REPORT

The Million Song Dataset

Margarita Grishechkina
Scalable data, coursework, University of Canterbury, S1 2025

Table of Contents

Background

This work explores music recommendation using collaborative filtering techniques applied to the Million Song Dataset (MSD), focusing on the user-song interaction data from the tasteprofile subset. The goal was to build a scalable recommendation system that can provide personalised song suggestions based on implicit feedback (play counts).

Collaborative filtering models, matrix factorisation methods like Alternating Least Squares (ALS), were used to learn latent features of users and songs. Evaluation was conducted using ranking metrics such as Precision@10, NDCG@10, and MAP, which are more suitable for recommendation tasks than traditional classification metrics.

The dataset, due to its size and sparsity, presented practical challenges such as handling cold-start users, filtering out low-activity users and songs, ensuring compatibility between training and test sets, and optimising for distributed computation in PySpark. Recommendations were validated both quantitatively and qualitatively, and real-world evaluation strategies such as A/B testing and user engagement metrics were discussed.

Preprocessing

The Million Song Dataset (MSD) is organised into four main folders: genre, audio, main, and tasteprofile. The genre folder contains ground-truth labels for classification tasks, the audio folder provides numerical features and their associated attributes, and the tasteprofile folder supports collaborative filtering and recommendation. The main folder includes additional metadata and analysis information for each track and song. A directory tree showing below how the data is organized.

```
msd/
|---- audio/
|       |---- attributes/
|       |      |---- msd-jmir-area-of-moments-all-v1.0.attributes.csv
|       |      |---- msd-jmir-lpc-all-v1.0.attributes.csv
|       |      |---- msd-jmir-methods-of-moments-all-v1.0.attributes.csv
|       |      |---- msd-jmir-mfcc-all-v1.0.attributes.csv
|       |      |---- msd-marsyas-timbral-v1.0.attributes.csv
|       |      |---- msd-mvd-v1.0.attributes.csv
|       |      |---- msd-rh-v1.0.attributes.csv
|       |      |---- msd-rp-v1.0.attributes.csv
|       |      |---- msd-ssd-v1.0.attributes.csv
|       |      |---- msd-trh-v1.0.attributes.csv
|       |      |---- msd-tssd-v1.0.attributes.csv
|       |---- features
|       |      |---- msd-jmir-area-of-moments-all-v1.0.csv
|       |      |---- msd-jmir-lpc-all-v1.0.csv
|       |      |---- msd-jmir-methods-of-moments-all-v1.0.csv
```

```
|       |      |---- msd-jmir-mfcc-all-v1.0.csv
|       |      |---- msd-jmir-spectral-all-all-v1.0.csv
|       |      |---- msd-jmir-spectral-derivatives-all-all-v1.0.csv
|       |      |---- msd-marsyas-timbral-v1.0.csv
|       |      |---- msd-mvd-v1.0.csv
|       |      |---- msd-rh-v1.0.csv
|       |      |---- msd-rp-v1.0.csv
|       |      |---- msd-ssd-v1.0.csv
|       |      |---- msd-trh-v1.0.csv
|       |      |---- msd-tssd-v1.0.csv
|---- genre/
|       |---- msd-MAGD-genreAssignment.tsv
|       |---- msd-MASD-styleAssignment.tsv
|       |---- msd-topMAGD-genreAssignment.tsv
|---- main/
|       |---- analysis.csv.gz
|       |---- metadata.csv.gz
|---- tasteprofile/
|       |---- triplets.tsv
|       |---- mismatches/
|       |      |---- sid_matches_manually_accepted.txt
|       |      |---- sid_mismatches.txt
```

To create a table listing each dataset's name, size, format, data type, and row count, each file was loaded and examined, and its size converted from bytes to megabytes (Table 1).

