A Comprehensive Analysis of The Million Song Dataset

Course: DATA420 Scalable Data Science

Student Name: Junlin Jiang

Student ID: 79296358

Date: 27/10/2023

Background

The Million Song Dataset (MSD) is a comprehensive music dataset that integrates both audio and textual data. Established collaboratively by The Echo Nest and LabROSA, it has gradually become a cornerstone for music research and applications. The core dataset covers a multitude of detailed fields along with numerous audio attributes (Million Song Dataset, n.d.).

In addition, the MSD incorporates several sub-datasets contributed by different organizations and communities. Notably, there's the Taste Profile subset that logs real interactions between users and songs. There's also the MSD AllMusic Genre Dataset (MAGD) which offers detailed genre annotations for songs (Million Song Dataset, n.d.).

Before diving deep into music data research, a pivotal step is data processing. we undertook a preliminary exploration of each dataset. Subsequently, this report delves into the audio features of the MSD. These features, meticulously extracted by the Music Information Retrieval research group at the Vienna University of Technology, span rhythm patterns, statistical spectral descriptors, and more (Vienna University of Technology, 2007). Our aim is to unveil the intrinsic properties of these features, their interrelationships, and their applicability in the realm of machine learning.

Next, we shift my focus to the song recommendation system, primarily based on the Taste Profile dataset. Our goal is to offer personalized song recommendations for each user through collaborative filtering algorithms. This necessitates a profound understanding of the dataset's structure, its unique songs and user information, and how to efficiently train the collaborative filtering models.

Throughout this journey, I encountered numerous challenges. The most salient was the less than ideal performance of specific evaluation metrics such as Precision @ 10, NDCG @ 10, and Mean Average Precision (MAP) when assessing the efficacy of my collaborative filtering models. This hints that I might need to further refine my model. Lastly, inconsistencies and matching errors in some of the data presented additional hurdles for me.

Data Processing

Q1

Processing large-scale music datasets often involves critical data preprocessing. This ensures data quality and consistency, providing a strong foundation for subsequent analysis and machine learning applications. Now we will explore how to read data from various datasets, their structures, and addressing potential data mismatch issues.

Our datasets are stored in several HDFS locations. Here's an overview:

Main Directory: This contains song metadata from the main million song dataset but excludes audio analysis, similar artists, and tags. Located at hdfs:///data/msd/main, it has a 'summary' sub-directory with compressed files: analysis.csv.gz and metadata.csv.gz. The

total size of data in this directory is approximately 174.4 M.

Taste Profile Directory: Found at hdfs:///data/msd/tasteprofile, it offers user-song play counts and includes logs for mismatches identified and manually accepted matches. There are two sub-directories: mismatches (with files sid_matches_manually_accepted.txt and sid_mismatches.txt) and triplets.tsv. The total size of data in this directory is approximately 490.4 M.

Audio Directory: Located at hdfs:///data/msd/audio, this directory from the Music Information Retrieval research group at Vienna University offers audio features. It's divided into three sub-directories: attributes (header names from the ARFF), features (audio features), and statistics (track statistics). The sizes of the three sub-directories are 101.4 K, 11.7696 GB, and 0.0403 GB respectively. Among all the directories, the "features" directory has the largest data size. This is due to its inclusion of some large datasets, such as msd-rp-v1.0.csv, which alone is approximately 4GB in size.

Genre Directory: Found at hdfs:///data/msd/genre, it has genre-related data with three files: msd-MAGD-genreAssignment.tsv, msd-MASD-styleAssignment.tsv, and msd-topMAGD-genreAssignment.tsv. The total size of data in this directory is approximately 30.1 M.

Datasets are primarily stored in CSV and TSV formats, with some files being compressed using GZ. The datasets consist of data types like StringType and DoubleType. The data is hierarchically organized in directories and sub-directories within HDFS, allowing for efficient distributed storage and processing.

We obtained the total number of unique songs by using .select(“song_id”).distinct().count() to get the unique values of “song_id”, and the result for the total number of unique songs is 998,964, which is close to a million songs. Additionally, Table 1 below shows the number of rows for some of the datasets, and the specific row counts can be found in the supplementary materials. The results indicate that the number of rows in each dataset (except for the metadata file under the main/summary/ path itself) is less than 998,964. This implies that not all songs in the main dataset have corresponding data in the supplementary datasets. For instance, not every song might have a user-song play count in the Taste Profile dataset or might not have audio features extracted by the Vienna University of Technology.

