EURECOM

Semester Project

Developing a Large Language Model-Based Playlist Continuation Recommender System

Students:
Enzo CHAROLOIS–PASQUA
Eléa VELLARD

Supervisors:
Youssra REBBOUD
Pasquale LISENA
Raphaël Troncy

January 19, 2026

Contents

Abstract

Platforms such as Spotify, with more than 600 million monthly users [9], and recommendation algorithms driving 3-4% of the industry's revenue [4], highlight the increasingly central role of AI in personalized music streaming experiences. For modern platforms, delivering relevant personalized experiences is a core challenge and they must rely on models that generate high-quality playlists tailored to user preferences.

This project, following the RecSys Challenge 2018 [2], explores an up-to-date strategy for playlist continuation using Large Language Models (LLMs). The Million Playlist Dataset (MPD) is the dataset used here, and the proposed pipeline enhances playlist generation through semantic clustering, embedding-based similarity matching, and fine-tuning of a sentence-transformer model.

Our approach starts with a preprocessing and clustering phase of the playlists based on semantic closeness, using `Sentence-BERT` to generate embeddings and applying the `K-means` algorithm. We then fine-tune a lightweight transformer model, aiming to classify playlists into thematic clusters so that it can better capture semantic relationships between playlists compared to baseline models.

Playlists are generated by evaluating the cosine similarity scores between a given playlist name as input and the existing ones. Finally, a voting system determines the final selection.

The performance of our system has been evaluated through both quantitative and qualitative metrics. The results highlight weaknesses of our fine-tuned model, which fails to outperform baseline models.

1 Introduction

In the era of personalized digital experiences, recommendation systems are key to delivering content customized to individual tastes. Music platforms, like Spotify, rely heavily on these systems, yet creating tailoredd playlists remains a challenging and time-consuming task for users.

Our work focuses on developing a Large Language Model-based playlist continuation recommender system and addressing challenges associated with large-scale music datasets. The Million Playlist Dataset, a widely-used benchmark, offers over a million user-generated playlists.

This report proposes a pipeline for efficient playlist generation using large language models (LLMs). By finetuning a pretrained model to capture semantic relationships between playlists and tracks on pre-computed clusters, we generate new playlists based on input names, supported by quantitative and qualitative evaluations.

Subsequently, we employ a voting-based recommendation strategy for playlist completion and experiment with cutting-edge LLMs, such as Ollama, to compare performance.

This paper is structured as follows: Section 2 presents previous work related to our topic, while Section 3 details the implementation of our approach. Section 4 describes and challenge the experimental results of our method, and we conclude this paper in Section 5.

The implementation of this approach is publicly available at `https://github.com/elea-vellard/LLM-project`.

2 Related work

In the realm of music recommendation systems, the challenge of recommendation systems has been a well-studied problem. Recommendation systems are filtering systems that provide suggestions for items most relevant to a particular user. Previous general work on those systems helped us to get familiar with those types of systems, such as the work of Robin Burke and al., which gives an overview of Recommender Systems [1] (insightful general overview, but unfortunately lacks specific focus on music or modern embedding techniques, as it is a bit old, from 2011). In 2018, the annual RecSys Challenge was centered around the task of playlist completion [2].

Thus, a lot of relevant work has been done on this Subject. Our main source of contribution is the work by Diego Monti, Enrico Palumbo, Giuseppe Rizzo, Pasquale Lisena, Raphaël Troncy, Michael Fell, Elena Cabrio, and Maurizio Morisio (2018), who proposed an ensemble approach combining recurrent neural networks with pre-trained embeddings to predict subsequent tracks in a playlist [5] (effective in 2018, but does not leverage transformer-based embeddings or recent fine-tuning methods, however this was the base of our project and turned out to be very relevant). In addition, we refered to specific papers to understand some approaches, such as the work of Nils Reimers and Iryna Gurevych about sentence-BERT embeddings [6] (providing relevant embedding methods, but does not focus on music recommendation) or the paper "A Closer Look at How Fine-tuning Changes BERT" which provided us with a valuable understanding of a model's fine-tuning process [11] (insightful for BERT fine-tuning but does not show any specific application, such as playlist continuation).