Table 1, MSD dataset files overview

Name	Size (MB)	Format	Data type	Number of rows
audio/attributes/				
msd-jmir-area-of-moments-all-v1.0.attributes	0.00	csv	string	21
msd-jmir-lpc-all-v1.0.attributes	0.00	csv	string	21
msd-jmir-methods-of-moments-all-v1.0.attributes	0.00	csv	string	11
msd-jmir-mfcc-all-v1.0.attributes	0.00	csv	string	27
msd-jmir-spectral-all-all-v1.0.attributes	0.00	csv	string	17
msd-jmir-spectral-derivatives-all-all-v1.0.attributes	0.00	csv	string	17
msd-marsyas-timbral-v1.0.attributes	0.01	csv	string	125
msd-mvd-v1.0.attributes	0.02	csv	string	421
msd-rh-v1.0.attributes	0.00	csv	string	61
msd-rp-v1.0.attributes	0.05	csv	string	1,441
msd-ssd-v1.0.attributes	0.00	csv	string	169
msd-trh-v1.0.attributes	0.02	csv	string	421
msd-tssd-v1.0.attributes	0.04	csv	string	1,177
audio/feature/				
msd-jmir-area-of-moments-all-v1.0	365.89	csv	double, string	994,623
msd-jmir-lpc-all-v1.0.csv	273.07	csv	double, string	994,623
msd-jmir-methods-of-moments-all-v1.0	150.17	csv	double, string	994,623
msd-jmir-mfcc-all-v1.0	326.02	csv	double, string	994,623
msd-jmir-spectral-all-all-v1.0	223.70	csv	double, string	994,623
msd-jmir-spectral-derivatives-all-all-v1.0	223.70	csv	double, string	994,623
msd-marsyas-timbral-v1.0	1,798.11	csv	double, string	995,001
msd-mvd-v1.0	6,137.63	csv	double, string	994,188
msd-rh-v1.0	881.90	csv	double, string	994,188
msd-rp-v1.0	21,889.38	csv	double, string	994,188
msd-ssd-v1.0	2,467.34	csv	double, string	994,188
msd-trh-v1.0	6,131.06	csv	double, string	994,188
msd-tssd-v1.0	17,315.29	csv	double, string	994,188
genre/				
msd-MAGD-genreAssignment	20.76	tsv	string	422,714
msd-MASD-styleAssignment	14.68	tsv	string	273,936
msd-topMAGD-genreAssignment	19.93	tsv	string	406,427
main/				

analysis	682.67	csv.gz	string, double, integer	1,000,000
metadata	542.49	csv.gz	string, double, integer	1,000,000
tasteprofile/				
triplets	4,569.51	csv	string, integer	48,373,586
sid_matches_manually_accepted	0.10	txt	string	938
sid_mismatches	2.15	txt	string	19,094

While exploring the datasets, I found that the metadata file contains 1,000,000 rows but only 998,963 unique song_ids, indicating a small amount of duplication. Upon inspection, these duplicates arise when a single song appears with multiple associated track_ids. For example, the song_id 418114 occurs 12 times, each linked to a different track_id, possibly due to re-releases, alternate versions, or something else.

In contrast, the triplets.tsv file contains over 48 million rows, reflecting user-song play counts - a key insight for collaborative filtering, meaning that many songs were listened to by multiple users. Similarly, the genre datasets (MAGD, MASD, and topMAGD) contain fewer unique track_ids than the total number of songs, suggesting that not every song has a genre label, and some may be assigned to more than one genre depending on the taxonomy used. This implies overlap or hierarchy in genre classification, which might affect genre distribution and model balance.

After exploration, I defined the schemas for the audio feature datasets using the accompanying attribute files, which list feature names and their data types as describing columns, not actual data. Keeping this information separate from the raw audio features helps reduce redundancy and speeds up loading. Then, I created a StructType schema automatically by mapping each attribute type to its corresponding PySpark type (e.g., float → FloatType()). I then used these mappings to generate a list of StructField objects and wrapped them in a StructType, which I applied when loading audio feature file. This method replaced the default column names (_c0, _c1, …) with informative names like MFCC_Overall_Average_1. I kept the full original names as they were already consistent and descriptive, which improves readability and avoids confusion during analysis or modelling.

Audio similarity

The audio features datasets provide numeric representations of various characteristics extracted from the audio signal, including timbral texture, spectral properties, rhythmic patterns, and other statistical summaries over time, which are precomputed and summarised on a per-track basis, making them well-suited for similarity comparisons or training classification models.

Due to the high dimensionality of the features, a sample of features from different families — area_, lpc_, spectral_, and timbral_ — was summarised to understand their distributions. All features were complete (with nearly 994,000+ entries), with minimal missing values.

