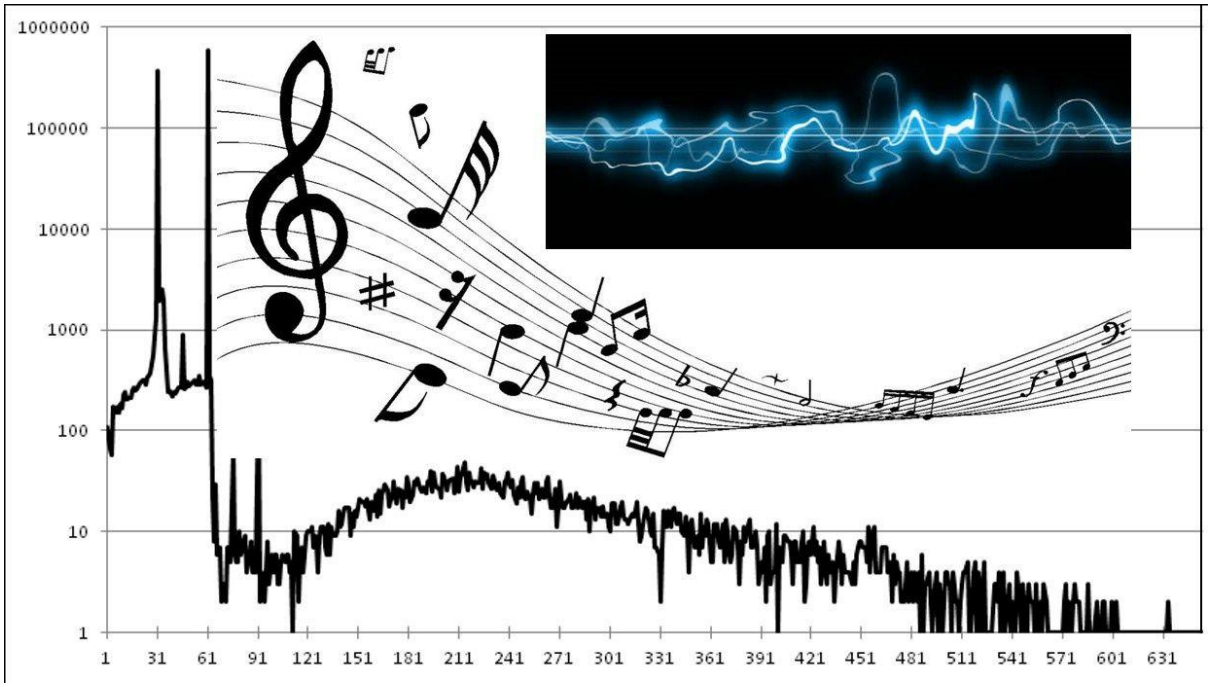
DATA420-24S1 (C)

Assignment 2

The Million Song Dataset (MSD)



Chathurangi Godahewa Gamage

Table of Content

Background

This assignment is directed at analysing the Million Song Dataset (MSD) to understand the audio and textual content of recorded music. The MSD consists of more than 50 fields, including song ID, track ID, etc. For this assignment, we are only concerned about the Taste Profile Dataset and the All-Music Genre Dataset in MSD. The main focus of the assignment is to apply practically what we learned from the lectures.

In the Processing section, we used Apache Spark and Hadoop to understand the structure, sizes, and other basic details, like how data files are saved in each directory. Firstly, I faced some difficulties in Q2 in the processing section. However, after James's Help sessions, lab classes provided immense support to solve my problems. This time, the assignment was not much more difficult than the first one because James provided many helping codes related to the questions. We gained much knowledge about descriptive statistics, classification methods, resampling methods, collaborative filtering, and recommendations, and how we practically use them and analyse data using the results we got. ChatGPT also supported understanding each concept and related codes.

I think it's essential to refer to most related pages and blog posts in the Million Song Dataset to get an idea about the background. Apart from that, like Assignment 1, PySpark master documentation became helpful for learning how to apply functions correctly.

Data Processing

Q1

Structure of the datasets

The MSD (Million Song Data) set is outlined with the "hdfs:///data/msd" directory because we hosted data in HDFS (Hadoop Distributed File System). The data set consists of different information types of directories, including audio, genre, main and tasteprofile. Audio data frames consist of 'attributes', 'features', and 'statistics' files. Attributes files and features files are situated as 'CSV' files, while statistics files form 'csv.gz', GZIP-compressed files to save space.

The genre directory consists of three items formed as TSV files. The tasteprofile data set consists of two items. The first is a mismatched file formed as TXT files, and the other is triplets.tsv, which consists of tsv.gz compressed file forms. The main/ summary directory consists of CSV.GZ compressed files to save space.

(A graphical picture of how data is structured is in the appendix B)

Data sizes in the MSD dataset

File size (compressed)	File size (without compress)	Data Files /data/msd/
12.3 G	**98.1 G**	**audio**
30.1 M	**241.0 M**	**genre**
174.4 M	**1.4 G**	**main**
490.4 M	**3.8 G**	**tasteprofile**

Table 1 - Distribution of data sizes in the MSD dataset

Audio

File Name	Size	Format	Data Types	Storages in HDFS
attributes	103.0K / 824.3 K	CSV	String	Total Blocks – 13 Avg Replication - 8
features	12.2 G / 97.8 G	CSV	String Double	Total Blocks – 175 Avg Replication - 8
statistics	40.3 M / 322.1 M	CSV.GZ	String	Total Blocks – 1 Avg Replication - 8

Table 2 – Distribution of Audio Directory

Genre

File Name	Size	Format	Data Type	Storages in HDFS
MAGD	11.1 M / 88.7 M	TSV	String	Total Blocks – 1 Avg Replication - 8
MASD	8.4 M / 67.3 M	TSV	String	Total Blocks – 1 Avg Replication - 8
topMAGD	10.6 M / 85.0 M	TSV	String	Total Blocks – 1 Avg Replication - 8

Table 3 – Distribution of Genre Directory

Main / Summary

File Name	Size	Format	Data Type	Storages in HDFS
Analysis / Metadata	55.9 M/ 447.5 M 118.5 M/ 947.7 M	CSV.GZ	String	Total Blocks – 2 Avg Replication - 8

Table 4 – Distribution of Main/Summary

Taste Profile

File Name	Size	Format	Data Type	Storages in HDFS
Mismatches	2.0 M / 16.2 M	Txt	String	Total Blocks – 8 Avg Replication - 8
Triplets.tsv	488.4 M / 3.8 G	TSV	String	Total Blocks – 2 Avg Replication - 8

Table 5 – Distribution of Taste Profile

Count the number of rows in each of the datasets.