Finally, the work of G. Shani and al. about Evaluating Recommendation Systems was very valuable to understand how to evaluate our systems and identify relevant metrics [7] (highlights useful evaluation metrics but does not focus on the specific topics of music playlist generation or even clustering techniques).

3 Method

3.1 Processing the dataset

The Spotify Million Playlist Dataset is a set of one million user-generated playlists that is commonly used for understanding music preferences and recommendation algorithms. It is available as a set of JSON slices, each slice containing 1000 playlists. Each playlist is composed of the playlist's title, a unique playlist identifier (pid) as well as some additional information we will not be using in this project. Finally, each playlist is composed of a certain number of tracks and artists, that can be identfied by their unique uri. In order to better manage the data, we decided to convert the data into easily-readable CSV files:

- *playlists.csv*: contains the main information of each playlist: its pid, its title, the number of tracks etc.
- *items.csv*: serves as a bridge between tracks and their corresponding playlists. It connects every track (identified by its uri) to its playlist (pid) and writes the track's position within the playlist.

- "tracks": contains information about each track: its unique uri, the title, the artist, the album uri and name, and the duration.

3.2 Clustering the dataset

Our approach is based on identifying semantic common themes between playlists and tracks. By computing embeddings of playlists, we are able to apply a clustering algorithm and create groups of playlists, which we will then use to fine-tune a model.

3.2.1 Embedding model

There are existing models that have already been designed to identify and capture those links to create relevant embeddings of words or sentences.

Initially, we tested basic models such as FastText, but through evaluation, we found that it was not efficient enough for our use case, as it primarily focuses on word-level representations, which did not capture enough the contextual meaning of playlist titles. Thus, the K-Means algorithm (cf. 3.2.2) was failing to cluster our playlists, since their representation were all really similar. Figure 1 shows the tsne representation of 10 clusters, highlighting poor data separation using FastText.

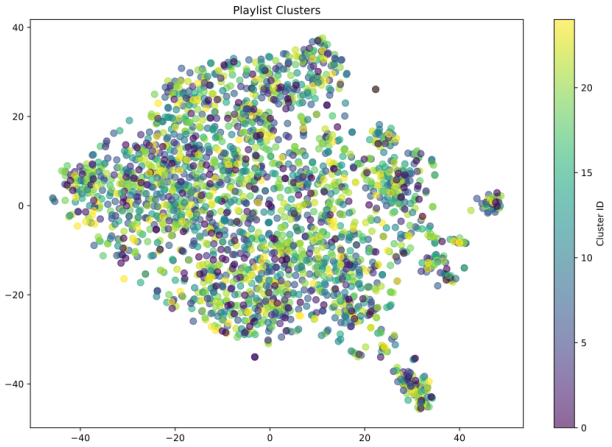

Figure 1: T-SNE plot of the clusters for the training set using FastText

Thus, we narrowed the choice of our model to 2 options: BERT and Sentence-BERT, which is an improvement of BERT. While BERT already considers contextual information using self-attention mechanisms, Sentence-BERT builds on this by leveraging Siamese BERT-Networks, where two BERT blocks process sentence pairs simultaneously, followed by a pooling layer on top to generate fixed-size sentence embeddings [6]. This enables Sentence-BERT to create more effective representations for sentence-level tasks, like our goal of capturing the semantic meaning of playlist titles. Thus, we selected the Sentence-BERT model.

In order to capture a more relevant context, we decided to use a content-based approach: we use Sentence-BERT to embed the mean embedding of all the tracks of each

playlist e to represent it:

$$\mathbf{e}_{\text{playlist}} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{e}_{\text{track}_i}$$

where n is the number of tracks within e.

We found that embedding the tracks rather than the playlists' titles allows to obtain more meaningful clusters. The embeddings are pre-computed and stored in a 'pickle' file as a dictionary whose key is the playlist identifier (pid) and the value is the embedding.