Table 2, Descriptive statistics of audio features

Summary	area_2	area_5	lpc_2	lpc_5	spectral_2	spectral_5	timbral_2	timbral_5
count	994623	994604	994623	994623	994623	994623	994622	994622
mean	5500.4633	7.8348	0.17126	0.1429	0.0557	0.0022	0.0691	-45.8930
stddev	2366.1293	1.5826	0.04528	0.0398	0.0265	0.0010	0.03733	5.2715
min	0.0	0.0	0.0	0.0	0.0	0.0	0.0016	-143.1411
max	46860.0	8.124E+10	0.5592	0.4871	0.3739	0.01256	0.823652	0.0

As showing in Table 2, the area_5 feature shows an extremely large range, with a maximum of over 8.12×10^{10} and a very high standard deviation, suggesting the presence of extreme outliers or inconsistent units compared to other features. This may require scaling or transformation (e.g., log) before modelling. In contrast, lpc_ and spectral_ features are tightly distributed between 0 and ~0.5, with relatively low standard deviations. These ranges are consistent and well-bounded, making them model-ready after basic scaling. Feature timbral_5 has a broad range with negative values, going as low as -143.14, indicating it may be a centred or normalised feature (e.g., dB scale). Meanwhile, timbral_2 spans from near 0 up to 0.82, which may reflect magnitude-based measures like spectral envelope strength. These differences highlight that features have varying scales and distributions, and suggest the need for standardisation. As well as that, the large spread and outliers (particularly in area_5) may skew the model if not handled appropriately.

To explore the similarity between features, I computed pairwise correlations across all audio features in the cleaned merged dataset of 994,623 tracks. This analysis revealed 567 feature pairs with strong correlations ($|r| > 0.85$), suggesting potential redundancy in the data. In addition to correlations, I examined the distribution of individual features and observed that many features from the same family exhibited similar distribution shapes, such as normal-like or right-skewed patterns. This was particularly evident in standard deviation measures derived from frequency-based features.

These findings indicate that some features may carry overlapping information. To address this and improve model efficiency, I plan to apply dimensionality reduction techniques (e.g., PCA) or feature selection methods to minimise multicollinearity prior to training the final model. For classification tasks, tree-based models such as Random Forest or Gradient-

Boosted Trees are more robust to correlated features, whereas methods like Logistic Regression may require additional preprocessing to handle feature correlation.