File	NumberOfRows
msd-jmir-area-of-moments-all-v1.0.attributes.csv	21
msd-jmir-lpc-all-v1.0.attributes.csv	21
msd-jmir-methods-of-moments-all-v1.0.attribute...	11
msd-jmir-mfcc-all-v1.0.attributes.csv	27
msd-jmir-spectral-all-all-v1.0.attributes.csv	17
msd-jmir-spectral-derivatives-all-all-v1.0.att...	17
msd-marsyas-timbral-v1.0.attributes.csv	125
msd-mvd-v1.0.attributes.csv	421
msd-rh-v1.0.attributes.csv	61
msd-rp-v1.0.attributes.csv	1441
msd-ssd-v1.0.attributes.csv	169
msd-trh-v1.0.attributes.csv	421
msd-tssd-v1.0.attributes.csv	1177
msd-jmir-area-of-moments-all-v1.0.csv	994623
msd-jmir-lpc-all-v1.0.csv	994623
msd-jmir-methods-of-moments-all-v1.0.csv	994623
msd-jmir-mfcc-all-v1.0.csv	994623
msd-jmir-spectral-all-all-v1.0.csv	994623
msd-jmir-spectral-derivatives-all-all-v1.0.csv	994623
msd-marsyas-timbral-v1.0.csv	995001

Table 1 Row counts of each dataset

Q2

Given the nature of large datasets, mismatches and inconsistencies are often inevitable. In the Taste Profile dataset, some tracks in the MSD are incorrectly matched with songs. This issue may not be significant for our subsequent research if we only need to use the song_id. However, since our next section involves song recommendations and audio similarity, which involves the extensive use of track_id, I believe we need to consider the mismatches.

To address this issue, we used the code provided by James to read these mismatched files from the Hadoop Distributed File System (HDFS) and processed them to extract relevant information. Mismatches are identified by lines starting with “ERROR:” or “< ERROR:”. Each line was then parsed to extract details of the song and track, such as song_id, song_artist, and track_id, etc. We then used Spark's DataFrame API to transform the extracted data into a structured format. Our analysis showed that 19,094 mismatches were automatically identified. Upon cross-referencing with manually accepted mismatches, we found that 4,888

mismatches were not accepted.

When we compared the complete triplet dataset with the unaccepted mismatch dataset, we noticed the count decreased from 48,373,586 to 45,795,111.

To automate the process of loading these datasets with the correct schema, the attribute files were first checked to identify the unique data types present. Four distinct data types were identified: “real”, “NUMERIC”, “STRING”, and “string”. A mapping was then created to translate these data types to their respective PySpark data types.

For each attribute file, a schema was generated based on the feature names and data types listed in the file. This schema was then used to load the corresponding audio feature dataset into a DataFrame. This approach ensures that each dataset is loaded with the correct schema, facilitating accurate and efficient data processing and analysis.

The automated schema generation process ensures efficient loading and processing of multiple datasets without manual intervention. It also guarantees consistency in data types across different datasets, simplifying subsequent data processing and analysis.

Audio similarity

Q1

In this section, we selected the “msd-jmir-methods-of-moments-all-v1.0” dataset. This dataset is one of the smaller audio feature datasets, providing a manageable size for efficient data analysis.

We used .describe() and .summary() to observe the descriptive statistics of the chosen dataset, and the following summary statistics for each feature column were observed (Detailed descriptive statistics can be found in the supplementary materials):

The dataset contains a total of 994,623 records.
Features such as “Method_of_Moments_Overall_Standard_Deviation_1” have a mean value of approximately 0.155 and a standard deviation of approximately 0.066.
On the other hand, features like “Method_of_Moments_Overall_Average_5” have a much larger mean value of approximately 2.396783048473542E7 and a standard deviation of around 9307340.299219666.
Some feature values range from 0 to smaller maximum values (e.g., 0.959 for “Method_of_Moments_Overall_Standard_Deviation_1”), while others have larger ranges (e.g., -146300.0 to 452500.0 for “Method_of_Moments_Overall_Average_4”).

To examine the relationships between the audio features, Pearson correlation coefficients were calculated for each pair of features. Using a threshold of 0.95 for strong correlations, the results(Table 2) indicate that several feature pairs exhibit strong correlations. This suggests the presence of potential multicollinearity. Such multicollinearity can affect the performance and interpretability of certain machine learning models.

0	1	2	3	4	5	6	7	8	9
1	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0	1	1

Table 2 Correlation Matrix Above Threshold

To understand the distribution of genres and their representation in the dataset, we initiated the process by loading the entire MSD music genres dataset (MAGD). First, we used Spark's DataFrame API to read the dataset. The data is tab-separated and has no headers. A predefined schema, genres_schema, was applied to ensure the correct data types.

After loading, we discovered that each track ID is associated with a specific genre.

For the analysis of genre distribution, we grouped songs by their genre, counted them, and then sorted them in descending order based on the counts.