3.2.2 Clustering algorithm

The playlists are then clustered using the K-Means algorithm. Since playlists embeddings remain relatively close, we choose the number of clusters ourselves instead of using density-based algorithms, such as Density-Based Spatial Clustering of Applications with Noise (DBSCAN), that were not able to differentiate the different themes. This is also the reason why metrics like the Silhouette or Davies-Bouldin scores, that calculate inta and inter-distances of each cluster, were not relevant in our specific use case. The clusters are saved in a CSV file, that is composed of the cluster ID, its number of playlists, the title of each playlist as well as the tracks they contain.

Figure 2 shows the T-SNE plot computed over 50 clusters for visualization. It reveals that the clusters are not easily separable. Furthermore, some clusters are significantly larger and more dispersed than others. These larger clusters were later identified as "miscellaneous clusters." (cf. 3.2.2).

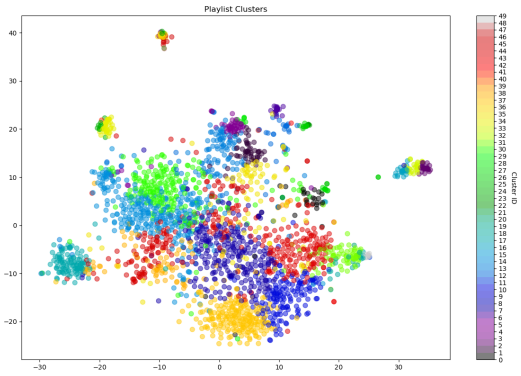

Figure 2: T-SNE plot of the clusters for the training set

To simulate a manual check of clusters and find the most appropriate number of clusters, we computed for each cluster the percentage of exact matches among the playlists' titles. To distinguish "good" and "bad" clusters, we need to look at the coherence of playlists inside clusters, meaning that all playlists within a cluster must be similar, rather than the completeness, which would be to have only one cluster per theme.

However, during training, we encountered issues such as imbalanced clusters, which impacted the classification accuracy. To mitigate this, we used the percentages to identify what we qualify as "miscellaneous" clusters: clusters that contain very big amount of playlists that are not somehow relevant and coherent between them. By looking at the percentages of exact matches, we decided to use 200 clusters to obtain relevant enough

clusters, and we decided to remove clusters whose percentages were lower than 2%, because they were too big and not coherent enough. This action removed around 40% of our whole dataset, which we estimated was low enough for us to cut the line at this threshold, and left us with a total of **158** clusters.

In addition, we implemented an elbow curve approach to estimate an optimal number of clusters for the MPD. However, since our main goal was to ensure coherence within each cluster, the elbow curve's output served more as a guideline to establish a minimum value rather than a definitive choice for the number of clusters. Figure 3 represents the elbow curve and the suggested number of clusters (in red).

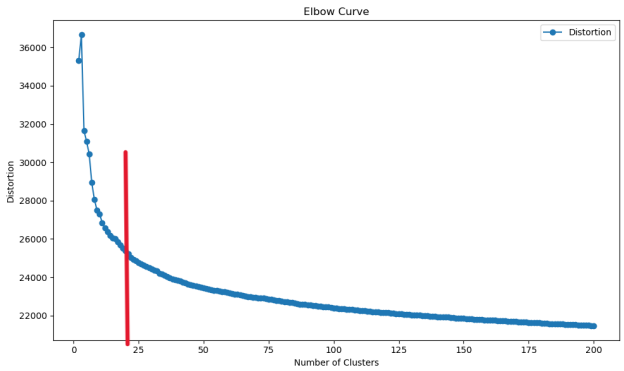

Figure 3: Elbow curve

In fact, while the elbow curve suggested a low number of clusters (~ 25), we ultimately chose 200, which is almost eight times the estimated value. This choice was driven by the idea that having multiple clusters with similar themes was not an obstacle to our goal.