File Name	Count
Audio - Attributes	3929
Audio - Features	12927867
Statistics	992865
Main - summary	2000000
Taste Profile – Triplets.tsv	48373586
Mismatches	20032

Table 6 – Row count of datasets

The main summary data set contains 2000000 rows. 998964 unique songs can be seen. This shows that the dataset includes additional data points more than just unique songs. This can happen because of the different versions or duplication of some songs. The TasteProfile data set consists of many rows – 48373586 compared to the number of unique songs.

Q2

Mismatches

The Million Song Dataset (MSD) is essential for machine learning tasks and music recommendations. However, there is a difference between track ID in MSD and song ID in the Taste Profile Dataset. Therefore, addressing these mismatches is essential, as well as enhancing model performance and ensuring data integrity. There are two ways of identifying mismatches. One manually accepts mismatches, and the other automatically identifies mismatches. Manually accepted mismatches are reviewed and confirmed manually, while automatically identified mismatches are detected through an automated process. First, load both manually accepted and automatically identified mismatches to handle those mismatches. Then, those mismatches were parsed and structured for further processing. This involved extracting suitable fields like ‘song_id, ‘song_artist’, ‘song_title’, etc. Thereafter, the triplets dataset was filtered to remove mismatches consisting of user-song play counts. Performed anti-join operation to remove automatically identified mismatches from the manually accepted ones. Also, anti-join was performed to remove the rest of the mismatches from the triplets dataset.

Then got the results as,

- Manually accepted mismatches: 488
- Automatically Identified mismatches: 19,094
- Original Triplets Count: 48,373,586
- Triplets Without Mismatches: 45,795,111

By doing this, we increased the dataset's accuracy and quality. This preprocessing step is essential to maintaining the data's integrity and helping create the best model for further analysis.

Audio/ attributes

First, I searched the main/summary directory and found two files, analysis.csv.gz and metadata.csv.gz. Both are compressed files. Thereafter, I checked how datasets are structured and found a bunch of columns inside each dataset. Then, I checked how many lines were in each dataset and found 10000000 (one million) tracks and 10000000 (one million) songs in both data sets. This means it has exacted the same number of tracks and songs in the original dataset.

Audio/ features consist of the number of CSV file directories. Inside one directory, it has 8-part files. (The number of files inside can be changed according to the directory file). Names of the files are consistent in both features and attribute directories. Using the Python list function, I made a list of all attribute files and converted whatever the data type name provided. There is consistency in the labelling of the attribute files. Then, I tested one of the datasets and found it was really hard to deal with some columns. Therefore, it simplified the schema to be easy to deal with.

Finally, most of the tracks were represented in this dataset, but not exactly one million datasets.
e.g.,
msd-jmir-area-of-moments-all-v1.0: 994623, 20
msd-jmir-lpc-all-v1.0: 994623, 20
msd-jmir-methods-of-moments-all-v1.0: 994623, 10

According to the results obtained in the simplified dataset, we can easily find the count of each dataset and how many columns each dataset consists of. This is probably helpful for choosing whichever dataset is suitable for further analysis. For instance, we can select the “msd-jmir-methods-of-moments-all-v1.0: 994623, 10” data set for our further analysis because it consists of the lowest number of columns. While doing analysis, it helps to reduce the time consumed in loading and can reduce the cost.

Audio Similarity

Q1

Here, I have chosen one of the data sets of feature files in the Audio directory. Choose file - msd-jmir-mfcc-all-v1.0.csv.
This file consists of 26 variables, including one string-type variable and the rest of 25 double-type variables. This audio feature file includes 994623 rows. When computing the descriptive statistics, choose all 25 double-type variables except string-type variables. After computing descriptive statistics, correlation analysis was performed. The correlation plot shows some highly correlated features. High positive values (close to 1) indicate a strong positive correlation, high negative values (close to -1) indicate a strong negative correlation and values close to 0 indicate no linear correlation. E.g. If we consider highly correlated variables have more than 0.70 thresholds, _c3 and _c4 (0.74), _c12 and _c11(0.83), _c10 and _c11(0.84), etc.