The genre distribution chart (as shown) illustrates the popularity of various music genres within the dataset. Genres such as “Pop_Rock,” “Electronic,” and “Rap” had the highest counts, making them the most predominant genres in the dataset. On the other hand, genres like “Holiday,” “Children,” and “Classical” had lower counts, indicating their lesser popularity. The distribution of genres gives a foundational understanding of our dataset. By knowing which genres dominate our dataset, we can infer potential biases in the results of song similarity. For instance, due to their large numbers, songs from the “Pop_Rock” genre might have a higher likelihood of being chosen as similar songs compared to those from the “Classical” genre.

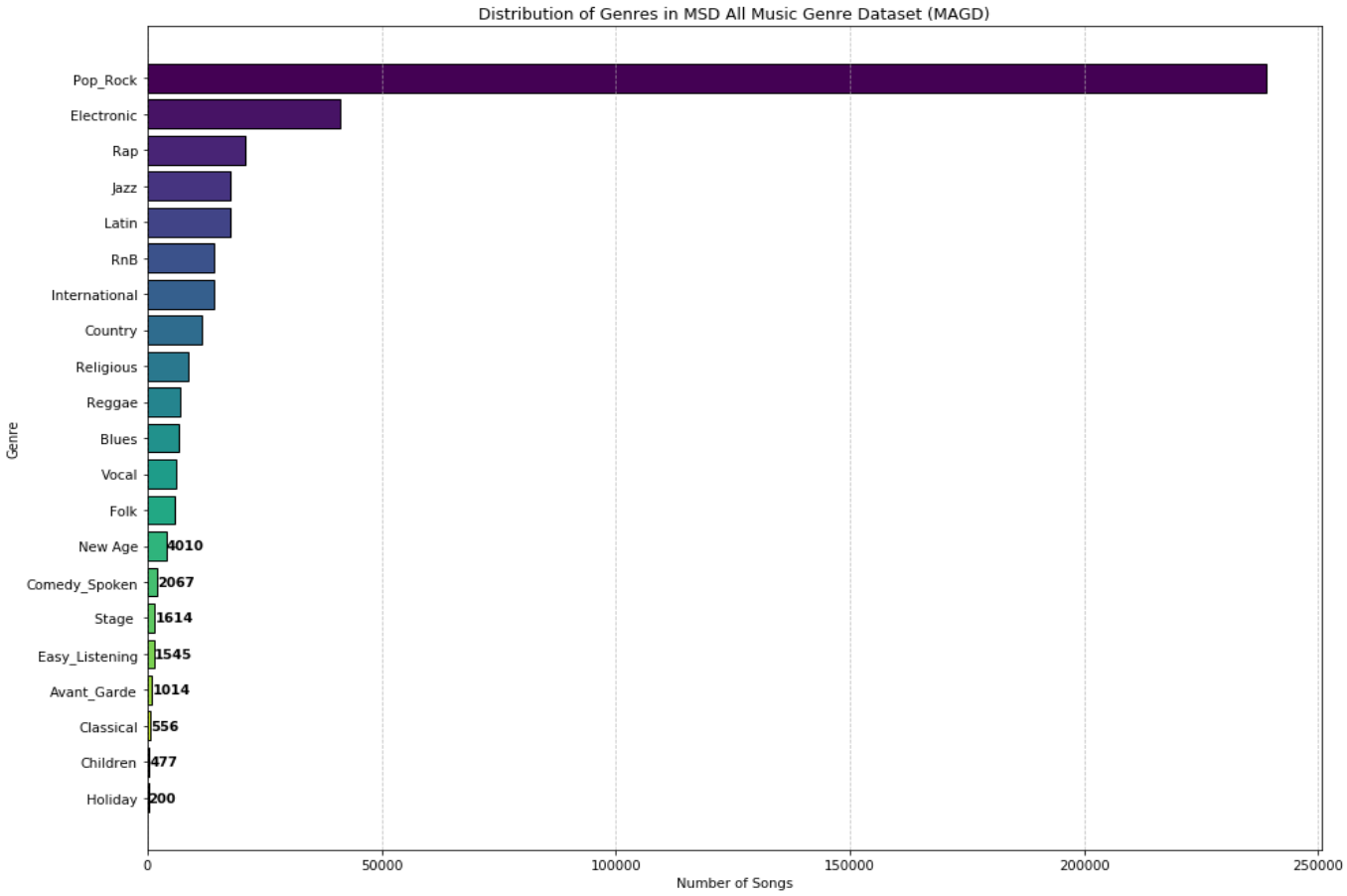

Figure 1 Distribution of Genres in MSD ALL Music Genre Dataset (MAGD)

Next, we are going to merge two datasets: genres and audio features. First, the dataset with song features has a column named MSD_TRACKID which contains track IDs with single quotes. We have removed these single quotes and renamed the column to track_id for consistency.