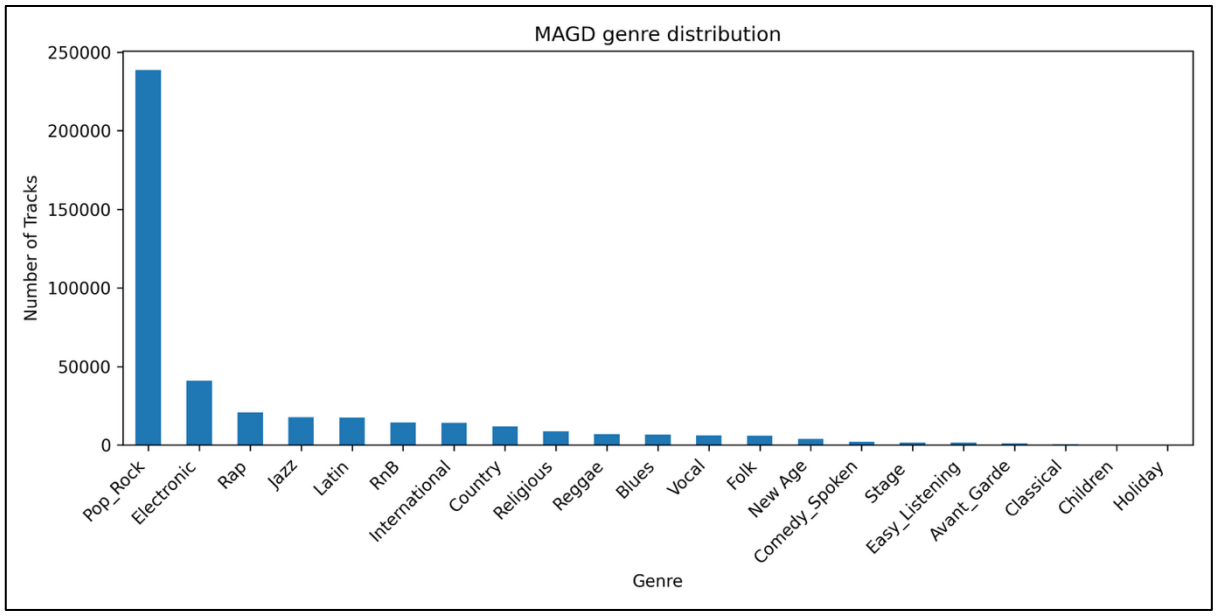

Figure 1, MADS genre distribution

Figure 1 shows the distribution of genres in the MAGD dataset. The distribution is highly imbalanced: the Pop_Rock genre alone accounts for over half of all tracks, while genres like Holiday, Children, and Classical are extremely underrepresented. This skewed distribution can negatively affect the performance of classification models by biasing them toward predicting the majority genres. For example, a naive classifier might achieve high accuracy simply by always predicting Pop_Rock, but it would perform poorly for minority genres. To address this, I plan to apply strategies such as class weighting to penalise misclassification of minority genres. stratified sampling to ensure balanced validation folds, evaluation using precision, recall, and F1-score per genre rather than just overall accuracy. These adjustments are important for building a fair and generalisable classifier across all genre categories.

For the genre classification task, only the area_ feature group was used. This decision was guided by both instructional design and practical considerations. Using a single audio feature family simplifies the model comparison and evaluation process, allows for faster training, and provides more interpretable results. Additionally, it avoids issues related to multicollinearity and overfitting that could arise when training on the full, high-dimensional merged feature set. The full feature combination was reserved for the audio similarity task, where more expressive representations are beneficial.

Three classification algorithms were tested for predicting the genre of audio tracks based on extracted audio features: Logistic Regression, Random Forest Classifier, and Gradient

Boosted Trees (GBT). These methods represent both linear and tree-based approaches, each with different strengths and limitations. Comparing their characteristics in advance helps inform model selection and anticipate the need for preprocessing or tuning.

Logistic Regression is a linear model that performs well on well-scaled and relatively low-dimensional data. It is highly interpretable but assumes linear relationships between features and the target, which may not capture the complex structure of audio data. It also requires feature standardisation and may suffer from multicollinearity, so strongly correlated features were removed beforehand.

Random Forest by building multiple decision trees can handle non-linear relationships and does not require standardisation or correlation filtering. It is robust and can rank feature importance but is slower to train and less interpretable than linear models.

Gradient Boosted Trees (GBT) also use an ensemble of trees but train them sequentially to correct errors of previous trees. GBTs generally achieve the highest accuracy but are sensitive to hyperparameters and require more training time. Like Random Forest, they are robust to scaling and correlation.

Table 3, Models summary

Algorithm	Interpretability	Accuracy	Speed	Needs Scaling	Handles Correlation	Tuning Required
Logistic Regression	High	Moderate	Fast	Yes	No	Low
Random Forest	Moderate	High	Moderate	No	Yes	Moderate
GBT	Low	Very High	Slow	No	Yes	High

To perform classifications the dataset was split into 80% training and 20% test sets. Stratified sampling was used to preserve the class distribution in both sets, ensuring that rare genres are represented in both training and evaluation. Stratification is especially important given the imbalance in genre labels, as simple random sampling could exclude minor classes from the test set entirely, distorting evaluation.

No resampling was applied at this stage to avoid introducing noise into high-dimensional feature space. Instead, model choice and evaluation metrics were selected to handle imbalance effectively. Confusion matrices were used to examine model performance per genre and identify classes that were poorly predicted. In future improvements, metrics like macro-F1 or AUC-PR may also be useful for imbalanced classification.

To simplify the classification task and reduce class imbalance, only the most common genre was selected for binary classification. This choice encourage a sufficient number of samples

for each class, improves model stability, and helps avoid issues caused by underrepresented genres. It also allows for clearer interpretation and model comparison before extending to multi-class classification.

The binary labels are moderately imbalanced, with 237,641 tracks assigned to class 1 and 182,963 to class 0, corresponding to approximately 56.5% of samples in class 1 and 43.5% in class 0. Although this imbalance is not extreme, it may still influence classifier performance—particularly for algorithms that are sensitive to class frequency, such as Logistic Regression.

