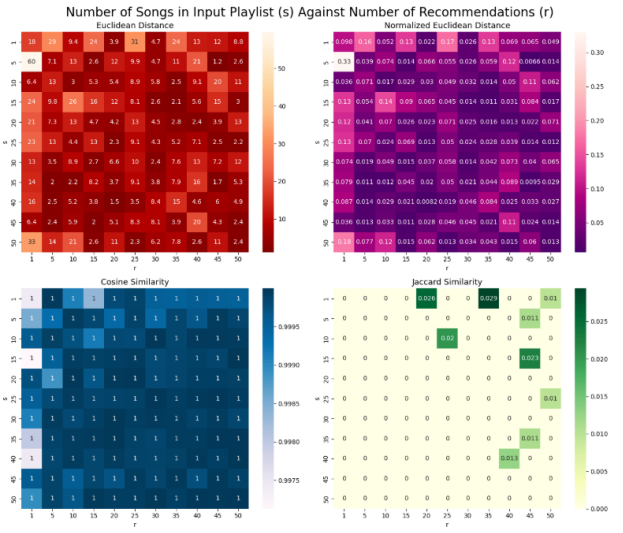

Figure 3: Heatmap of the ranges of `s` and `r` within the 4 similarity metrics.

It is important to consider which similarity metric is best for the comparison of songs. Cosine similarity is particularly useful in recommendation systems, where each item (each song, in this case) can be represented as a vector of features (each song is represented by its danceability, acousticness, and more)[7].

In our initial exploration of the number of songs in the given playlist, `s`, and the number of recommendations requested by the model, `r`, we concluded that they did not influence the similarities between the two models. In the heatmap in Figure 3, we can see different trends with each similarity metric.

While there isn't a strong trend in the Euclidean distance and normalized Euclidean distance heatmaps, we saw higher values when `r` and `s` were both small. This could indicate that when having a small input playlist and asking for fewer recommendations, it is less likely for the two models to produce the same or similar recommendations. In the cosine similarity heatmap, there is a clear indication that when `r` is small, the similarity between the recommendations from each model is not as similar as when using a larger value for `r`. In the Jaccard similarity heatmap, it is obvious that the recommendations from each model are not the same exact songs recommended by the other.

Even though we were unable to draw much from our initial visualizations, there was a slight indication from the cosine similarity metric that when `r` is smaller the models tend to not be as similar in comparison to when `r` is larger. Following this observation, we constructed further visualizations to dive deeper into two of these metrics.

To further illustrate the calculated normalized Euclidean distance, we plotted this metric against the number of songs in the given playlist (`s`) and the output size (`r`), in Figure 4 and Figure 5 respectively. In Figure 4, it is clear that there is a leftward skew indicating that when the number of songs in the input playlist is relatively smaller, the two models recommendations are more different than when using an input size of 10-40 songs. This could be because having fewer songs to base recommendations on makes the differences in the inner-workings of both models more apparent. The bar chart in Figure 5 illustrates a clear leftward skew describing the relationship between the number of recommendations and the similarity of the models. In this visualization, we can explain that when the two models are requested for fewer recommendations, the songs that are recommended tend to be less similar than when requested for larger amounts of recommendations.

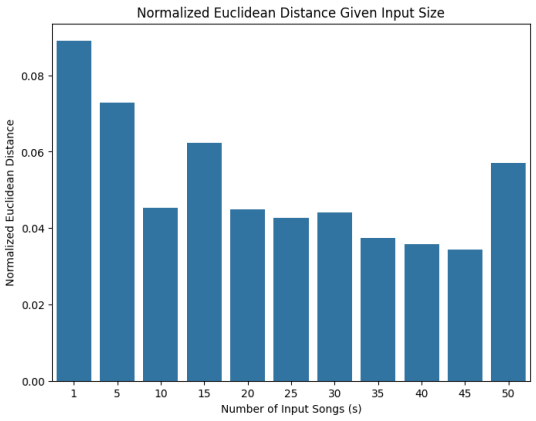

Figure 4: Normalized Euclidean against the given number of songs in input playlist.