The cleaned dataset was then merged with the genres_MAGD dataset using the track_id column as the common key. This procedure also made sure that only the songs with a genre label were retained. The merged dataset displays a series of songs, each with a set of features – “Method of Moments Overall Standard Deviation” and “Method of Moments Overall Average” for five different segments.

Q2

I have chosen Logistic Regression, Random Forest, and Gradient-Boosted Trees (GBTs) as classification algorithms.

Logistic Regression

Logistic Regression is one of the simplest and most interpretable classification models. The weights assigned to each feature can be directly analysed to understand the influence of each feature on the outcome. For data that is linearly separable or datasets with a large number of features (like audio datasets), logistic regression can offer competitive

performance. Being a linear model, it's computationally efficient and trains quickly. The primary hyperparameters to adjust are maxIter (maximum number of iterations), regParam (regularization parameter), and elasticNetParam (mixing parameter for ElasticNet regularization), which can help in preventing overfitting. Logistic Regression might struggle with very high-dimensional data unless regularized. Feature scaling is crucial since the algorithm relies on gradient descent.

Random Forest
While tree-based models are inherently interpretable, Random Forest, which aggregates many trees, may be harder to interpret than a single decision tree. However, it provides feature importance scores. Random Forests tend to be among the top performing models for tabular data due to their ability to capture non-linear patterns. Training multiple trees can be parallelized, but it might be slower than simpler models on large datasets. There are several hyperparameters to consider, such as numTrees (the number of trees) and maxDepth (the maximum depth of the trees), but they offer a great deal of flexibility. Random Forests handle high dimensionality well and are not sensitive to feature scaling.

Gradient-Boosted Trees (GBTs)
Like Random Forest, GBTs are harder to interpret than a single decision tree but offer feature importance scores. GBTs typically offer superior performance by sequentially building trees, where each tree corrects the mistakes of the previous one. Training is sequential, so it might be slower than random forests, especially with a large number of trees. Parameters such as maxIter (maximum number of iterations) and maxDepth (maximum depth of the trees) need to be tuned. Proper tuning is crucial to avoid overfitting. They handle high-dimensional data well and are not highly sensitive to feature scaling, although they might benefit from it.

From the descriptive statistics, it's evident that the range and scale of the features vary widely. For Logistic Regression, which relies on gradient descent, such variability can have a significant impact. Thus, it's crucial to centre and standardize the features using the StandardScaler, which we plan to do later. This process aligns the features' mean at 0 and standardizes their standard deviation to 1. Although GBTs and Random Forests are not sensitive to feature scaling, usually undertaking this procedure is harmless. Moreover, in some instances, it might enhance the model's convergence speed and performance.

For analytical ease, the genre column was transformed into a binary format. Songs of the "electronic" genre were assigned a value of 1, while all other genres were given a value of 0. The purpose of this transformation was to identify and quantify electronic songs in the dataset relative to other genres. Upon computing class balance, it was found that the number of electronic songs was 40,666 while songs of other genres amounted to 379,954. When these figures are converted to percentages, it is evident that the dataset contains approximately 9.67% of electronic songs, with the remaining 90.33% being non-electronic songs. This information is crucial as it reveals an imbalance between the two classes. Such imbalances typically lead to a bias in machine learning models, potentially resulting in poor

generalization, especially for the minority class (in this case, electronic songs).

Given the observed class imbalance, stratified random sampling was adopted when splitting the dataset into training and testing sets. Here, we employed the Window function instead of the built-in Spark randomSplit function. While the randomSplit function indeed provides a straightforward way to randomly split the dataset, it doesn't ensure an exact distribution for each class. With the current data imbalance, using the Window function allows us to prevent any class from being underrepresented in either the training or test sets. The data was divided into 80% training and 20% testing. Subsequently, the features of the dataset were standardized and centred using the StandardScaler. This step is vital to ensure all features have the same scale, facilitating better convergence and performance of machine learning algorithms.

However, given the severe class imbalance, resampling techniques were deemed necessary. For this, Poisson random oversampling was employed on the training data. Oversampling helps equalize the number of instances between the two classes, potentially enhancing model performance, particularly for the minority class. After resampling, the distribution in the training set was nearly equal, with the results indicating 379,409 songs for the electronic genre (roughly 49.94%) and 379,859 songs for other genres (about 50.00%). This balanced representation is anticipated to be beneficial during the model training phase, ensuring the model doesn't exhibit a strong bias towards the majority class.

Following this, we trained each model using the upsampled training data and tested the models on the test set. Subsequently, a series of performance metrics were calculated for each model, including precision, accuracy, recall, and AUROC.

