CS7641 Group 61

Semester project

View on GitHub

Introduction

The exponential growth of digital content has created a significant challenge known as information overload, making it difficult for users to find items that match their personal tastes. To address this, recommender systems have become an essential technology for providing personalized content filtering and suggestion [1]. This challenge is particularly relevant in music streaming, where platforms host tens of millions of songs, making manual discovery almost impractical.

Prior works

To understand the decisions made in this project, we have to look at two main philosophies used for Music Recommedation Systems (MRS): Collaborative Filtering (CF), which leverages community wisdom, and Content-Based Filtering (CBF), which leverages item attributes. However, it was found that pure CBF systems are prone to over-specialization, creating a “filter bubble” where users are never exposed to diverse genres, which limits music discovery [2].

As a result, most of the modern music recommendation systems prefer Collaborative Filtering (CF) through matrix factorization models. The premise of CF is based on a key idea: users who agreed in the past will likely agree in the future. This approach, which dominates commercial applications, effectively captures the “wisdom of the crowd” without needing to analyze the item itself. However, the literature identifies critical shortcomings in pure CF. The most prominent is the cold start problem: the system cannot recommend a new item until it has been rated by a sufficient number of users. In the context of music, where thousands of new tracks are released daily, this is unacceptable. Additionally, matrix factorization models are fundamentally linear. They assume that the complex interaction between a user’s taste and a song’s attributes can be modeled by a simple linear combination of factors. This linearity limits the model’s expressiveness, preventing it from capturing intricate, non-linear patterns in user behavior [3].

The limitations of linear matrix factorization led to the adoption of Deep Learning (DL) in recommender systems. Neural networks introduce non-linearity through activation functions (e.g., ReLU, Sigmoid), allowing the model to approximate any continuous function. The Neural Collaborative Filtering paper demonstrated that replacing the dot product of MF with a Multi-Layer Perceptron (MLP) could significantly improve recommendation accuracy [4].

The consensus in the literature is that Hybrid Recommender Systems (those that combine collaborative and content-based techniques) offer the most robust solution. Hybrid models can leverage content features to bridge the cold start gap while utilizing collaborative data to refine recommendations for popular items.

This project differentiates itself from standard benchmarks (like those using the static Million Song Dataset) by integrating implicit social feedback from **This Is My Jam** with high-level audio features from **Spotify**. While typical studies might use passive scrobble data (Last.fm), TIMJ data represents active curation. Furthermore, rather than relying on raw audio (which is computationally expensive), we utilize Spotify’s psycho-acoustic features (Energy, Valence, Danceability, etc.). This aligns with the trend toward “interpretable” AI, where the input features have semantic meaning. Unlike prior work that feeds latent factors from unsupervised models into downstream networks, our supervised stage instead learns its own user and item embedding spaces directly from IDs, combining them with Spotify features to avoid propagating noise from the unsupervised models and to better handle sparsity.

Datasets

We used two main sources of data for this project:

- Kaggle Spotify Dataset: It provides multiple audio features for songs, such as danceability, energy, acousticness, instrumentalness, and valence.
- This Is My Jam Dataset: It provides both user activity data, which can be used to infer songs that users liked and Spotify URIs that can be used for mapping.

The "This Is My Jam" (TIMJ) Dataset

This Is My Jam was a social music platform active between 2011 and 2015. Unlike streaming services that record every passive listen, TIMJ required users to actively search for and post a single track as their current "jam." This jam would persist on their profile for up to a week. Beyond the primary "jam" action, the dataset also includes over 5.9 million "likes," representing secondary endorsements.

The critical distinction of this dataset is the "cost" of the interaction. A jam is a high-cost signal compared to a stream. It implies a strong, conscious preference. However, this comes at the cost of data volume per user. The median number of jams per user is only 3. This presents a radically different modeling challenge than a streaming dataset where a single user might generate thousands of data points in a month.

Spotify Audio Feature Data

To augment the sparse collaborative data, we utilized the Spotify Kaggle dataset to get audio features for the tracks identified in the TIMJ dataset. Spotify uses proprietary algorithms to analyze the raw audio and extract high-level attributes that describe the song's character. These features allow the system to quantify the content of the music. For example, a "cold start" song with zero jams that has High Energy (0.9) and Low Valence (0.2) can be identified as similar to "Metal" tracks that a user has previously liked, enabling a recommendation based purely on song similarity.

Problem definition

Faced with the vast number of songs in music libraries, it is difficult for people to find ones that suit their tastes. Listening to every song to see if you enjoy them is impossible, so you need to have something that suggests songs you will like and weeds out the ones you won't.

We aimed to develop and evaluate a series of recommendation models to accurately predict user preferences and provide meaningful music recommendations from large-scale, implicit feedback data, and see what effect adding song audio features will have on performance.

Methods

We built a hybrid recommendation system to combine the strengths of collaborative filtering (user behavior) and content-based filtering (audio features), as it was proven to improve the quality of recommendations.