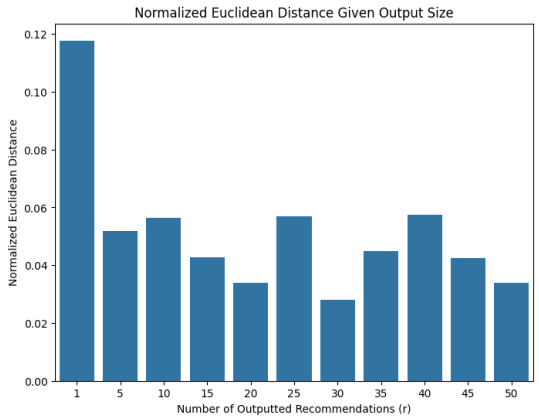

Figure 5: Normalized Euclidean against the number of outputted song recommendations.

To visualize the cosine similarity metric, we utilized box plots to understand the spread in relation to `r` and `s`. In Figure 6, there is a leftward skew in the distribution of the cosine similarity metric and the number of songs in the input playlist. Similar to what our bar chars for normalized Euclidean distance conveyed, this establishes further evidence for the number of input songs correspond to a strong similarity score between the recommendations for each model. With the box plot, the spread indicates more variability with smaller `s` values implying that when the input size of the playlist is small, the cosine similarity metric can be smaller. In other words, the similarity of the two models' recommendations is smaller and the output are relatively different than when `s` is large. In Figure 7 there is a similar distribution. This indicates that the number of requested recommendations correlates to the similarity between the models' outputs. When `r` is small, the cosine similarity value tends to vary but can be relatively small. This means that the similarity between the recommendations (when the requested recommendations is small) can be smaller. The opposite is true when `r` is large.

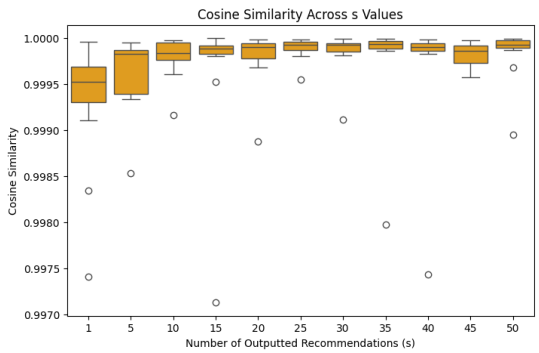

Figure 6: Cosine similarity against the number of songs in input playlist.

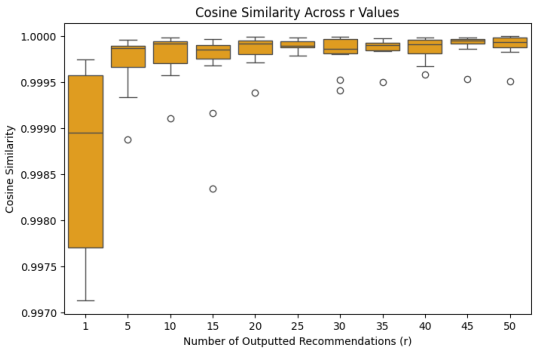

Figure 7: Cosine similarity against the number of outputted song recommendations.

From these visualizations, we were able to conclude that similarity between outputs dramatically decreases for both cosine similarity and Euclidean distance primarily for instances when the input size (`s`) or the output size (`r`) equal one. When `s` and `r` increase beyond a value of one, there is only a very minimal effect of `s` and `r` on output similarity.

5.3 Experimental Concerns

Discussion of this research must contain a commentary on the potential experimental concerns that may have had an impact on the results. It is possible that the results were limited by the data used, the technology available, and the research direction chosen. To further examine these concerns, future research should expand and work to lessen any potential impact.

The final song list used to build the models and produce song recommendations only contained 8,516 unique songs, which is a small sample of the total amount of songs available. It is possible that the high similarity found between the model outputs is a product of having a smaller sample size of songs to choose from, and that dramatically increasing the size of the song list would dramatically increase the complexity of the data, thus reducing similarity. These concerns notwithstanding, the similarity between the two model outputs is still experimentally notable.

Our experiment found only a weak relationship between the input size (`s`), the output size (`r`), and the similarity measures. This is likely due to the limited number of simulations (121), a limitation caused by limited processing power by the researchers' available technology. While this relationship was not identified as overly impactful, the findings are still interesting to highlight independently and to consider in future research.