While accuracy is included for reference, the comparisons focus on models' AUC, precision, and recall, as these give a more meaningful picture of how well each model handles both classes. Class imbalance can skew accuracy—always predicting the majority class (56%) ignores the minority. Precision and recall are more informative: precision measures correct positive predictions, and recall measures recovered positives. The F1-score balances precision and recall, which is crucial when both false positives and negatives matter while AUC offers a threshold-independent evaluation, making it robust to imbalance.

Given the class imbalance and the goal of maintaining performance across both classes, the weighted F1-score is the most appropriate metric for comparing models.

Table 4, Performance metrics comparison

Model	AUC	Accuracy	Precision	Recall
Logistic Regression	0.6935	0.6389	0.6366	0.6389
Random Forest	0.7058	0.6545	0.6509	0.6545
GBT	0.7316	0.6662	0.6661	0.6662

Among the tested models for binary classification, Gradient Boosted Trees (GBT) achieved the highest values across all metrics, suggesting it captured complex patterns better than the simpler Logistic Regression and the more interpretable Random Forest model.

Next, multiclass classification was performed using Logistic Regression. It supports multiclass classification through both one-vs-rest and multinomial (softmax) strategies. Most modern libraries, including scikit-learn, R's nnet::multinom, and PyTorch, offer configuration options for both approaches. The multinomial strategy models all classes jointly and is particularly suitable when class relationships matter or when well-calibrated probability estimates are required. To perform multiclass classification, the genre column was converted into integer labels using consistent encoding, making sure that each genre class was mapped to a unique numeric label, preserving reproducibility and compatibility with classification algorithms. As a result, class balance changed significantly compared to the binary setup - the multiclass task revealed a highly imbalanced distribution. For example,

genres like Pop_Rock were heavily represented, while others such as Punk, Avant_Garde, or Latin appeared much less frequently. This skew can bias classifiers toward over-represented classes and reduce performance on under-represented ones.

The dataset was split into training and test sets, using an 80/20 ratio. Stratified sampling was applied to preserve class proportions across the sets, reflecting the original class distribution, which is important for fair performance evaluation. A Logistic Regression classifier was trained using the multinomial strategy, using weighted metrics that account for class imbalance, including:

⇒ weighted precision and recall, to reflect average performance across all classes
⇒ weighted F1 score, as a balance between precision and recall
⇒ macro F1 score, to check how the classifier treats all classes equally regardless of size
⇒ accuracy, though less informative with imbalance, was also reported for completeness

The multiclass Logistic Regression model was evaluated on a test set of 83,700 tracks using precision, recall, and F1-score for each genre as shown in tables below.

Table 5, Performance metrics for multiclass classification

Metric Type	Score
Accuracy	0.56
Macro Avg F1	0.04
Weighted F1	0.41

Table 6, Multiclass classification by Logistic Regression

Genre	Precision	Recall	F1-Score	Support
Pop_Rock	0.57	0.99	0.72	47,195
Electronic	0.40	0.01	0.01	8,178
Rap	0.08	0.00	0.01	4,198
Jazz	0.25	0.01	0.02	3,492
Latin	0.00	0.00	0.00	3,452
RnB	0.03	0.00	0.00	2,894
International	0.00	0.00	0.00	2,844
Country	0.00	0.00	0.00	2,295
Religious	0.00	0.00	0.00	1,664

Reggae	0.00	0.00	0.00	1,437
Blues	0.00	0.00	0.00	1,368
Vocal	0.00	0.00	0.00	1,246
Folk	0.00	0.00	0.00	1,148
New Age	0.17	0.04	0.06	793
Comedy_Spoken	0.00	0.00	0.00	392
Stage	0.00	0.00	0.00	330
Easy_Listening	0.00	0.00	0.00	300
Avant_Garde	0.00	0.00	0.00	208
Classical	0.00	0.00	0.00	113
Children	0.00	0.00	0.00	93
Holiday	0.00	0.00	0.00	60

Table 7, Average performance metric for multiclass classification

Metric Type	Score	Support
Accuracy	0.56	83,700
Macro Avg	0.07 (P) / 0.05 (R) / 0.04 (F1)	83,700
Weighted Avg	0.38 (P) / 0.56 (R) / 0.41 (F1)	83,700

The classifier achieved 56% overall accuracy, primarily driven by correctly predicting the dominant genre Pop_Rock. However, the performance across other genres was poor, as indicated by the low macro-averaged F1 score (0.04). This is due to a severe class imbalance, where many classes have very few samples compared to Pop_Rock. Several genres had zero predicted samples, leading to undefined precision and recall and highlighting a critical limitation of the model in multiclass settings with imbalanced data.