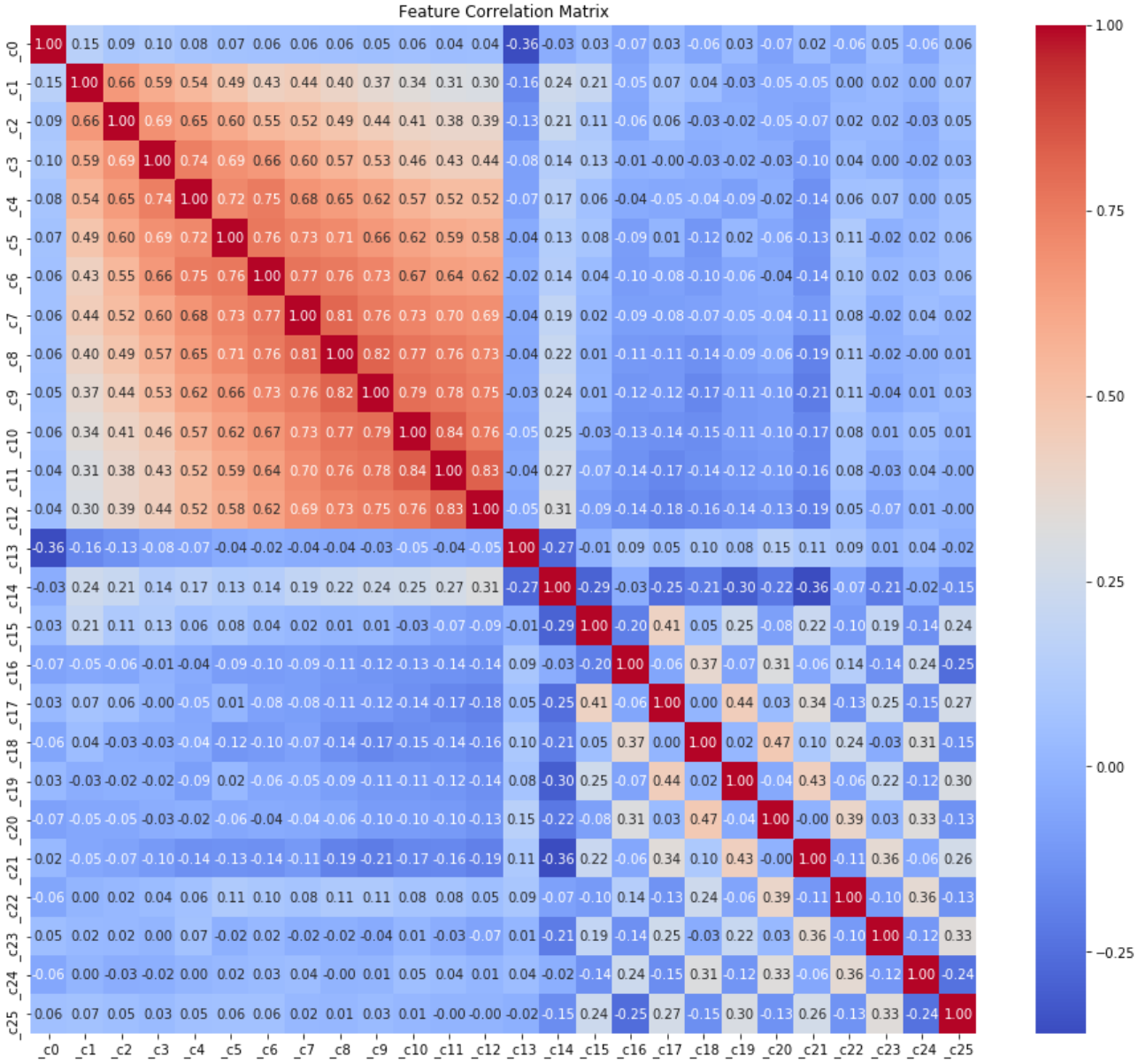


Figure 1 – Feature Correlation Matrix

From the genre directory, select only the MAGD data set. The data set consists of two string-type variables, track_id and genre.

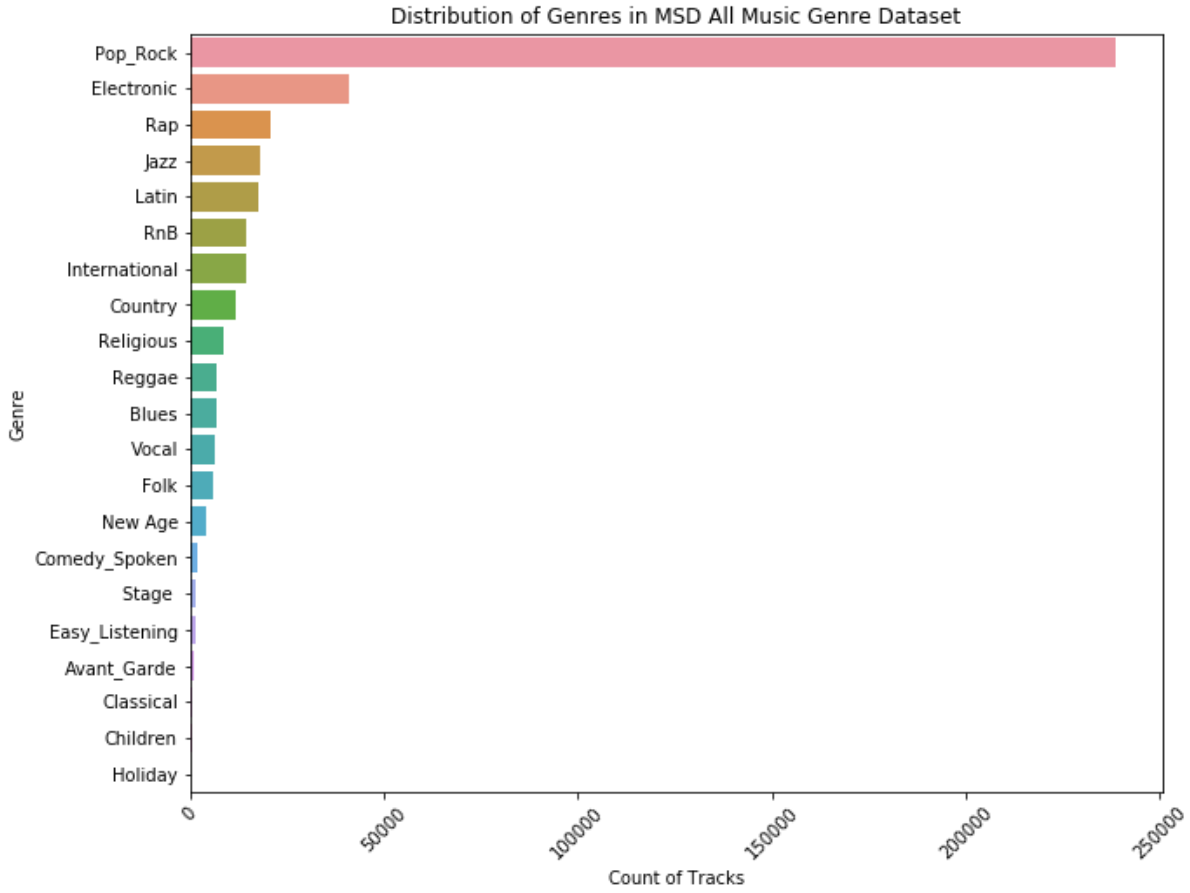


Figure 2 – Distribution of Genres in all music datasets (MSD)

Q2

I have selected three classification algorithms from the binary classification model using Spark's MLlib.

1. Logistic Regression
   - Explainability and Interpretability: Logistics regression is highly interpretable. Using the coefficients of the model, feature predictions can be easily made.
   - Predictive accuracy: It performs well, especially when relationships between features are approximately linear.
   - Training speed: efficient to large datasets.
   - Dimensionality: It performs well for a moderate number of features, but handling high dimensionality requires feature selections.
   - Use case: Its coefficient provides which features are more important for genre classification.
2. Random Forest Classifier
   - Explainability and Interpretability: Random Forest has good interpretability. This provides an understanding when making predictions to select the most influential features.

- Predictive accuracy: Due to its ensemble nature, it is highly accurate in overfitting.
- Training Speed: When the number of trees increases, this is slower than logistic regression.
- Dimensionality: Handle high-dimensional spaces.
- Hyperparameter Tuning: Parameters like depth of trees, number of trees, etc, need tuning.
- Use Case: Better for handling non-linear relationships.