Data Preprocessing Methods

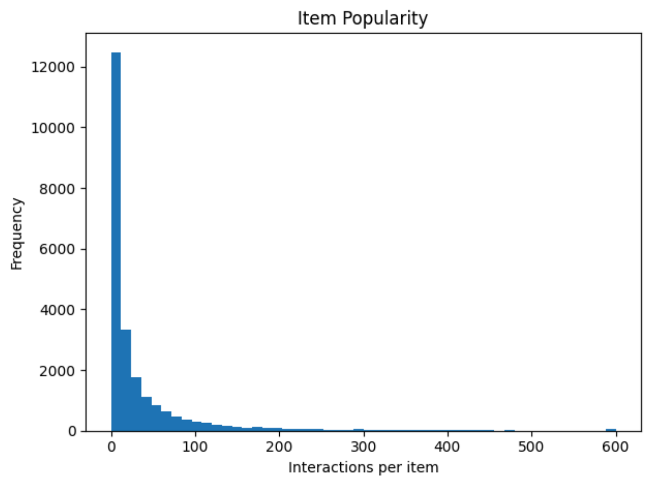
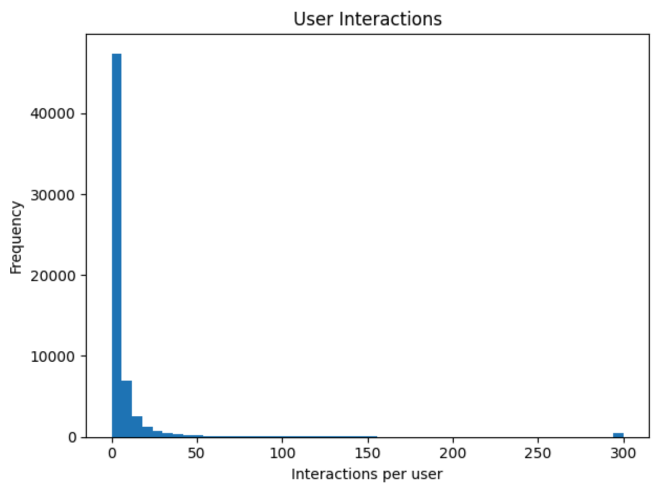
The main technical task was merging the Million Song Dataset with the Kaggle Spotify Dataset to enrich our song features. We employed an identifier-based strategy through Spotify song IDs.

To get the final preprocessed dataset we did the following:

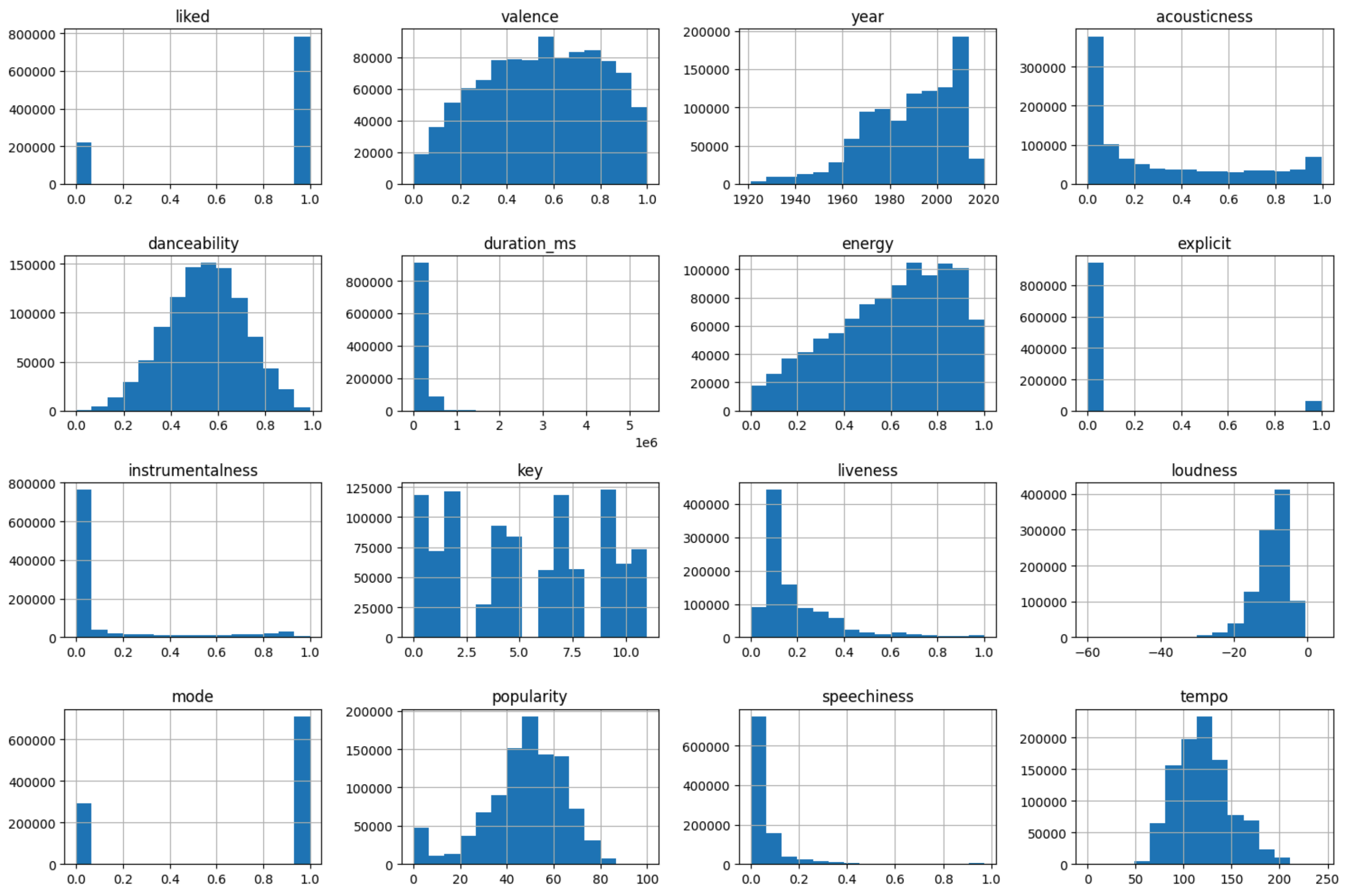
- We filtered the TIMJ data for entries that have a valid `spotify_uri`
- We extracted the Spotify track ID from this URI
- Positive interactions were generated from two sources in the TIMJ data creating a unified list of unique `(user_id, spotify_id, liked=1)` pairs:
    - Creators: The user who created the jam was recorded as having a positive interaction.
    - Likers: Any user who liked the jam was also recorded as having a positive interaction with the same song.