Lastly, our two unsupervised models were chosen based on the specific needs of the experiment and not chosen explicitly based on performance. To further expand on this, the matrix factorization and K-Nearest Neighbors models were chosen due to the tight time frame and specific research needs (i.e., two working models to compare with multiple input and output sizes). We

were able to implement them within the time and were able to configure the outputs to what we needed, but it's possible that stronger models were passed over due to being too difficult to implement quickly.

5.4 Future Work

This study was limited in terms of execution by the contents of the data and the technology available to the researchers in the given time frame. As such, future work should seek to generally amend this by building larger databases, exploring the relationships identified here more deeply, and considering implementing new technology to bolster the results.

A future development of this study could be a more in-depth exploration of the effect of input playlist size and output playlist size on the similarity between different recommendation systems. To do this, we would obtain larger processing power and run more simulations allowing us to explore the relationships between these parameters while expanding on the range of `s` and `r`. A similar research direction to consider is expanding upon the comparison aspect and building more complex models to act as recommendation systems. Comparing these models would allow for a more in-depth survey of the current technology and how they relate to each other.

Another area to be explored is the relationship between song attributes and the similarity of model recommendations. Even though matrix factorization does not use the attributes directly, it would be interesting to explore if the method still obtains similar attributes using user preferences. To do this, we could analyze how certain attributes impact the similarities between both models. A benefit to this area of research would be additionally databases with more songs or exterior validation of the song attributes by additional sources.

Introducing new technology could be useful for developing future research directions. Large-scale recommendation systems often use neural networks combined with techniques like matrix factorization to identify patterns in each listener. Additionally, having access to larger databases of songs, both attribute-wise and user-interaction based, would be beneficial to the reproducibility and potential impact of this work. This experiment could additionally benefit from seeking out qualitative data, such as seeking user feedback on recommendations from a random population of users. This qualitative data could help researchers develop a stronger hypothesis about the strength of their recommendation system.

6 References

References

[1] Joakim Arvidsson. *30000 Spotify Songs*. 2024. URL: https://www.kaggle.com/datasets/joebeachcapital/30000-spotify-songs/data.

[2] Francesco Casalegno. *Recommender Systems – A Complete Guide to Machine Learning Models*. 2022. URL: https://towardsdatascience.com/recommender-systems-a-complete-guide-to-machine-learning-models-96d3f94ea748/.

[3] Sai Karthik Chedella. *Understanding of Matrix Factorization (MF) and Singular Value Decomposition (SVD)*. 2020. URL: https://medium.com/analytics-vidhya/understanding-of-matrix-factorization-mf-and-singular-value-decomposition-svd-1a38c2d5bbaa.

[4] Rafael Gallo. *ML System Recommendation KNN — Spotify*. 2024. URL: https://www.kaggle.com/code/gallo33henrique/ml-system-recommendation-knn-spotify.

[5] Shivganga Gavhane. *Recommendation System Using KNN and Cosine Similarity*. 2020. URL: https://ijcrt.org/papers/IJCRT2009164.pdf.

[6] Jacob Murel and Eda Kavlakoglu. *What is content-based filtering?* 2024. URL: https://www.ibm.com/think/topics/content-based-filtering.

[7] MyScale. *Unveiling the Power: Cosine Similarity vs Euclidean Distance*. 2024. URL: https://medium.com/@myscale/unveiling-the-power-cosine-similarity-vs-euclidean-distance-43765e8b6da1.

[8] Ashay Pathak. *Recommendation Systems : User-based Collaborative Filtering using N Nearest Neighbors*. 2019. URL: https://medium.com/sfu-cspmp/recommendation-systems-user-based-collaborative-filtering-using-n-nearest-neighbors-bf7361dc24e0.

[9] Martin Pichl, Eva Zangerle, and Günther Specht. *Towards a Context-Aware Music Recommendation Approach: What is Hidden in the Playlist Name?* 2015. URL: https://www.kaggle.com/datasets/andrewmvd/spotify-playlists?select=spotify_dataset.csv.

[10] Yu Sun and Qicheng Liu. *Collaborative filtering recommendation based on K-nearest neighbor and non-negative matrix factorization algorithm*. 2024. URL: https://link.springer.com/article/10.1007/s11227-024-06537-4.