3. Gradient-Boosted Tress (GBTs)
    - Explainability and Interpretability: This is similar to Random Forest.
    - Predictive Accuracy: Provide much better accuracy than Random Forest
    - Training Speed: Can be slower than Random Forest
    - Dimensionality: Effective in high–dimensional spaces
    - Hyperparameter Tuning: Requires tuning parameters.
    - Use Case: This will be useful if you require high predictive performance.

Before training each model, we can pre-process things according to the descriptive statistics.
Before training the model, it is important to know whether or not the dataset consists of missing values. Missing values can lead to an inaccurate model. If the data set contains missing values, we can impute them using mean or mode or according to the values in the column. The second thing is the scaling feature into the standard range because, from descriptive statistics, the obtained feature values can have different ranges and scales. This is mainly because models like Logistic Regression and Gradient boosting can be highly sensitive to the scale of the features.

After converting the Genre column into the Binary column, I got the result as
Class ‘1’ (Electronics): 40666 songs
Class ‘0’ (Other): 379,954 songs
The proportion of class ‘1’ is approximately 9.67% (40666/ 420,620(Total)), and the proportion of the Other Genre is approximately 90.33%. That means the dataset is imbalanced. It is imbalanced with most of the class ‘0’ – ‘Other Genre’ than class ‘1’ – ‘Electronic’. This can lead to bias when modelling because the model can be biased towards predicting the majority class (‘Other Genres’). To handle the class imbalance, we can use resampling techniques.

First, I did the stratified random sampling to split the data set into train and test. This random sampling ensures that training and testing have the same class distribution as the original dataset. From this sampling, I got 333,495 for the training set, 32,532 for ‘Electronics’ and 303,963 for ‘Other’ genres. For the test set, I got 8413 for ‘Electronics’ and 75,991 for ‘Other’ genres. As for the resampling techniques, I have conducted down-sampling, up-sampling, and observational weighing. The down-sampling technique reduces the number of samples in the majority class to match those in the minor class. According to my results, the training set becomes balanced with a 1:2 ratio. This helps to model a balanced data set, but it can pave the way to missing the useful data from the majority class. The up-sampling technique increases the number of samples in the minor class until it matches the major class. According to my results, the training set becomes balanced with the total sample. This is a good technique but can lead to overfitting. Observation weighting assigns

higher weights to the minor class and lower weights to the majority class. According to my result, this approach was used to prevent overfitting of the sample and loss of information. Overall, I think up-sampling and observation weighting are more appropriate techniques because detecting minor classes is crucial ('Electronic'). Down-sampling can be a good approach for a balanced performance across the classes.

	Down-Sampling	**Up-Sampling**	**Observation Weighting**
Logistic Regression	Precision: 0.37089 Recall: 0.50135 Auroc: 0.80023	Precision: 0.57686 Recall: 0.21821 Auroc: 0.79955	Precision: 0.09717 Recall: 0.99791 Auroc: 0.794891
Random Forest Classifier	Precision: 0.43107 Recall: 0.37484 Auroc: 0.78651	Precision: 0.73959 Recall: 0.05901 Auroc: 0.76279	Precision: 0.81420 Recall: 0.01831 Auroc: 0.769947
Gradient Boosted Trees	Precision: 0.37925 Recall: 0.472461 Auroc: 0.81074	Precision: 0.37925 Recall: 0.47246 Auroc: 0.81074	Precision: 0.37925 Recall: 0.47246 Auroc: 0.81075

Table 7 – Classification algorithm performance with resampling techniques.

Overall Classification Algorithm Performance

Logistic Regression:

This gives a reasonable balance performance with precision and recall. However, the result depends on the resampling method. Overall, this indicates making a good model.

Random Forest:

This resulted in higher precision in over-sampling and observation weighing compared to logistic regression. It handles the non-linear relationships well.

Gradient Boosted Trees:

It shows robust performance across different resampling methods. It has stable Precision, Recall, and Accuracy. Overall, it showed better performance with a high AUROC value.

Impact of class balance on algorithm performance

Precision and Recall Trade-off: In an imbalanced dataset, models will have a high number of false negatives and false positives. Oversampling and observation weighing techniques focus on balancing the trade-off. Accuracy can mislead the imbalanced dataset. Even if the majority class predicts correctly, the accuracy is shown as high in value. Also, the resampling methods impact class balance.

Q3

I tried to do hyperparameters and cross-validation to my chosen classification algorithm. However, it was time-consuming, especially in random forests and gradient boost trees. In Logistic Regression, the result shows that 'regParam' is 0.0,

elasticNetParam is 0.1. In performance metrics, it gives an accuracy of 0.80287, a precision of 0.4064, and a recall of 0.80828. This indicates that a low regularisation was optimal. For the random forest, the expected impact of the hyperparameter tunings is increasing the number of trees (improves the performance) and adjusting 'maxDepth' to balance the bias and variance. For Gradient Boost Trees, 'maxDepth' controls the complexity of each tree. Maximum iteration can increase accuracy, and small step sizes can improve performance.

How cross-validation works

The complete data set is split into equally sized (k) folds. The model is trained 'k' times. K-1 folds are used for training for each iteration. The remaining fold is used for validation. Each iteration's performance metrics such as accuracy, precision, and recall are computed on the validation set. After identifying the best hyperparameter, the model was again retrained.

Why cross-validation is used for hyperparameter tuning.

Cross-validation gives the best accurate model performance on unseen data compared to a simple train and test split. This is called a generalisation. Cross-validation helps to reduce bias and variance. It also helps to identify the optimal set that gives the best performance by allowing systematic experimentation with different hyperparameter values. Cross-validation highly involves model stability.

Steps for hyperparameter tuning

1. Define the model and hyperparameters
   First, we must choose the classification algorithm. Then, identify the hyperparameters that need to be tuned.
2. Define the parameter grid (example)
   For logistic regression: 'regParam' – [0.0,0.01,0.11.0], 'elasticNetparam' - [0.0,0.5,1.0]
   For Random Forest: 'numTrees'- [10,50,100], 'maxDepath – [5,10,20]
   For GBT: 'numIterations' – [10,50,100], 'learningRate '- [0.01,0.1,0.2], 'maxDepth'- [3,5,10]
3. Select Evaluation Metric
   Choose a suitable metric to evaluate model performance. (e.g., accuracy)
4. Perform cross-validation
5. Select the best hyperparameter
6. Train the final model

Logistic Regression gives an accuracy of approximately 80%, and hyperparameter tuning might improve the accuracy by around 80%- 85%. Likewise, Random Forest might improve the accuracy by 85%—90%.