Including multiple genres reduced model performance. While binary classification yielded good F1 scores (~0.66), the multiclass setup caused accuracy to drop to ~56% and macro F1 to ~0.04–0.07. The model overpredicts the majority class (PopRock), while minority genres are poorly classified due to imbalance. This highlights the need for resampling or class weighting. These mitigation strategies can be incorporated during model configuration and hyperparameter tuning.

Table 8, Hyperparameters summary

Algorithm	Hyperparameter	Description	Effect on Model
Logistic Regression	regParam	Regularisation strength (L2 penalty)	Prevents overfitting by shrinking coefficients; higher values increase regularisation
elasticNetParam	Mix of L1 and L2 regularisation (0 = L2, 1 = L1)	Adds sparsity (feature selection) when closer to 1; improves generalisation	
maxIter	Maximum number of iterations	Affects convergence speed and model accuracy	
tol	Convergence tolerance	Affects when optimisation stops; too small may lead to long training time	
Random Forest	numTrees	Number of decision trees in the ensemble	More trees generally improve accuracy but increase training time
maxDepth	Maximum depth of each tree	Controls overfitting; deeper trees may capture more complexity but risk overfitting	
maxBins	Number of bins for continuous features	Affects how numeric features are split; higher values allow finer splits	
minInstancesPerNode	Minimum samples required to split a node	Higher values reduce overfitting but may underfit	
Gradient Boosted Trees	maxIter	Number of boosting iterations (trees)	More iterations typically improve accuracy but increase risk of overfitting
stepSize	Learning rate	Controls how much each tree influences the final model; smaller values improve generalisation but need more trees	
maxDepth	Maximum depth of each tree	Deeper trees model more interactions; risk of overfitting if too deep	

Cross-validation is a method to estimate model performance more reliably by splitting the data into k folds, training on k–1 of them, and validating on the remaining fold—repeating this process k times. This reduces the risk of overfitting and is especially useful for hyperparameter tuning. In Spark, CrossValidator is used with an estimator, evaluator. like F1-score, and a grid of hyperparameter values, like ParamGridBuilder. To tune a model we need to define a grid of hyperparameters (e.g., maxDepth = [5, 10, 15]), evaluate each

combination using cross-validation, then select the best model based on average validation performance.

Tuning can improve metrics like F1-score or AUC by 2–5%. Logistic Regression benefits from adjusting regularisation (e.g., elasticNetParam), while Random Forest and GBT gain from tuning tree depth and number of trees. With many correlated features, tuning helps reduce overfitting and improve generalisation.

Song recommendations

The triplets dataset contains over 48 million rows of user-song play interactions, where each row records a single instance of a user playing a song, including the user ID, song ID, and play count. Given the large size of the dataset (approximately 4.6 GB and 48,373,586 rows), repartitioning and caching are recommended to optimise performance. Repartitioning ensures better load balancing across the Spark cluster, and caching avoids repeated disk reads during iterative model training.

Repartitioning and caching can improve parallelism and speed up repeated access in ALS by reducing data shuffling during training and evaluation. However, caching uses memory and may not be feasible if resources are limited. Additionally, poor repartitioning can lead to skew or unnecessary overhead if it isn't aligned with data access patterns. Given the iterative nature of collaborative filtering algorithms, caching after repartitioning (e.g., by user ID) is justified for this workload.

Table 9, Statistics overview

Statistic	Value
Number of unique users	1,019,318
Number of unique songs	384,546
Songs played by the most active user	3,462
Percentage of all unique songs they played	~0.9%

User activity was defined as the number of unique songs listened to by each user, while song popularity was measured by either the number of distinct users who played a song or its total play count. As shown in Table 9, the most active user listened to only about 0.9% of the total catalogue, highlighting the sparsity of the user-song interaction matrix. This sparsity is a well-known characteristic of recommendation datasets and has important implications for algorithm design, scalability, and evaluation.