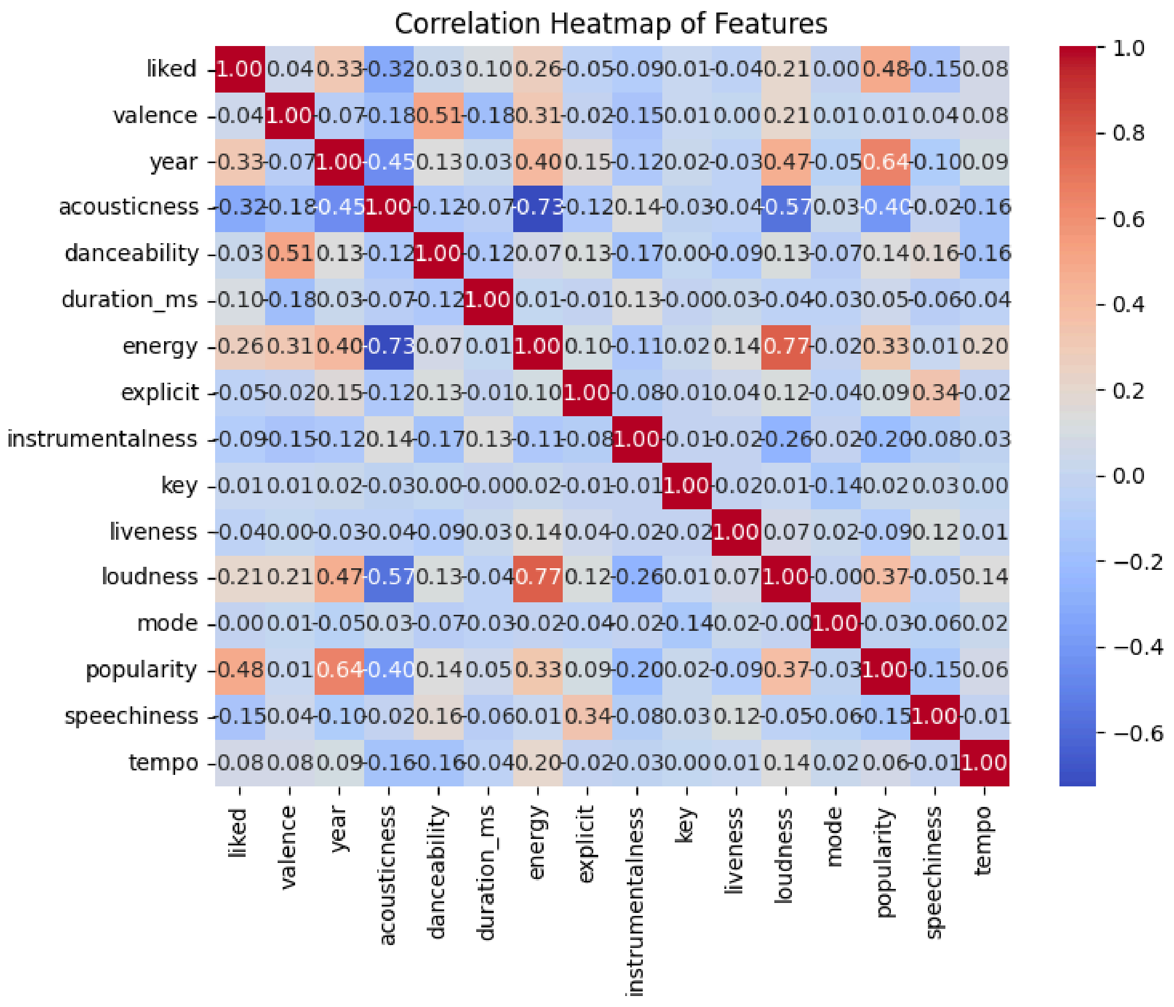
- The extracted TIMJ preferences were merged with the Spotify features dataset on `spotify_id`. This linked all positive user-song interactions to their corresponding audio features. This intermediate dataset (positive interactions only) was saved and used to build the sparse User-Song matrix for unsupervised methods.
- To train our supervised models, we needed negative examples, so for each user, we randomly sampled a small number of songs (capped at 10) from the full Spotify catalog that the user has not interacted with and assigned `liked=0`. This dataset was saved and used for the supervised methods.

The histograms for the Song and User interactions are shown below. It is clear that on average there are very little interactions per user and per song item, which was already discussed above. However, adding the information provided by the song features should improve performance in those cases.

Item Popularity	User Interactions
	

For supervised methods, we needed to implement some feature engineering to reduce noise for the recommender. This involved an exploration of the data which resulted in the following histograms for numerical data distribution, and a correlation heatmap between all features. Noticeable patterns include skew in various features, and certain features contributing significantly to a song being generally liked (e.g. popularity).