Although fine-tuning hyperparameters is crucial, feature engineering frequently yields significant gains in model performance. Creating new features, altering already-existing features, or choosing the most relevant characteristics are all part of feature engineering. Because feature quality directly affects the amount of data available for the model to learn from, it can occasionally produce better results than

hyperparameter optimisation alone. Improved recall, accuracy, precision, and general robustness of a model can be achieved by efficient feature engineering. An essential stage in machine learning model optimisation is hyperparameter tuning. Through cross-validation, we may systematically experiment with various parameter values to determine the optimal settings for our algorithms. However, since enhancing the quality of the input features might have an even greater influence on model performance, feature engineering should not be disregarded. weighing the two strategies equally

Q4

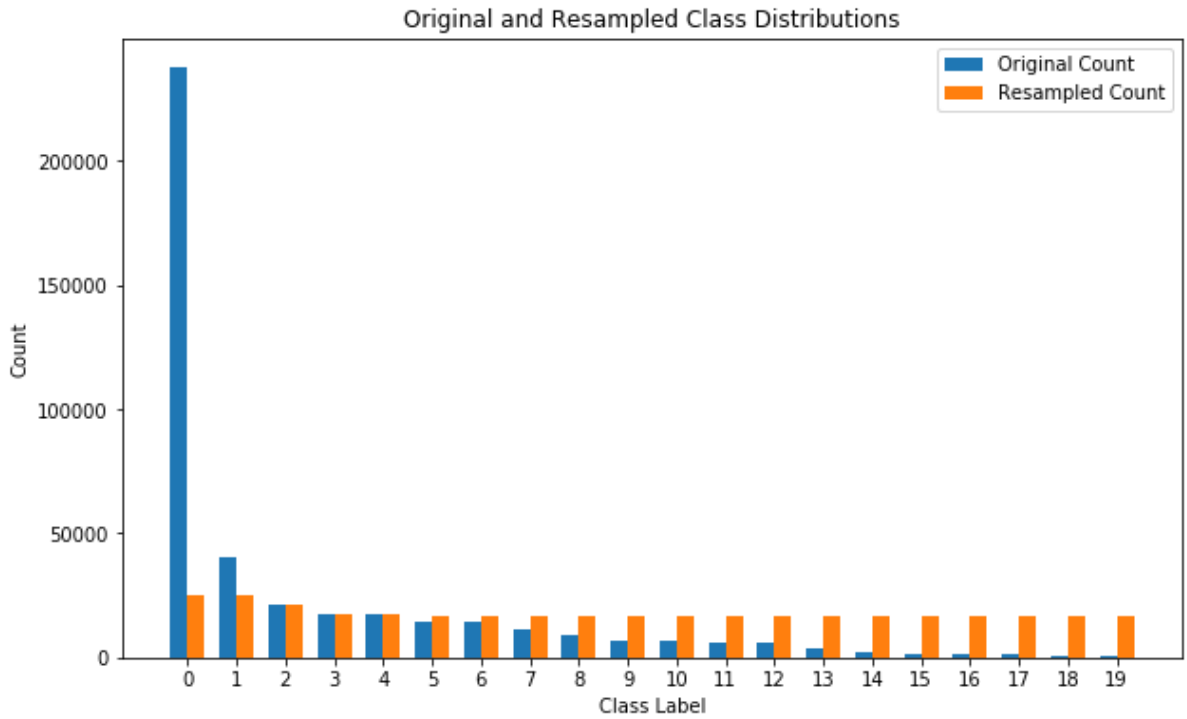
Multiclass Classification

Resampling and Dividing Data

I found a significant class imbalance while obtaining our dataset ready for multiclass classification. We balanced the class distributions by implementing a per-class sampling technique to resolve this. With 237,649 occurrences, class 0 dominates the initial class distribution, as seen in the plot, while other classes have much fewer instances—some classes have less than 1000 instances.

I resampled the classes to ensure an even distribution to decrease this imbalance. The resampled counts for each class in the figure are more balanced than the original counts, demonstrating the resampling effect. This resampling technique aimed to provide the model with a more evenly distributed training set, enhancing its performance. Because of its flexibility and capacity to handle multiclass issues well, we selected the Random Forest classifier for the multiclass classification task. A training set (80%) and a test set (20%) were generated from the dataset. The training set was used to train the Random Forest model, and the test set was used to evaluate it.
The model evaluation metrics showed,
Accuracy: 0.5725
F1 Score: 0.4257
Weighted precision: 0.4281
Weighted Recall: 0.5725

The model's performance has been significantly impacted by handling class imbalance and including many genres. With an accuracy of 0.5725, the model accurately predicts the genre 57.25% of the time. Even though this is moderate, it's crucial to consider the task's complexity and the initial class imbalance. The model's performance is balanced, considering both false positives and false negatives, as indicated by the F1 score of 0.4257, which is the harmonic mean of precision and recall. A weighted precision of 0.4281 is the average precision over all classes. This score shows that, on average, 42.81% of the time, the model is correct when predicting a genre. With a weighted recall of 0.5725, the model can accurately identify 57.25% of real cases for each genre on average.



Plot 1 – Original and resampled class distribution

The resampling technique provided a more balanced dataset, which increased the model's capacity to generalise across various classes. However, the model's performance indicators indicate that there may still be space for improvement despite these efforts. The model does rather well, although it has trouble with some of the less common classes, as evidenced by the modest accuracy and F1 score. Model performance is strongly impacted by the class balance, particularly in multiclass classification tasks where certain classes are underrepresented. I attempted to address this problem by resampling; nevertheless, accurate classification is still difficult because of the intrinsic complexity of genres and the possibility of overlap.

In conclusion, the development of several genres and the attempts to distribute the classes uniformly have made the classification assignment more realistic and difficult. Although the model performs moderately, higher accuracy and better overall performance may require more feature engineering improvements, complex resampling methods, or advanced models.