Table 10, Additional statistics for the dataset

Statistic	Value	Explanation
Mean play count	2.87	On average, a user listens to a track around 2.87 times, indicating light repeat behaviour.
Minimum play count	1	Many users only listen to a track once, which is typical in large music datasets.
Maximum play count	9667	Some users replay songs extensively, showing long-tail heavy users.
Mode play count (count = 1)	28,755,966	Most interactions are one-off listens, contributing to matrix sparsity.
Distribution snapshot	See below	The distribution rapidly drops off with increasing play counts.

Table 11, Top 10 most common play counts

Play Count	Number of Interactions
1	28,755,966
2	7,336,825
3	3,214,271
4	1,805,081
5	2,250,999
6	1,004,971
7	675,413
8	488,641
9	363,860
10	433,977

These statistics show that most tracks are played only once, while a few tracks receive thousands of plays. As a result, most user-song pairs are empty, which increases matrix sparsity and affects collaborative filtering performance or popular songs can dominate recommendations unless counts are normalised. As well as that, loss functions need to handle extreme play counts without being dominated by outliers and evaluation may require log-scaling play counts to prevent high outliers from skewing predictions.

In the context of collaborative filtering, song popularity and user activity were defined based on interaction counts in the user-song play dataset. Thus, song popularity was measured by the total number of times a song has been played across all users (i.e., cumulative play count). This metric identifies which songs are most frequently listened to overall. User

activity was measured by the number of unique songs a user has listened to. This helps capture how broadly a user explores the music catalogue, rather than how many times they replay the same songs.

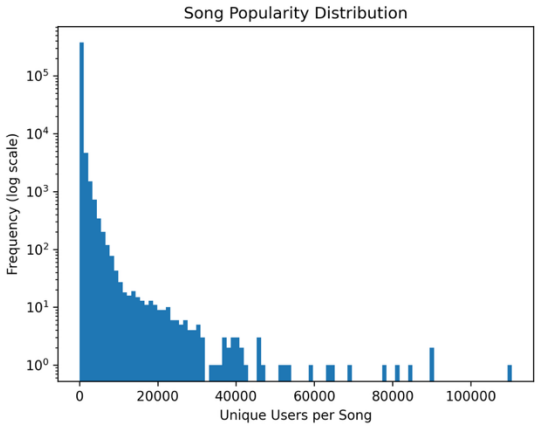

Figure 3, Song popularity

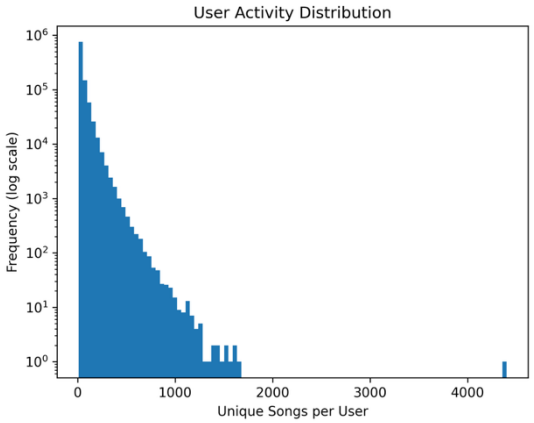

Figure 2, User activity

The first plot (Song Popularity Distribution) shows that most songs are played by very few users, while a small number of songs are listened to by thousands. This long-tail pattern means a few hits dominate plays. The second plot (User Activity Distribution) shows that most users listen to only a handful of unique songs, while a small number of highly active users listen to thousands. Again, this creates a long-tail effect where a few users generate most of the interaction data. To address this, I filtered the dataset to retain only users who listened to at least 20 unique songs, and songs that have been played at least 20 times across all users. These thresholds help to remove noise from rare interactions, which typically do not contribute much signal for collaborative filtering. After filtering:

⇒ 661,103 unique users were retained, and 358,215 users were excluded,

⇒ 161,173 unique songs were retained, and 223,373songs were excluded.

Filtering them out enhances model performance and scalability. This step was implemented by aggregating user_id and song_id counts, applying filters, and joining the results back to the main interaction dataset to retain only valid user-song pairs.

To ensure compatibility between the training and test sets, it is essential that every user and song in the test set also appears in the training set. This is particularly important for matrix factorisation methods such as ALS, which learn latent embeddings only for users and items seen during training.