The results were as follows (round to two decimal places):

Logistic Regression Model: Precision: 20.04%, Recall: 72.29%, Accuracy: 69.44%, AUROC: 77.39%

Random Forest Model: Precision: 21.73%, Recall: 70.79%, Accuracy: 72.52%, AUROC: 78.94%

Gradient-Boosted Tree Model: Precision: 23.11%, Recall: 70.27%, Accuracy: 74.52%, AUROC: 81.12%

For precision, the Gradient-Boosted Tree model leads with 23.11%, followed by the Random Forest at 21.73% and the Logistic Regression at 20.04%. Precision indicates the accuracy of the predicted electronic songs.In recall, the Logistic Regression tops with 72.29%, showing the proportion of actual electronic songs correctly predicted. This suggests the model identifies most electronic songs, potentially with some false positives. For accuracy, the Gradient-Boosted Tree model stands out with 74.52%, measuring overall correct predictions.

Regarding AUROC, the Gradient-Boosted Tree again leads at 81.12%, reflecting its ability to distinguish between song classes.

Overall, the Gradient-Boosted Tree model proves the most effective, trailed by the Random Forest and Logistic Regression. The model's performance is influenced by the class balance. Our dataset's minority is electronic songs compared to non-electronic ones. This imbalance can skew accuracy metrics. Although we've resampled to address the imbalance, the models' precision remains modest. This means that while the models identify many electronic songs (high recall), they also misclassify several non-electronic ones. Techniques like SMOTE or adjusting decision thresholds might further improve these results.

Q3

In this section, we discuss the fundamental role of hyperparameter tuning in enhancing the performance of machine learning algorithms. Below are the hyperparameters for the models trained in Q2.

Logistic Regression:

elasticNetParam (0.0): Controls the mix between L1 and L2 regularization. A value of 0 implies L2 regularization, and a value of 1 implies L1. Currently, L2 regularization was used. Consider adjusting for a more detailed grid search.
fitIntercept (True): Specifies whether to fit an intercept term. It's set to true, ensuring the model accounts for the bias.
maxIter (100): Represents the maximum number of iterations. More iterations can lead to a better fit but might also lead to overfitting. Here, consider adjusting the number of iterations to see if it can optimize the model's performance.
regParam (0.0): Regularization parameter. A value of 0 implies no regularization. Consider adjusting for a more detailed grid search.
standardization (True): Whether to standardize the training features before fitting the model.

Random Forest:

numTrees (20): The number of trees in the forest. More trees can reduce the variance. Given the ensemble nature of the random forest, adding more trees might improve performance, but the computational cost will also increase. I might test with 10, 20, and 30 trees.
maxDepth (5): Depth of the tree, which affects the model's complexity. I might test different depths.
maxBins (32): Number of bins used for splitting features.
featureSubsetStrategy (auto): Number of features to be used for training each tree.
subsamplingRate (1.0): Fraction of the training data used for learning each decision tree.

Gradient-Boosted Tree:

maxDepth (5) & maxBins (32): Same as Random Forest. Likewise, I might also test different depths.
maxIter (20): The number of boosting iterations. I might test different iterations to see if it can optimize the model's performance.

stepSize (0.1): Step size to be used for each iteration of optimization.
lossType (logistic): The loss function to be optimized.

The fundamental idea behind cross-validation is to divide the training dataset into two parts: a sub-training dataset and a validation dataset. The model is trained on the sub-training dataset and validated on the validation dataset. This process is repeated k times, producing k models and performance estimates. The final performance of the model is an average of these models. In this report, we have set up k-fold cross-validation, where k=5. This means our dataset is divided into 5 parts, the model is trained on 4 parts, and validated on the remaining 1 part. This process is repeated 5 times.

On one hand, cross-validation reduces overfitting and provides a generalized performance metric for the model. On the other hand, by combining grid or random search with cross-validation, we can identify the best hyperparameters for the model.

For the general approach to hyperparameter tuning, we first create a grid of possible hyperparameter combinations. Then, for each hyperparameter combination, we train the model using cross-validation and record performance metrics. After evaluating all combinations, we find the hyperparameters that produce the best performance on the validation set. Based on our results (refer to supplementary materials), hyperparameter tuning appears to have the most impact on tree-based models (Random Forest and Gradient-Boosted Tree), with improvements observed in accuracy and AUROC. For Logistic Regression, the benefits of tuning are minimal in this case.

Q4

We chose Logistic Regression from the algorithms in Q2 for multiclass classification. While traditionally used for binary classification, Logistic Regression can be extended for multiclass scenarios using methods such as the “one-vs-all” (or “one-vs-rest”) approach. In this strategy, for a problem with “N” classes, “N” separate binary classifiers are trained. For each classifier, one class is treated as positive while all other classes are considered negative. The class that gets the highest probability from its respective binary classifier during prediction is chosen as the final predicted class.