Song Recommendation

Q1

Repartition and cache Taste Profile

Optimising data processing for collaborative filtering is essential based on the size and complexity of the Taste Profile, which consists of a ‘triplets.tsv’ file including

48,373,586 rows and a total size of 3.8 GB. To enhance the parallel processing efficiency, we can repartition the dataset. The ideal number of partitions was calculated using the 4 * N * M formula. N indicates the number of executor instances, and M indicates the number of cores per executor. This gives the result as 32 partitions and shows the data is evenly distributed across the cluster nodes.

The data was cached to enhance the processing efficiency further. By keeping the dataset in memory after the initial computation, caching significantly reduces the period needed for subsequent operations. This is particularly essential for model training for collaborative filtering, which must review the data several times.

We ensure that the dataset is managed successfully by implementing these enhancements into practice, which minimises computation time and resource consumption. This is crucial for managing big datasets like the Taste Profile.

Unique Songs and Users

We examined the overall number of unique songs and unique users in the Taste Profile dataset to understand its characteristics prior to training the collaborative filtering algorithm. To tidy up the dataset, we eliminated the original 48,373,586 rows. This produced a corrected dataset that, after 2,578,475 mismatched entries were removed, had 45,795,111 rows.

Unique Songs: 378,548
Unique Users: 1,019,318

These unique counts define the depth of the collaborative filtering model, which is essential for developing a successful recommendation system. By managing mismatches, the dataset's consistency for training and evaluation was improved, and only real song-user interactions were taken into account. Knowing the unique objects in the dataset can help me better understand the recommendation system's complexity and potential, and we can then modify our strategy to handle a high volume of unique interactions effectively.

Most active user and percentage

I examined the number of unique songs played by the most active users and their importance to the total dataset to understand user behaviour, particularly engagement.

Most active user: The user with ID ‘093cb74eb3c517c5179ae24caf0ebec51b24d2a2’ played a total number of 13,074 times.

Unique song played: 195

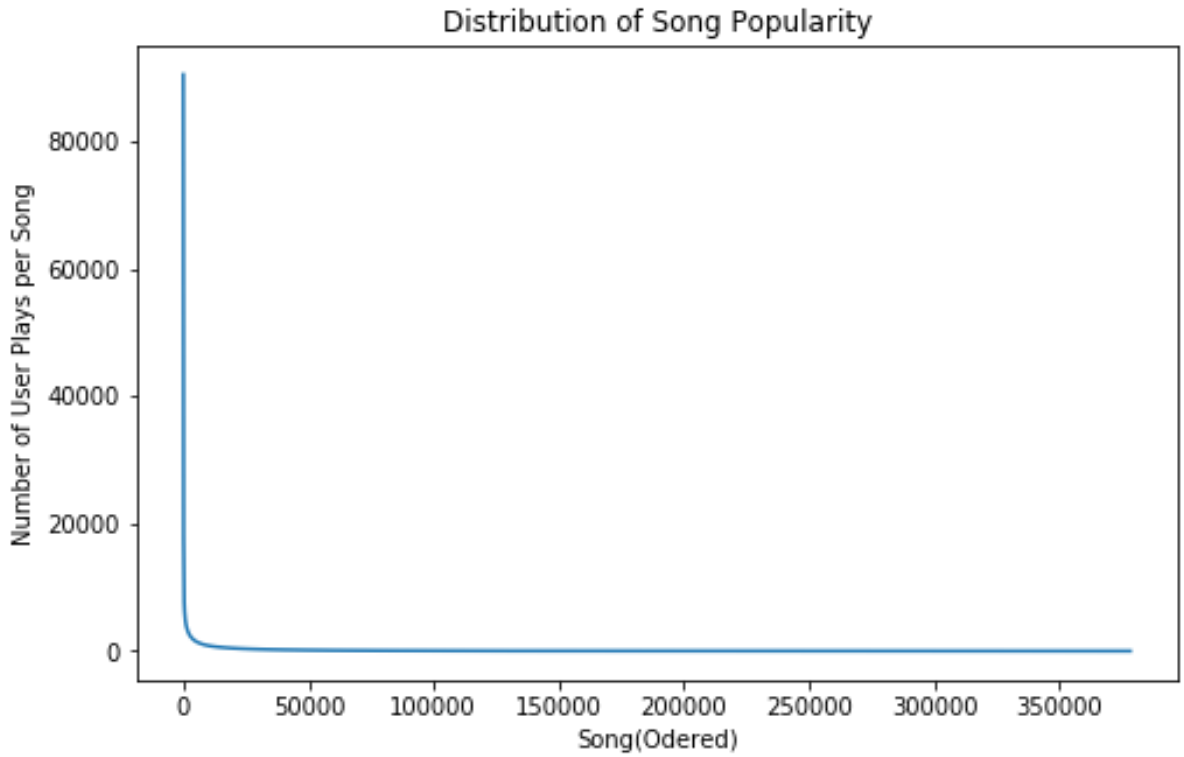
Percentage of unique songs: 0.052% of the unique songs are available in the dataset. This represents a relatively small fraction of all unique songs accessible in the dataset and suggests that even the most active users only play a small portion of the collection.

This research focuses on users' favoured portions of the song collection, highlighting patterns of user engagement and allowing for the creation of personalised suggestions.

Visualisation of the distribution of song popularity and user activity

We used the distributions of song popularity and user activity better to illustrate the features of the Taste Profile dataset. These graphics make it easier to understand how frequently various songs are played and how involved various users are in selecting a wide variety of music to listen to.