Finally, we split our processed dataset into 3 sets: a training set (80% of the data), a validation set (10%) and a test set (10%). We also ensured a representation of each cluster in every set, so that the model can train and evaluate its performance on specific clusters. The three sets were saved as CSV files.

3.3 Finetuning the model

Fine-tuning a pre-trained model is a critical step in adapting it to the specific needs of a task. In this case, the model should be tailored to capture semantic relationships between playlists and tracks. By fine-tuning a sentence-based model on our clustered playlist data, we ensure the model learns to distinguish and classify playlist themes effectively. The chosen model is "*all-MiniLM-L6-v2*" from the sentence-transformers models' library. It maps sentences to a 384 dimensional dense vector space and can be used for tasks like clustering, while being compact enough to ensure computational efficiency.

3.3.1 Dataset preparation and architecture

The previously clustered playlists serve as the basis for fine-tuning. We feed it using the playlist titles as text and the cluster IDs as labels.

Concerning the architecture, we used the the pre-trained transformer backbone from "*all-MiniLM-L6-v2*" with additional classification layers, composed of a dropout layer for regularization and a fully connected layer to predict cluster labels. Those layers were already included in the *Hugging Face Transformers* library. The model is fine-tuned using the cross-entropy loss function, which is commonly used for classification tasks [3][8].

The fine-tuning process involves training the model on a training set and evaluating it on a validation set to monitor performance and prevent overfitting. We used standard training arguments, including a batch size of 8, a maximum of 5 epochs, an evaluation strategy each epoch, and a learning rate of 2×10^{-5}. We also defined a mapping from original cluster IDs to sequential labels, to ensure compatibility with the classification setup of the model. Throughout the process, we monitor the loss on both the training and validation sets to detect any signs of overfitting. The model is evaluated both quantitatively, with the plot of the loss, and qualitatively, by evaluating its ability to generate coherent recommendations (cf. 4.1).

3.4 Playlist generation

The final step of our pipeline focuses on generating playlists that align with a user's input query. Leveraging the fine-tuned model, we aim to recommend tracks that matches the input theme, without having to be exact matches.

First, for optimization purpose, the embeddings of each playlist titles are computed using the new model and stored in a pickle file for later use. Compared to existing available models, our fine-tuned model is now able to capture semantic relations beyond exact word matches, which allows us to refine the embeddings.

The generation process begins with a user giving the algorithm a name of playlist, that can be composed of one or multiple words, as an input. The fine-tuned model then computes an embedding for the input, capturing its semantic representation. This embedding is compared with the embeddings of all playlists using cosine similarity [10]. The cosine similarity between two embeddings e_a and e_b is defined as:

$$\text{cosine\_similarity}(\mathbf{e}_a, \mathbf{e}_b) = \frac{\mathbf{e}_a \cdot \mathbf{e}_b}{\|\mathbf{e}_a\| \|\mathbf{e}_b\|}$$

The playlists with the highest similarity scores are selected and stored in a list. Then, tracks from the matched playlists are aggregated, and a voting mechanism is applied: tracks that appear frequently across the matched playlists receive higher scores, indicating their relevance to the input theme. The top k most frequently occurring songs are then recommended to the user.

4 Results

4.1 Metrics

To evaluate the effectiveness of the generated playlists, we employed a combination of quantitative and qualitative metrics.

Quantitative evaluation focuses on usual metrics, computed on playlists of the test set.

For each playlist, the set of relevant songs R is defined as the actual tracks contained within that playlist. The set of recommended songs is noted as S. The metrics we focus on are computed for a playlist of N songs.[7]:

- **precision@N**: measures the proportion of recommended tracks that are relevant, with N being the number of recommended songs.

$$precision@N = \frac{len(R \cap S)}{len(S)} \tag{1}$$

- **recall@N**: assesses the proportion of relevant tracks that were successfully recommended.

$$recall@N = \frac{len(R \cap S)}{len(R)} \tag{2}$$

- MRR@N (Mean Reciprocal Rank): Estimates the rank of the first relevant song in the recommended songs.

$$MRR@N = \frac{1}{Rank\ of\ the\ first\ relevant\ song\ in\ the\ recommended\ songs} \tag{3}$$

In addition to this, qualitative evaluation involved human reviews, judging the thematic coherence and overall appeal of generated playlists. For this step, we relied on our familiarity with the songs, and, if necessary, listened to them, to evaluate each track in the playlist and give it an overall qualitative score.