By using the StringIndexer class from PySpark's MLlib, we could consistently encode each unique genre into a respective index. The advantage of this method is that it consistently indexes each genre and requires minimal manual intervention. Then, by using cast("int"), we converted the index from double type to integer type.

Before splitting into training and testing sets, we first selected columns related to the “Method_of_Moments” feature and standardized these features using the StandardScaler. For our multiclass dataset, the training-test split must preserve the class distribution to avoid introducing bias. We used a stratified sampling method to divide the data into 80% for training and 20% for testing.

Upon examining the class distribution in the training set, we observed a class imbalance. To address this, we adopted a hybrid approach combining both oversampling and undersampling techniques. The class counts were re-sampled using Poisson random sampling to fall between the 50th and 75th percentiles of the original class distribution.

After resampling, we trained a logistic regression model on the training data and evaluated its performance on the test set using various metrics, ultimately obtaining the following results (round to two decimal places). Test Accuracy: 34.47%, Weighted Precision: 55.01%, Weighted Recall: 34.47%, Weighted F1 Score: 39.49%.

From these metrics, it seems that incorporating multiple genres can lead to a decline in model performance. One possible reason is that the audio features of some genres are closely related.

Song recommendations

Q1

Based on the information we obtained before, we know that the size of the Taste Profile dataset is 488.4 M. Loading such a dataset into memory multiple times can be inefficient. Given its format (tsv and gzipped), repeatedly reading it can consume a lot of resources. The dataset has over 48 million rows (before mismatches are removed), meaning there's a significant number of rows that will be involved in the matrix factorization step of collaborative filtering, leading to substantial shuffle operations.

In the song recommendation section, our Spark configuration is set to (executor_instances=4, executor_cores=2, worker_memory=4, master_memory=4). With the current setup, the dataset will have roughly 32 partitions (4 executors x 2 cores x 4). Repartitioning can reduce data shuffle across the nodes. The number of partitions should be set to a multiple of the total cores available to ensure optimal parallelism. Caching would also be beneficial. Collaborative filtering requires iterative operations, meaning the same dataset will be accessed multiple times. Caching the dataset in memory would significantly speed up these operations. Given that our total executor memory is 16GB (4 executors x 4GB), and the dataset is 488.4 M, it should fit easily into memory.

After repartitioning and caching, we used the triplets dataset without mismatches to identify the unique songs and users. This was effortlessly achieved using the methods select("song_id").distinct().count() and select("user_id").distinct().count(). The results yielded were: Total Unique Songs: 378,310 and Total Unique Users: 1,019,318. It's an interesting insight, indicating that even the most engaged users have only scratched the surface of the song repository.

Next, we grouped the data by user_id and sorted it in descending order by count to identify the user who has played the most songs. For the identified user, we then filtered the dataset to determine how many distinct songs this individual has played. The results showed that the

most active user in the dataset has played 4,316 unique songs, which represents 1.14% of the total unique songs available in the dataset.

In the visualization section(Shown in Figure 2), we represented the distributions of song popularity and user activity in two steps.

First, we grouped the data by song_id and counted the play instances for each song, ordering them in descending order by play count. Similarly, we grouped the data by user_id and tallied the number of songs played by each user, arranging them in descending order based on song counts.

In the second step, we employed histograms to visualize the data. For the song popularity distribution, the x-axis displays the number of plays, and the y-axis presents the number of songs, with data plotted in a "dodgerblue" colour. For the user activity distribution, the x-axis illustrates the count of songs played, the y-axis indicates the number of users, and the data is plotted in a "coral" hue.