Leave k out split method

Since the dataset contains relatively little data for each user, we still wanted to test the average performance of the model's recommendation capabilities for each user. Therefore, we chose the leave k out method. In leave k out, we chose k_eval of liked songs per user to bulid test dataset, k_val of liked sons per user to bulid validate dataset and used all songs left to build train dataset. Because the large number of users, these datasets still have enough samples. The leave-k-out method got a more reliable and unbiased estimate of a model's performance by testing on every possible subset of k samples. It maximized training data in each run, which was especially valuable for small datasets, and ensured every data point is evaluated.

Unsupervised Approaches

- Alternating Least Squares (ALS)

Alternating Least Squares (ALS) is a classic **model-based collaborative filtering** method widely used for large-scale recommendation systems. Unlike traditional SVD, which struggles with sparse implicit-feedback data, ALS is specifically designed to handle missing values by treating unobserved interactions as unknown rather than negative [5]. For the ALS approach, we treated every observed user–item interaction as a positive preference (p_{ui} = 1) and missing interactions as unknown. Following Hu, Koren & Volinsky (2008) [6], we assigned each interaction a confidence weight ($c_{ui} = 1 + \alpha \cdot r_{ui}$), so the model places more emphasis on items a user has interacted with. ALS then learns latent factors for users and items by alternating between two closed-form least-squares updates, avoiding the need for negative sampling or gradient descent. This makes ALS stable and well-suited for sparse implicit-feedback data while still producing meaningful recommendation scores through dot products of user and item factors.

Supervised Approaches

- Supervised Model-Based Cluster Filtering (MBCF)

We used the positive interaction between user and item as positive samples of model input. For the model, we let the model learn hidden 64-dim feature for each user and item. We used dot product between user feature and item feature to measure their match degree. During training, we used BCE loss between model output and expected output (for positive samples, it should be 1). However, during training, we found that if all inputs are positive samples, the model tended to blindly set the output to 1 instead of learning. Thus, we added some negative samples as input and transmitted this model to supervised model.

- Neural Network (NN)

We combined the features we get from the Spotify dataset with the user and song ids from TIMJ. For the model, we used a global Neural Network with a hidden layer with 128 neurons with RELU activation. We trained a single global model that learns across all users using their embedding representations along with audio features. The NN took in user_embedding, item_embedding (vectors in 64 dimensions) and spotify features. These were concatenated and passed through a fully connected network with a 128-dimensional hidden layer and ReLU activation, ending in a single sigmoid output that estimated the probability that a user will like a given song.

By using user and item ID embeddings instead of unsupervised latent factors, the supervised model avoided noise / sparsity patterns from the earlier stage and instead learned a representation optimized directly for the ranking task.

Results and Discussion

Metrics

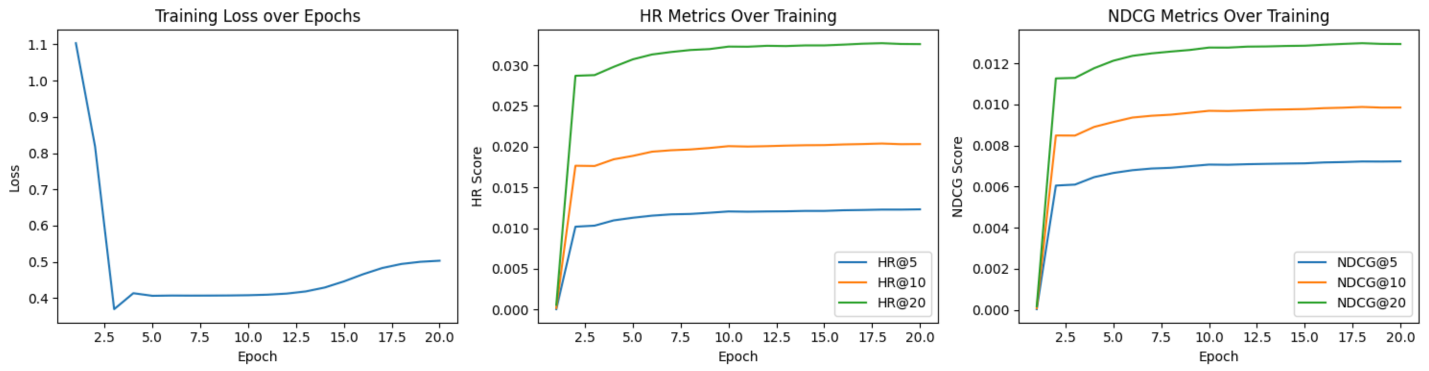
- **HitRate@K**: Simply tells if a good recomendation showed up in the top K recomendation for a user. Shows if a model can find the songs a user will like and recommend them.
- **Mean Reciprocal Rank (MRR)**: A metric, which can show how well the correct recomendation ranks among all the recomendations given by a model.

- **Normalized Discounted Cumulative Gain (NDCG@k)**: Its a better metric for situations where the ranking itself is critical by giving higher scores if correct recommendations are ranked higher. It accurately models user behavior (users pay more attention to the first few results)

Unupervised Models

ALS

Results



As the first figure shows, the ALS model converged very quickly. The training loss dropped sharply in the first few epochs and stabilized soon after, which is expected because ALS updates are solved in closed-form rather than by gradient descent.

In the HR and NDCG plots, we observed a similar pattern: both metrics rose steeply at the beginning and then flattened. HR@20 performed the best, followed by HR@10 and HR@5, indicating that ALS is more effective when the recommendation list is longer. NDCG showed the same trend, with NDCG@20 reaching the highest score and exhibiting stable behavior after early training steps.