Song Popularity

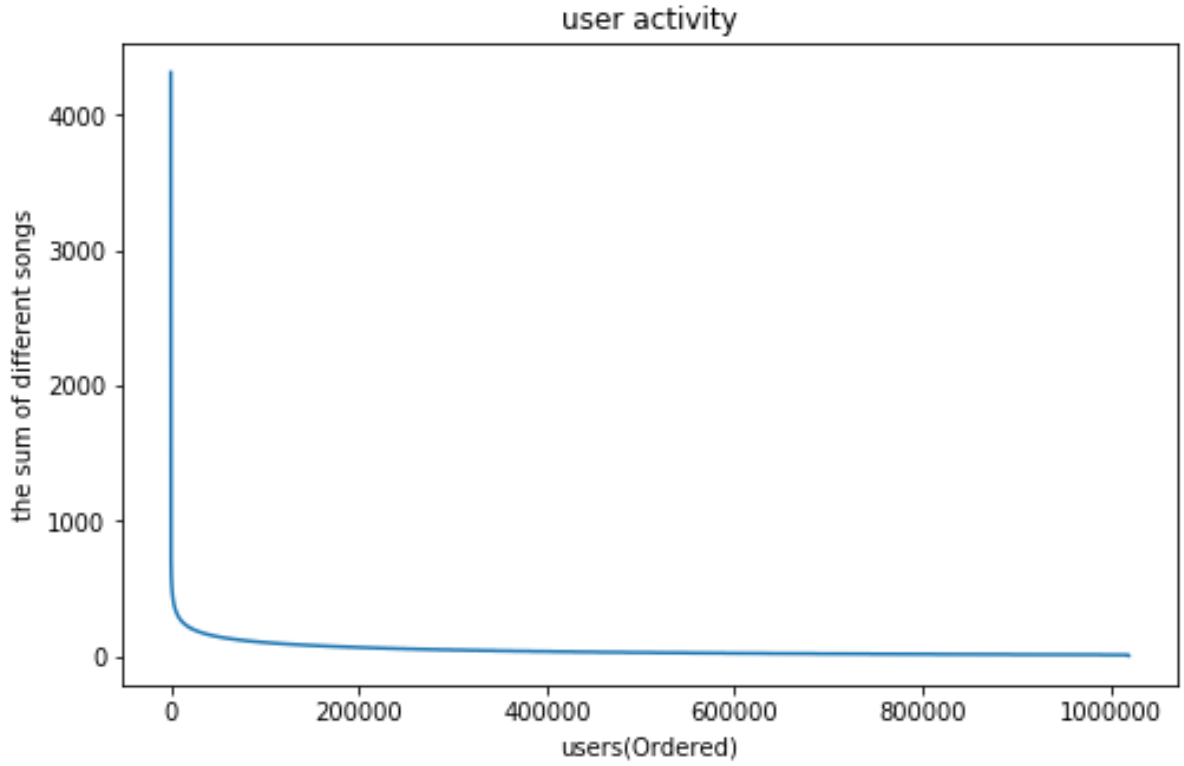


Plot 2 – Distribution of Song Popularity

The X-axis represents the order of the individual songs by popularity, and the Y-axis represents the number of users who play each song.

The distribution of song popularity shows a ‘heavy tail’ distribution. Head indicates the play counts for a select few songs are extremely high. These are the most widely played songs among users, with a large number of them. Tail indicates that there aren't many plays for most of the songs. A small percentage of people listen to and play these songs, which are not as well-known. The dataset's heavy tail suggests that, despite a few extremely well-known songs, most songs are only played infrequently. This feature is common in datasets about media use, including music streaming.

User Activity



Plot 3 – Distribution of user activity

In this plot, the X-axis represents individual users ordered by their activity level, while the Y-axis represents the number of different songs each user has listened to. The visualisation's distribution shows ‘heavy tail’ patterns.

Head indicates a small proportion of people like a large musical selection. These are the most frequent users, going through a lot of the collection. Tail indicates that most users listen to only a small selection of songs. These are fewer frequent users who may just listen to a few songs they enjoy. According to this distribution, most users are less active, which leads to a highly skewed interaction pattern, while a small percentage of users are very active and greatly contribute to the diversity of interactions.

Both distributions' heavy tails’ indicate a user-item interaction matrix with high sparsity, which is typical of datasets used for collaborative filtering. Due to this sparsity, the model may find it difficult to locate sufficient interaction data to provide accurate suggestions for songs and users who are less active. The interaction data is dominated by popular products and active users. This may cause the model to be biased in favour of suggesting well-liked products to users who engage with it frequently. This bias can be lessened with the use of regularisation, weighted sampling, or hybrid models. To guarantee a diverse list of recommendations, the model must be trained to examine the long tail and provide individualized recommendations in addition to the most popular products.

Q2

Collaborative filtering data preparation

To create a strong collaborative filtering model, I must first clean the dataset by removing less useful data points. This entailed eliminating people who had only listened to a small number of tracks and songs with extremely few plays. Since these data points don't add to our understanding of user preferences or song popularity patterns, they are of little help in collaborative filtering.

As the first step, I have analysed the distribution. First, we looked at the distributions of user activity and song popularity. These distributions' statistical summaries shed light on how plays and listens were distributed among songs and users, respectively. Song Popularity obtained a median of 13 plays and a 75th percentile value of 50 plays; the average number of plays per song was 121.05. In User Activity, it obtained a median of 26 songs and a 75th percentile value of 53 songs; the average number of songs per user was 44.93.

When applying filters, I selected reasonable numbers for N and M to filter the dataset based on these distributions. Songs that had been played less than 121 times were eliminated, and users who listened to less than forty-five songs were eliminated. In the process of filtering, I utilised these selection standards based on user activity and song popularity information. This produced 55,648 tracks with over 121 turns each and retained 298,388 individuals who had each listened to over 45 songs.

I used ‘String Indexer’ to turn the song IDs and user IDs into numerical codes to prepare the data for model training. This change made it easier to combine datasets effectively. I ensured only the filtered songs and people were included in the final dataset by combining the filtered song and user datasets with the original triplets dataset. The user-song play interactions in the final dataset, ‘triplets_merge’, match the filtering criteria, improving the data quality for collaborative filtering. This preprocessing phase must be followed to improve the collaborative filtering model's ability to produce accurate suggestions and guarantee that the model is trained on meaningful and useful data.

Split data set into train and test

I divided the dataset into training and test sets according to specified requirements to efficiently train and test our collaborative filtering model. A minimum of 25% of the total play counts must be represented in the test set, and each user in the test set needs to have interacted with songs in the training set. I have made predictions from collaborative filtering models based on past interaction data to ensure user presence in the training set. The model cannot produce suggestions for a user in the test set if

there are no interactions for that user in the training set. For reliable predictions, it is necessary to ensure that every user in the test set has interactions in the training set.

25% of the total play count was calculated to determine the threshold, and 17,658,957.5 was obtained. This limit assures that a sufficient amount of the data for a robust evaluation is present in the test set. An initial random split was done to divide the dataset into 75% training and 25% test sets. This ensures unbiased sampling. To adjust the test set, the first set's overall play count, roughly 17,685,403, satisfied the requirements. I would have shifted more rows from the training set to the test set until the threshold was reached if the play count had been lower.