We believe this measure is important because purely quantitative metrics do not always reflect the model's performance for every playlist, especially when user preferences and the actual thematic do not align. Together, these metrics ensured a complete and human-centered evaluation of the system's performance.

4.2 Quantitative results

4.2.1 Model's performance results

The fine-tuning process demonstrated promising results, with the model achieving low loss in classifying playlist clusters both in the training and validation sets. Overfitting has been prevented by training the model on only 5 epochs.

Figure 4 represents the plot of the loss for the training and validation phases through 5 epochs.

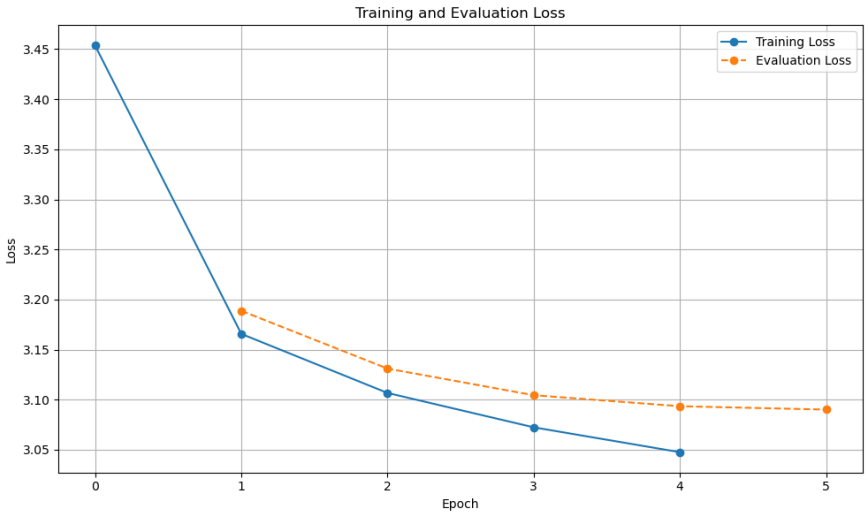

Figure 4: Training and validation loss accross 5 epochs

The figure highlights the model's ability to learn based on the clusters, with no sign of overfitting.

4.2.2 Experiment

Next, to assess the performance of our playlist generation pipeline, we employed several established metrics that evaluate the effectiveness of the system, as described in section 4.1. To evaluate our model and compare it against existing approaches, we decided to test three models:

- Our fine-tuned model;
- A pre-trained Sentence-BERT;
- Llama 3.1: an advanced large language model used here to compare performance, used employing a local instance. For each playlist theme, we provided a prompt of the following form:

  ```
  "Give me the 66 most relevant songs that were released before
  October 2017 for a playlist named input.
  Do a list of songs using the format (song name, artist), without
  any other text formatting."
  ```

  This allowed us to obtain a simple format of recommended songs in order to easily compute quantitative metrics. Beyond this zero-shot approach, we explored the possibility of a multi-shot method to improve results. Specifically, we used ChatGPT 4.0 and provided it with the entire dataset of songs (in a compressed file) so that it would select recommendations directly from our dataset, without us adding bias (e.g., as we would have done if we had provided ChatGPT with a list of all songs from our most similar playlists based on cosine similarity). However, the results showed no significant improvement over the zero-shot approach; consequently, we did not adopt ChatGPT 4.0's multi-shot method for further experiments.