Quantitative Metrics

Following metrics are measured under 20 ALS iterations, 40 latent factors, and α = 40 confidence weight.

- HR@5: 0.011
- HR@10: 0.020
- HR@20: 0.032

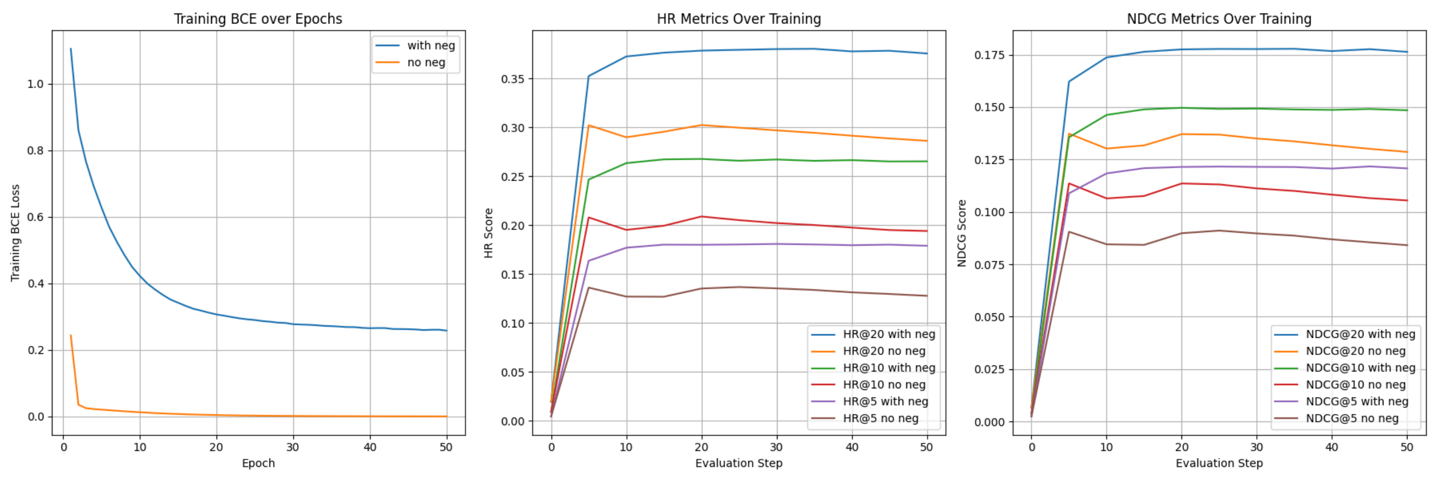
- NDCG@5: 0.007
- NDCG@10: 0.010
- NDCG@20: 0.013