To address this, I split the dataset after filtering, making sure all users in the test set had at least some interactions in the training set, but users with fewer than 5 interactions were

excluded to avoid cold-start issues, which collaborative filtering models are not equipped to handle without prior data.

For qualitative assessment, I extracted a sample of recommended songs for selected users and compared them with actual played songs. Where possible, I joined the results with metadata (e.g., artist name, song title) to interpret relevance as shown in table below.

Table 12, Example for recommendation

User ID	Played Songs	Recommended Songs
U1	“Let It Be” by The Beatles, “Imagine” by John Lennon	“Hey Jude” by The Beatles, “Bohemian Rhapsody” by Queen

In this example, the recommendations reflect stylistic and temporal similarity with the user’s past preferences, which provides the basis for evaluating model performance using ranking metrics on the test set.

The evaluation procedure samples 500 users from the test set to reduce computational load and filters both the predicted recommendations and the ground truth accordingly. After several failed attempts due to an empty RDD, a condition was added to ensure that the joined RDD is non-empty before computing ranking metrics such as Precision@10, NDCG@10, and MAP. This helps prevent runtime errors during evaluation.

Table 13, Ranking metrics

Metric	Value
Precision@10	0.075
MAP@10	0.079
NDCG@10	0.005

Precision@10 measures the proportion of relevant songs in the top 10 recommendations. One limitation is that it ignores the order of items and does not consider anything beyond the top 10. NDCG@10 rewards relevant songs that appear earlier in the list. A limitation is that it assumes a fixed drop-off in user attention, which may not match real user behaviour. MAP evaluates overall ranking quality across all users. It is sensitive to users who have very few relevant items, which can skew the results.

To evaluate a music recommendation system in a real-world setting, the focus shifts to observing user behaviour in production. Real-time evaluation is essential to measure the actual impact of recommendations, refine models, and ensure long-term user engagement, based on next methods:

1. Online A/B Testing.

   Users are randomly assigned to different algorithm groups (e.g., current system vs. new version). By comparing engagement patterns (such as skips, saves, or exits), we can determine which algorithm performs better in practice.

2. User Engagement Signals

   The system tracks implicit feedback from user interactions with recommended songs, including:

   ⇒ Play – positive signal indicating interest.
   ⇒ Skip (within a few seconds) – negative signal suggesting poor match.
   ⇒ 3+ Skips followed by exit – strong dissatisfaction signal.
   ⇒ Saved to playlist – strong positive indicator of long-term value.
   ⇒ Search for artist/album after play – shows curiosity and deeper relevance.
   ⇒ Share with others – top-tier signal; indicates social value and confidence.
   ⇒ Free vs. Paid Users – behaviour differs by tier:
     - Paid users show deeper intent; their actions are more stable for evaluation.
     - Free users may behave differently due to platform limits; they should be analysed separately.

3. Key Evaluation Metrics

   To quantify performance, several metrics are commonly tracked

Metric	**Description**
Precision@K	Proportion of top-K recommendations that were positively engaged with
Recall@K	Proportion of total user-engaged items that appeared in top-K
NDCG	Weighs the rank of relevant items — higher rank = higher reward
MAP	Aggregates precision across users; useful for system-wide relevance
Churn Rate	Measures drop in usage post-recommendation; high values suggest disengagement
Conversion Rate	% of users who upgraded or took desired actions after exposure

4. Segmented Evaluation

   Metrics should be reported separately for free and paid users. This controls for behavioural differences and reveals whether the model performs consistently across user types.

Conclusions

This project developed a scalable music recommender using ALS on filtered Million Song Dataset interactions. Despite the sparse data, the model achieved reasonable ranking metrics (e.g., Precision@10 = 0.075). Filtering improved performance, and sample recommendations matched user taste. While offline metrics were modest, the system shows potential for real-world use when paired with engagement-based evaluation.

References

1. Machine Learning Library (MLlib), https://spark.apache.org/docs/latest/ml-guide.html
2. Algorithmic Music Recommendations at Spotify, https://www.slideshare.net/slideshow/algorithmic-music-recommendations-at-spotify/29966335
3. Machine Learning for Recommender systems — Part 1 (algorithms, evaluation and cold start) https://medium.com/recombee-blog/machine-learning-for-recommender-systems-part-1-algorithms-evaluation-and-cold-start-6f696683d0ed
4. Grammar checker, OpenAI, ChatGPT, June 2025