We evaluated each model using the same metrics on playlists from the test set. Due to computational constraints and to maintain consistency across all metrics and models, we limited our evaluation to a set of 22 specifically chosen playlists. Table 1 indicates the selected playlists. Our selection criteria included:

- Playlists with at least 10 tracks
- A clear thematic alignment between playlist titles and their tracks
- A mix of both niche and more broadly themed playlists

PID	Playlist Title	Number of tracks
269808	K-pop	53
999749	Workout music	77
999645	Dance	124
999430	Rock	33
995645	SUMMER	107
1484	hawaii	42
1495	Classic country	53
1537	older songs	30
1628	"2016"	161
1712	dance	89
1720	Finesse	20
1733	oldies	19
1744	Rock	42
642838	Washed Out	27
2354	Christian	83
2524	Gaming	19
2882	classics	32
3559	Party	34
4090	workout	11
4097	WORK	66
4121	Love songs	53
696425	summer	57

Table 1: List of 22 playlists used in our evaluation.

For each model, the generated playlist consists of 66 tracks, corresponding to the average number of songs per playlist across the whole dataset.

In addition, selecting a relatively large number of tracks allowed us to have more perspective for our qualitative metrics and ensured better representation for the quantitative metrics as well.

4.2.3 Models' results

Table 2 shows the average metrics computed over multiple input playlists.

As highlighted in the table, the metrics for the fine-tuned model and the pre-trained model are identical. Indeed, we realized both models were recommending the same songs,

Model	Fine-tuned model	Pre-trained model	llama3.1
Precision@10	0.0586	0.0586	0.0008
Recall@10	0.1295	0.1295	0.0119
MRR@10	0.2189	0.2189	0.0109

Table 2: Comparison of quantitative scores between 3 models.

highlighting a major problem in our pipeline: the fine-tuned model was actually not fine-tuned, which means the model we used was the baseline pre-trained model. However, we could not identify the root cause of this problem, as we can clearly see on figure 4 that the model did learn and that there was no sign of overfitting. Our biggest source of improvement would be to identify that cause in order to properly evaluate our fine-tuned model.

However, we can observe much better results from the pre-trained transformers model than from llama3.1. This could be explained by the established voting system in our pipeline. By selecting the most similar playlists, we narrow the choice of songs to a limited amount, allowing the model to obtain better performance. In addition, sentence-transformers are designed to fulfill specifically the task of sentence-level completion, allowing for more representative understanding of our use case, while llama3.1 was not optimized for classification tasks in the same way.

4.3 Qualitative results

Model	Fine-tuned model	Pre-trained model	llama3.1
Qualitative score	9.33	9.33	8.93

Table 3: Comparison of qualitative scores between 3 models.

These results suggest that quantitative metrics alone do not fully capture model performance. While the quantitative scores may not be perfectly aligned with user-generated playlists, the high qualitative metrics across all three models indicate their ability to create meaningful playlists. It is particularly noticeable for the llama3.1 model, which generates meaningful recommendations that do not match the users' preferences.

5 Conclusion and future work

Our study explored a Large Language Model-based approach for playlist continuation, leveraging semantic clustering, embedding-based similarity research, and fine-tuning techniques. The results indicate that our model effectively learns to capture playlist semantics while training, but fails to differentiate from a pre-trained model, leading to no improved recommendations over baseline approaches.

On one hand, our approach demonstrated good quantitative performance in recommending relevant tracks. The combination of clustering and fine-tuning enhanced the model's ability to generalize to diverse playlists' themes.

On the other hand, some limitations remain. First, our fine-tuned model is not different from a base-line model. Then, the presence of "miscellaneous clusters" indicates a poor clustering approach, which might negatively impact the fine-tuning phase. Additionally, our model's performance was constrained by computational resources, limiting the number of epochs during fine-tuning.

Thus, several improvements can be made to enhance our system. First, identifying the root cause of the fine-tuning issue in order to evaluate an actually fine-tuned model need to be done to understand the efficiency of fine-tuning a model. Linked with this issue, training a larger model, increasing training epochs or refining the fine-tuning parameters could help to obtain more refined embeddings and better performance overall. Finally, a source of improvement could be exploring alternative clustering techniques to address the issue of "miscellaneous clusters" in a more efficient way.