- MRR: 0.005

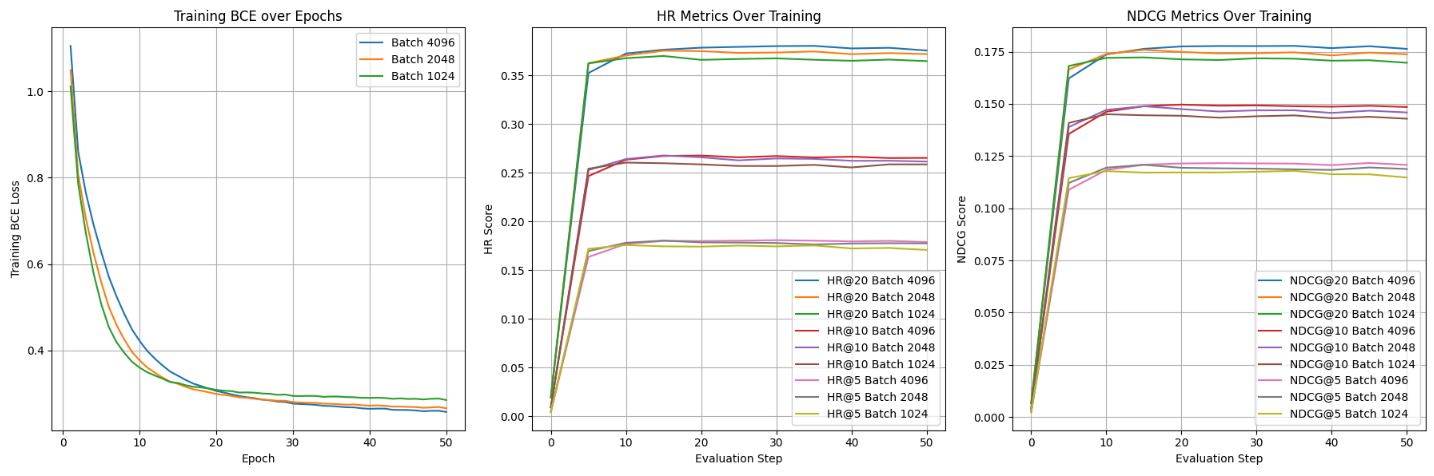
Supervised Model

MBCF

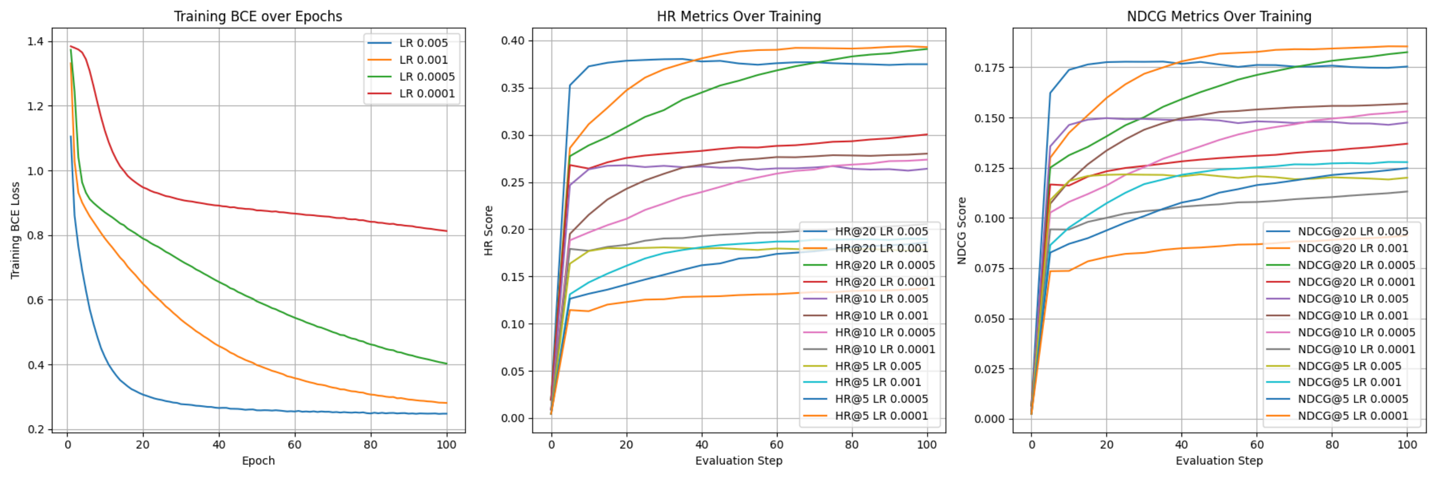
Negative Samples and Hyperparameter Research



As Figure shows, when we added negative samples, the model performed better. In following hyperparameter research, we used training with negative samples as default.



In limited 50 time steps, training with 4096 batch size performed better than 2048 and 1024 batch size.



In limited 100 time steps, training with 1e-3 learning rate performed better than 5e-3, 5e-4 and 1e-4 learning rate.

Quantitative Metrics

Following metrics are measured under 4096 batch size, 1e-3 learning rate and 100 training steps.

- HR@5: 0.190
- HR@10: 0.280
- HR@20: 0.393

- NDCG@5: 0.128
- NDCG@10: 0.157

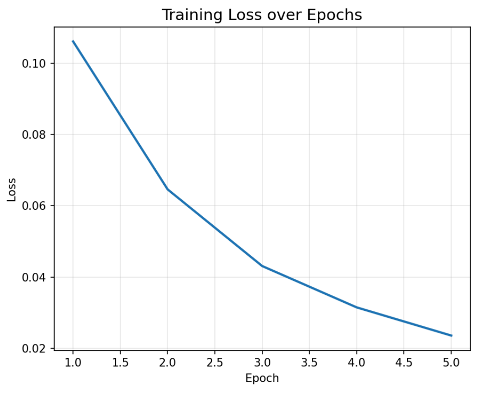
- NDCG@20: 0.185
- MRR: 0.136

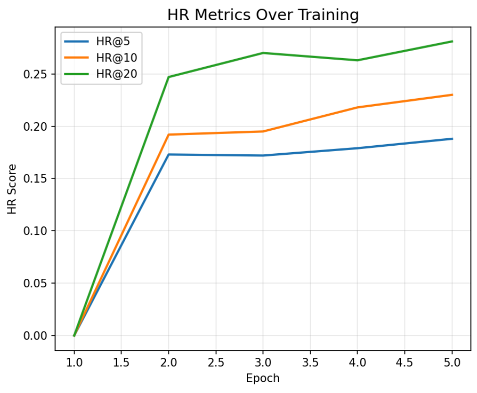
NN

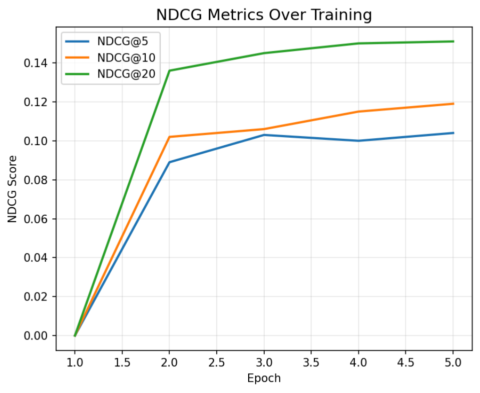
During training, the loss steadily decreased across epochs, stabilizing after the second one. The loss curve below shows this flattening behavior, suggesting the model converged quickly.

Visualizations

We plotted histograms of feature distributions to verify normalization and potential bias. The training loss curve above reveals consistent learning and no sign of divergence, meaning the optimizer was stable. However, even though training looked smooth, the ranking metrics suggest weak generalization. The model can separate positive examples during training, but not rank them high enough when compared to random negatives.







Quantitative Metrics

Evaluating on 100 users produced the following results:

- HR@5: 0.188
- HR@10: 0.230
- HR@20: 0.281

- NDCG@5: 0.104
- NDCG@10: 0.119
- NDCG@20: 0.151

- MRR: 0.098

Other attempts at pipelines

We tried user-specific fine-tuning supervised models on small subsets, even though this was computationally heavy. We ended up achieving 0 hit rate using this method of training separate models per user. It seems as though supervised learning on its own is difficult for music recommendation on this dataset, since the model will generalize according to the audio features and ignore personalization.