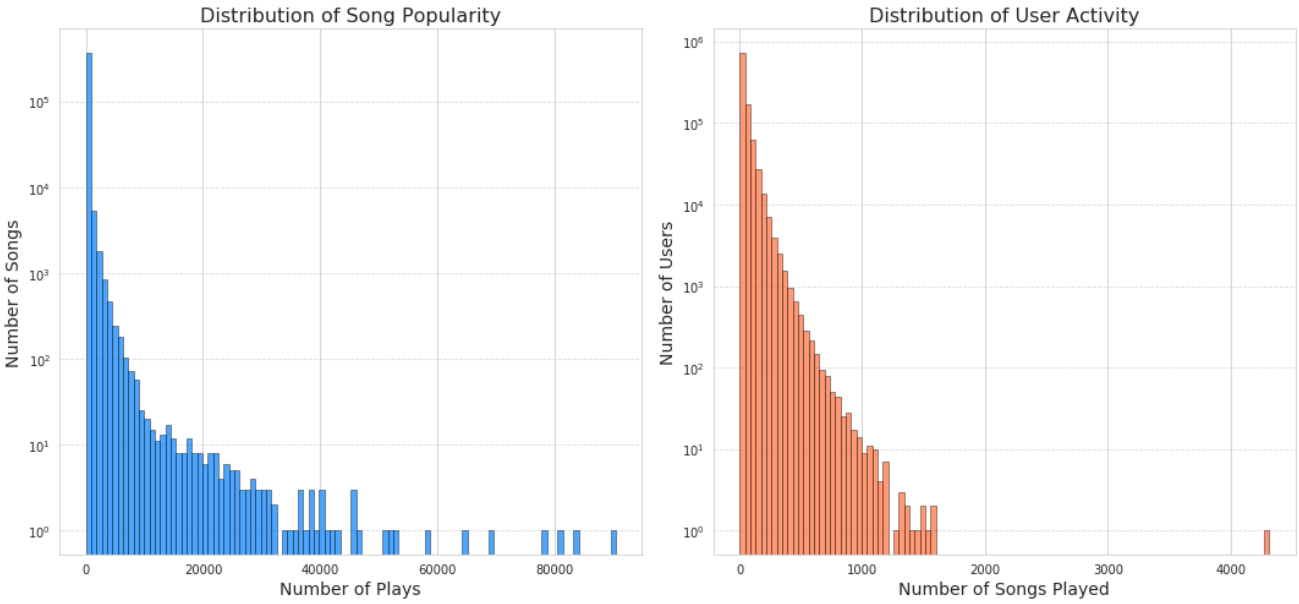

Figure 2 Distribution of Song Popularity and User Activity

The distribution of song popularity exhibits a severe right-skew. This implies that a small fraction of songs are extremely popular and account for the majority of play counts, while the vast majority of songs have fewer play counts. Similarly, the distribution of user activity displays a similar right-skewed pattern. A small group of users have a high number of song plays, indicating their high level of interaction with the platform. In contrast, most users have relatively fewer songs played.

Given the large number of songs and users, the user-song matrix will be sparse, meaning that most users have not listened to most songs. Collaborative filtering algorithms,

particularly matrix factorization, can handle this sparsity. However, for new songs or users with limited interaction data, providing accurate recommendations may be challenging, and alternative strategies may be necessary in such cases.

Q2

To optimize the performance of our collaborative filtering model, it's essential to work with a representative dataset. In the vast pool of songs and users, not all contribute substantial information for meaningful predictions. For our dataset, to define the threshold for the "few times" songs are played and the "minimal interactions" a user has, we took a data-driven approach: We calculated the 25% quantile for song plays and user activity. This means that we considered that songs below the 25% quantile (N = 4.0) have been played less frequently, and users below the 25% quantile (M = 15.0) have interacted with fewer songs.

By choosing these values for N and M, we removed the lower quartile of data. This decision ensures that our model is built on songs and users with a relatively higher interaction, enhancing the quality of our training data and thereby potentially increasing the accuracy of our recommendations.

Ensuring that every user in the test set also has some user-song plays in the training set is paramount for collaborative filtering models. This is because collaborative filtering relies on historical data to make recommendations. If users in the test set are not present in the training set, the model will have no historical data to make recommendations for those users. Additionally, when evaluating the model using the test set, the goal is to predict songs based on a user's past preferences. If a user is absent from the training set, it's impossible to assess the model's accuracy for that user.

To achieve this, in our data splitting process, we used the rand function to assign a random value between 0 and 1 to each user-song play. Then, we filled the training dataset by filtering rows where the assigned random value was less than or equal to 0.75. This ensured that on average, 75% of each user's plays went into the training set.

For the test set, we specifically used a "left anti join" on the full dataset using the training dataset as the reference. This approach guaranteed that the test dataset consists of the 25% of plays that were not included in the training set.

By following this method, we inherently made sure that every user in the test set also had plays in the training set. The randomness in the process came from the initial random assignment of values, ensuring that there was no fixed bias in the selection of songs for the training and test sets for any given user.

To train a collaborative filtering model, we used the spark.ml library to train an implicit matrix factorization model using Alternating Least Squares (ALS). To ensure the model can interpret the categorical values for user_id and song_id, these were converted into numerical values using the StringIndexer method. By converting these identifiers into encoded

numerical values, the data could be more readily processed and manipulated. The ALS parameters were set with a maximum of 5 iterations, a regularization parameter of 0.01, and implicit preferences turned on to focus on the implicit feedback from the dataset.

To evaluate the performance of the trained ALS model, we carefully selected a few users from the test dataset and generated recommendations. This helps visually assess how well the model's song recommendations match the songs that users have historically played.

We focused on the top 5 users in terms of the most played songs in the test dataset. Then, we generated subsequent recommendations for each of these users. The results indicate that for user 0.0, there is no overlap between the played songs and the recommended songs. Similarly, for user 6.0, there are no overlapping songs either. In contrast, for user 3.0, there are 2 overlapping songs between the played songs and the recommended list, while user 1.0 has 3 overlapping songs, and user 4.0 has 2 overlapping songs. This variation in overlap highlights the complexity of song recommendations and suggests that while the model captures the preferences of some users effectively, there is still room for improvement in capturing the preferences of other users.