Finally, specific parts of our pipeline could be refined, such as using a different dataset to incorporate user evaluations to provide more insights on our metrics, or optimizing the code, especially in the recommendation part, in order to evaluate our model on the whole dataset instead of a specific number of selected playlist, which we could not do because of time limitations.

6 What we have learned

This project allowed us to gain valuable technical skills in AI and research-oriented aptitudes, such as:

- Managing an entire project on an SSH server with a fully configured environment and linking it with GitHub for efficient code storage (we usually preferred Google Colab, but it lacked computational power).
- Handling JSON files as a database, broadening our previous expertise, which was limited to CSV and SQL formats.
- Understanding the fundamentals of Natural Language Processing (NLP), particularly embeddings and how to use models such as BERT and its variants.
- Fine-tuning LLMs, learning how to tailor them to a specific task with a specific range of data.
- Evaluating recommender models using specific metrics such as `Precision@N` and `Recall@N`.
- Embracing a research-oriented approach for a mid/long-term project that combines both coding and theoretical exploration.

These insights were truly valuable, and we hope they will help us evolve efficiently in a world increasingly driven by AI.

References

[1] Robin Burke, Alexander Felfernig, and Mehmet H Göker. Recommender systems: An overview. *Ai Magazine*, 32(3):13–18, 2011.

[2] Ching-Wei Chen, Paul Lamere, Markus Schedl, and Hamed Zamani. Recsys challenge 2018: Automatic music playlist continuation. In *Proceedings of the 12th ACM Conference on Recommender Systems*, pages 527–528, 2018.

[3] Omar Espejel. Train and fine-tune sentence transformers models. *Hugging Face*, 2022.

[4] Jacca-RouteNote. Spotify's algorithms drives 3-4% of industry revenue. *RouteNote Blog*, 2024.

[5] Diego Monti, Enrico Palumbo, Giuseppe Rizzo, Pasquale Lisena, Raphaël Troncy, Michael Fell, Elena Cabrio, and Maurizio Morisio. An ensemble approach of recurrent neural networks using pre-trained embeddings for playlist completion. In *Proceedings of the ACM Recommender Systems Challenge 2018*, pages 1–6. 2018.

[6] Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China, November 2019. Association for Computational Linguistics.

[7] Guy Shani and Asela Gunawardana. *Evaluating Recommendation Systems*, pages 257–297. Springer US, Boston, MA, 2011.

[8] Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. How to fine-tune bert for text classification? In *Chinese computational linguistics: 18th China national conference, CCL 2019, Kunming, China, October 18–20, 2019, proceedings 18*, pages 194–206. Springer, 2019.

[9] Backlinko Team. Spotify user stats. *Backlinko*, 2024.

[10] Tan Thongtan and Tanasanee Phienthrakul. Sentiment classification using document embeddings trained with cosine similarity. In Fernando Alva-Manchego, Eunsol Choi, and Daniel Khashabi, editors, *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: Student Research Workshop*, pages 407–414, Florence, Italy, July 2019. Association for Computational Linguistics.

[11] Yichu Zhou and Vivek Srikumar. A closer look at how fine-tuning changes BERT. *CoRR*, abs/2106.14282, 2021.

Code and Libraries used

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

Wolf, T. "Huggingface's transformers: State-of-the-art natural language processing." arXiv preprint arXiv:1910.03771 (2019).

Resnik, Philip, and Jimmy Lin. "Evaluation of NLP systems." The handbook of computational linguistics and natural language processing (2010): 271-295.

Hunter, John D. "Matplotlib: A 2D graphics environment." Computing in science & engineering 9.03 (2007): 90-95.

Harris, Charles R., et al. "Array programming with NumPy." Nature 585.7825 (2020): 357-362.

McKinney, Wes. "Data structures for statistical computing in Python." SciPy. Vol. 445. No. 1. 2010.