Model comparison and discussion

Metrics for all models

Model	**HR@5**	**HR@10**	**HR@20**	**NDCG@5**	**NDCG@10**	**NDCG@20**	**MRR**
Unsupervised ALS	*0.011*	*0.020*	*0.032*	*0.007*	*0.010*	*0.013*	*0.005*

Model	HR@5	HR@10	HR@20	NDCG@5	NDCG@10	NDCG@20	MRR
Supervised MBCF	*0.190*	*0.280*	*0.393*	*0.128*	*0.157*	*0.185*	*0.136*
Supervised NN	*0.188*	*0.232*	*0.281*	*0.104*	*0.119*	*0.151*	*0.098*

Discussion

ALS did not perform as good as the other models likely because the user-item matrix was very sparse due to the nature of our data. Usually recommendation systems would use data based on streams, so every user would have more songs they interacted with and every song would have more users that listened to it. In our case, a jam is a high-cost signal compared to a stream, so there are significantly less entries per user and per item in the dataset. As a result, ALS did not perform as well as the other models due to the sparsity of the matrix.

In our experiments, the ALS model consistently performed worse than the MBCF (Neural Matrix Factorization) model across all metrics, including HR@K and NDCG@K. This performance gap is primarily due to fundamental differences in how the two methods learn from implicit data. ALS, following the classic Hu–Koren–Volinsky formulation, treats all unobserved interactions as “unknown” rather than negative and applies a uniform confidence weighting ($c_{ui} = 1 + \alpha \cdot r_{ui}$). Because our dataset contains only binary implicit feedback (each interaction occurs once), ALS receives almost no variation in confidence signals and therefore struggles to learn strong personalized preferences. In contrast, MBCF explicitly incorporates negative sampling during training, allowing the model to directly contrast positive interactions against sampled non-interactions. This contrastive learning significantly sharpens ranking signals and leads to higher HR and NDCG scores. Moreover, MBCF benefits from nonlinear neural architectures, flexible optimization through SGD, and hyperparameters such as learning rate, batch size, and negative ratio—none of which ALS can exploit due to its closed-form least-squares updates.

As a result, MBCF is better aligned with ranking-based evaluation metrics and more robust to sparsity, leading to noticeably superior recommendation performance compared to ALS in our setting. While the Neural Network outperforms ALS, it is still short of the accuracy obtained by MBCF. A hit rate of 0.23, for instance, means that in about one in every four users, the correct song was somewhere in their top ten recommendations. This suggests that the model learns useful user–item signals, just not at a level of resolution comparable to the contrastive training strategy employed by MBCF. Unlike MBCF, which contrasts positives directly against sampled negatives at training time, the supervision network takes in negative samples that are not necessarily reliable. Some sampled “negatives” may be nothing more than songs a user liked but simply didn’t interact with. This weakens supervision and can make ranking more challenging.

The slightly lower performance can also be attributed to the way the supervised model learns personalization. It learns a single global embedding space for all users and items, so two very disparate listeners may end up with embeddings that are too close together. This effect is caused by compression-a common issue in sparse implicit datasets. In contrast, MBCF benefits from stronger separation between user vectors because the negative sampling objective forces the network to explicitly push embeddings apart. The supervised model also relies on a small set of audio features combined with ID embeddings, giving it a more limited view of item similarity than the richer latent structure learned by MBCF through its nonlinear architecture. Finally, there is the issue of data distribution. The TIMJ dataset is long-tailed: most songs have only a few interactions, and most users have very few jams. This presents a challenge for the supervised model when generalizing to cold users and sparse items, given it cannot rely on repeated patterns of co-occurrence. While the audio features help regularize this sparsity, they cannot fully replace the missing collaborative signal. The rapid flattening of the loss curve suggests it converged to a stable but shallow optimum-one which captures broad trends, such as popularity or high-level audio traits, but struggles to learn fine-grained ranking structure under limited data.

References

[1] G. Adomavicius and A. Tuzhilin, “Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions,” IEEE Transactions on Knowledge and Data Engineering, vol. 17, no. 6, pp. 734-749, Jun.

2005.

[2] D. Paul and S. Kundu, “A survey of music recommendation systems with a proposed music recommendation system,” in Emerging Technology in Modelling and Graphics, Kolkata, India: Springer, 2020, pp. 279–285.

[3] Y. Wang, “A hybrid recommendation for music based on reinforcement learning,” in Proc. 24th Pacific-Asia Conf. Knowl. Discovery and Data Mining (PAKDD), Singapore, May 11–14, 2020, pp. 91–103.