To evaluate the effectiveness of the collaborative filtering model, we calculated the following metrics:

Precision @ 10: This metric calculates the proportion of recommended songs in the top 10 that were actually listened to by the users. Our model achieved a precision of 0.12000, indicating that 12% of the top 10 recommended songs were indeed played by the users.

Mean Average Precision (MAP) @ 10: MAP takes into account the order of the recommendations, rewarding more for relevant songs that appear earlier in the recommendation list. A MAP score of 0.08810 suggests room for improvement in our model.

NDCG @ 10 (Normalized Discounted Cumulative Gain): NDCG, like MAP, accounts for the rank of the recommendation but also includes a logarithmic discount for songs that are lower on the list. Our model achieved an NDCG of 0.14944.

These metrics collectively provide a comprehensive assessment of the model's performance in terms of relevance, ranking, and overall precision. By considering these metrics, we can measure the model's ability not only to recommend relevant songs but also to rank them effectively. However, these metrics also have some limitations. Firstly, high precision or MAP does not necessarily imply that the recommendations are diverse or novel for users. Secondly, they primarily focus on the top N recommendations, which may not always reflect the overall performance.

In a real-world scenario, A/B testing can serve as another effective approach. Users can be randomly divided into two groups: one group receives recommendations from the current

service, while the other group receives recommendations from the new recommendation service. Comparing user engagement metrics can provide insights into the real-world effectiveness of the recommendation algorithms.

Moreover, if we could measure future user-song plays based on our recommendations, other useful metrics could include:

Listening Duration: An aggregate of the duration for which users listen to the recommended songs.

Playthrough Percentage: The proportion of recommended songs that users listen to completely, indicating song satisfaction.

Conclusions

In this report, we conducted a comprehensive exploration of the Million Song Dataset (MSD). Firstly, the report focuses on audio similarity and genre distribution, with the primary genres identified as "pop-rock," "electronic," and "rap." Classification models such as logistic regression, random forests, and gradient boosting trees were employed for genre prediction, with the gradient boosting tree model showing the most promising results. Hyperparameter tuning was a critical aspect of optimizing model performance and received careful attention. Cross-validation combined with k-fold techniques enhanced the optimization process, with tree-based models benefiting the most.

Furthermore, the report also meticulously examined the dataset to gain insights into and optimize a music recommendation system. Milestones achieved in this process included rigorous data cleaning, in-depth visualizations of song popularity and user activity, and a systematic, data-driven approach to model optimization and training. Visualizations demonstrated a significant right skew in the distribution of song popularity and user activity, implying that a limited set of songs received special favour while a minority of users were particularly active.

However, our journey was not without challenges, and there were tasks that we couldn't complete. One prominent issue was our inability to further refine and optimize our recommendation model due to time and cluster resource constraints.

Overall, this study delved into the complexity of the MSD and made significant progress, but there is still ample room for improvement and further research.

References

Bertin-Mahieux, T., Ellis, D. P. W., Whitman, B., & Lamere, P. (2011). The Million Song Dataset. *In Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011).*

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., Oliphant, T. E. (2020). Array programming with NumPy. *Nature, 585,* 357–362. https://doi.org/10.1038/s41586-020-2649-2

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering, 9*(3), 90–95.

Inc., P. T. (2015). *Collaborative data science.* Plotly Technologies Inc. Montreal, QC: Plotly Technologies Inc. https://plot.ly

Kettle, S. (2017). *Distance on a sphere: The Haversine Formula.* Esri. https://community.esri.com/t5/coordinate-reference-systems-blog/distance-on-a-sphere-the-haversine-formula/ba-p/902128#:~:text=For%20example%2C%20haversine(%CE%B8),longitude%20of%20the%20two%20points.

McKinney, W., & others. (2010). Data structures for statistical computing in python. *In Proceedings of the 9th Python in Science Conference, 445,* 51–56.

Menne, M. J., Durre, I., Vose, R. S., Gleason, B. E., & Houston, T. G. (2012). An overview of the global historical climatology network-daily database. *Journal of atmospheric and oceanic technology, 29*(7), 897-910. https://doi.org/10.1175/JTECH-D-11-00103.1

Million Song Dataset. (n.d.). *Welcome!* http://millionsongdataset.com/

Vienna University of Technology. (2007). *Audio Feature Extraction.* http://www.ifs.tuwien.ac.at/mir/audiofeatureextraction.html

Waskom, M. L., (2021). seaborn: statistical data visualization. *Journal of Open Source Software, 6*(60), 3021, https://doi.org/10.21105/joss.03021.