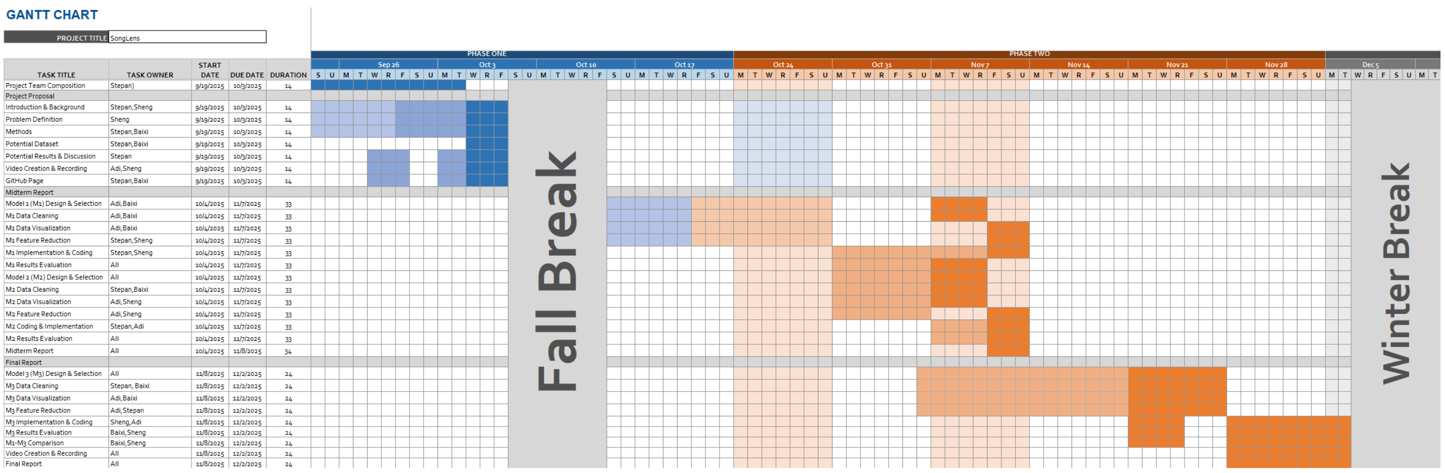
[4] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, “Neural collaborative filtering,” in Proc. 26th Int. Conf. World Wide Web, Perth, Australia, Apr. 2017, pp. 173–182.

[5] Y. Koren, R. Bell, and C. Volinsky, “Matrix factorization techniques for recommender systems,” Computer, pp. 30–37, Aug. 2009.

[6] Y. Hu, Y. Koren and C. Volinsky, “Collaborative Filtering for Implicit Feedback Datasets,” 2008 Eighth IEEE International Conference on Data Mining, Pisa, Italy, 2008, pp. 263-272, doi: 10.1109/ICDM.2008.22.

Project Timeline

Here is the Gantt chart for our project schedule:



GANTT CHART

PROJECT TITLE SongLens

TASK TITLE	TASK OWNER	START DATE	DUE DATE	DURATION
Project Team Composition	Stepan)	9/19/2025	10/3/2025	14
Project Proposal				
Introduction & Background	Stepan,Sheng	9/19/2025	10/3/2025	14
Problem Definition	Sheng	9/19/2025	10/3/2025	14
Methods	Stepan,Baixi	9/19/2025	10/3/2025	14
Potential Dataset	Stepan,Baixi	9/19/2025	10/3/2025	14
Potential Results & Discussion	Stepan	9/19/2025	10/3/2025	14
Video Creation & Recording	Adi,Sheng	9/19/2025	10/3/2025	14
GitHub Page	Stepan,Baixi	9/19/2025	10/3/2025	14
Midterm Report				
Model 1 (M1) Design & Selection	Adi,Baixi	10/4/2025	11/7/2025	33
M1 Data Cleaning	Adi,Baixi	10/4/2025	11/7/2025	33
M1 Data Visualization	Adi,Baixi	10/4/2025	11/7/2025	33
M1 Feature Reduction	Stepan,Sheng	10/4/2025	11/7/2025	33
M1 Implementation & Coding	Stepan,Sheng	10/4/2025	11/7/2025	33
M1 Results Evaluation	All	10/4/2025	11/7/2025	33
Model 2 (M2) Design & Selection	All	10/4/2025	11/7/2025	33
M2 Data Cleaning	Stepan,Baixi	10/4/2025	11/7/2025	33
M2 Data Visualization	Adi,Sheng	10/4/2025	11/7/2025	33
M2 Feature Reduction	Adi,Sheng	10/4/2025	11/7/2025	33
M2 Coding & Implementation	Stepan,Adi	10/4/2025	11/7/2025	33
M2 Results Evaluation	All	10/4/2025	11/7/2025	33
Midterm Report	All	10/4/2025	11/8/2025	34
Final Report				
Model 3 (M3) Design & Selection	All	11/8/2025	12/2/2025	24
M3 Data Cleaning	Stepan, Baixi	11/8/2025	12/2/2025	24
M3 Data Visualization	Adi,Baixi	11/8/2025	12/2/2025	24
M3 Feature Reduction	Adi,Stepan	11/8/2025	12/2/2025	24
M3 Implementation & Coding	Sheng,Adi	11/8/2025	12/2/2025	24
M3 Results Evaluation	Baixi,Sheng	11/8/2025	12/2/2025	24
M1-M3 Comparison	Baixi,Sheng	11/8/2025	12/2/2025	24
Video Creation & Recording	All	11/8/2025	12/2/2025	24
Final Report	All	11/8/2025	12/2/2025	24

Contributions

Name	Final Report Contributions
Aditya Mahesh	Worked on supervised model (hybrid neural network) and setting up the training and testing pipeline to generalize for other models. Performed some feature engineering through exploratory data analysis. Wrote up findings from supervised neural network in the supervised section of the writeup and drew up graphs from EDA. Added graphs for other metrics.
Baixi Jiao	Implemented and analyzed a new unsupervised recommendation models ALS, visualized user–item interaction patterns. Integrating metrics and comparing unsupervised models performances.
Sheng Lu	Introduced leave k out method to divide the dataset into train, validate and test. Implement metrics for measuring model performance. Implemented unsupervised recommendation models (model-based collaborative filtering) and its supervised version with negative samples. Research hyperparameter under which the model performs best. Wrote up the corressponding part and proposed potential improvement method of model.
Stepan Kravtsov	Worked on exploring prior research to refine the models used, metric selection, results and discussion