I made sure that users in the test set interacted with users in the training set during the adjustment. To achieve this, more rows from the training set were chosen randomly, added to the test set, and then taken out of the training set. Unbiased sampling was ensured by employing the orderBy(rand()) method to keep the selection process random. As a result, a final set of data produced a test set that comprises 25.037% of the total number of plays. Both training and test sets are used so that the collaborative filtering model accurately predicts each user for whom they interact in the training set. The careful dataset preparation guarantees that our collaborative filtering model may be evaluated properly and trained on significant data, resulting in dependable user recommendations.

Use Alternating Least Square (ALS) algorithm

While dividing the dataset into training and test sets, it was essential to ensure that each user in the test set interacted with other users in the training set. The ALS algorithm picks up user and item characteristics from previous interactions, which explains why. The model cannot provide recommendations for a user in the test set if there are no interactions for that user in the training set, which prevents the model from producing reliable predictions.

Building recommendation systems with the ALS algorithm is valuable, particularly when working with large-scale implicit feedback datasets such as the one utilised. Through precise dataset preparation and reliable training procedures, we may create a model that offers users insightful and precise recommendations**.**

Compare these recommendations to the songs the user has actually played.

I used the collaborative filtering technique to generate music recommendations for user '104706.0'. We then compared these recommendations with the songs the user had actually played. The table of comparisons is as follows:

Played Song ID	Played Title	Played Artist	Recommended Song ID
42442	Mr.Wendal (edited)	Arrested Development	44044
10138	Namaste	Beastie Boys	34562
52150	Isopropanol	Aphex Twin	30286
49135	BYS (Explicit)	Gang Starr	7211
47201	Puss N Boots	The New York Dolls	35890

Table 8 - Actual song vs recommended Song Codes

The collaborative filtering model produced a list of recommended songs for user '104706.0'. I tried to compare these recommendations with the user's real play history to assess the model's performance. However, a thorough comparison proved difficult because it might be my fault that I couldn’t find recommended song information or the suggested songs' information was absent. The user has listened to many artists, including The New York Dolls, Beastie Boys, Aphex Twin, Gang Starr, and Arrested Development. This implies that the user enjoys a variety of musical genres. Ideally, this diversity should be represented by the recommendations as well. However, it's difficult to determine whether the model has accurately captured this element because the recommended songs' information is lacking or my fault.

Utilising user behaviour data, collaborative filtering models identify trends and offer suggestions. The model's promise is demonstrated by its ability to produce a list of suggested songs, even with the limits on metadata. With the addition of thorough metadata, the model should be able to make more accurate and appropriate recommendations to the user's wide range of musical tastes. The user's past indicates that they have a taste for a mixture of rock, electronic, and hip-hop music. Songs from similar genres and performers with similar styles would ideally make good recommendations. Although the model's structure indicates it should be able to do so, we cannot say if the suggestions match these requirements without metadata.

Based on user behaviour data, the collaborative filtering technique shows great promise in music recommendation. Unfortunately, inadequate metadata (or my issue of not finding song details) currently restricts the usefulness of this approach by preventing thorough and insightful comparisons. Resolving this issue and adding detailed song information to the dataset will be essential to precisely evaluating and enhancing the recommendation system. These improvements allow the model to

more accurately represent user preferences and offer more varied and meaningful recommendations.

Compute the metrics

Results obtained,

- Precision @ 10: 0.04772
- Mean Average Precision (MAP): 0.01947
- NDCG @ 10: 0.05125

Precision @ 10
Precision @ 10 quantifies the percentage of relevant items—songs in this case—in the top 10 suggestions. It is calculated by dividing the total number of relevant items in the top 10 recommendations by 10. This measure aids in determining the proportion of suggested songs that are genuinely related to the user. A higher precision means that the model is better at suggesting appropriate songs. Precision @ 10 may overlook important information since it does not consider the ranking of relevant items that are not in the top 10.

Mean Average Precision (MAP)
The mean of each user's average precision score is called MAP. The average of the precision values obtained at the rankings where pertinent items are found is the user's average precision. MAP considers the order of the relevant items, offering a more comprehensive assessment of the performance of the recommendation system. It highlights the fact that recommendations that are more relevant and show up earlier in the list are more valuable. A user's mean score is provided by MAP, and users who engage frequently have the potential to distort this value. It might not exactly represent the experiences of individual users.

NDGC @ 10
By considering the placement of relevant items in the top 10, the Normalized Discounted Cumulative Gain (NDCG) @ 10 calculates the ranking quality of the suggestions. It explains why the things at the top of the list are more significant than those at the bottom. By rewarding models that place relevant items higher in the list, NDCG offers information about the standard of recommendation ranking. Although NDCG @ 10 is sensitive to item placement, it may not accurately reflect a user's pleasure if their preferences are varied or evolve.

Evaluation and Restrictions
Evaluation and Restrictions

Low Metric Values: Precision @ 10, MAP, and NDCG @ 10 have computed values that are comparatively low. This implies that the collaborative filtering model is not particularly good at placing highly relevant song recommendations at the top.

Collaborative filtering models frequently have a cold start issue, which causes them to perform poorly for new users or items with sparse data. This might account for some of the poor metric readings.

Sparse Data: The dataset's sparsity, comprised of many users with few interactions, can result in lower precision and MAP scores. This can also impact the model's capacity to produce reliable suggestions.

Alternative Methods

User Surveys and Input: Gathering qualitative user input can reveal users' satisfaction with the recommendations and their relevance.

A/B Testing: To gain useful insights into the efficacy of the models, conduct A/B testing using various recommendation algorithms and track user engagement metrics like click-through rates, play counts, and session duration.

diversity and Novelty Metrics: By incorporating metrics that evaluate the variety and novelty of recommendations, the recommendation system may guarantee that users receive a wide selection of new and different songs, improving their overall experience.

Additional metrics for future user-song play

Retention Rate: The number of users who stay in contact with the recommendation system over time can be used to gauge the recommendations' significance and long-term worth.

Repeat Play Rate: Monitoring how frequently people listen to the suggested music more than once might be used to determine how relevant and well-liked the recommendations are.

Metrics related to user engagement with the recommendation system, such as average session duration, number of songs played during a session, and interaction with suggested music, might offer valuable insights.

Conclusion

I carefully studied the Million Song Dataset (MSD) for this assignment, concentrating on applying collaborative filtering methods for music recommendation. I first used descriptive statistics, random sampling, and resampling to comprehend the data distribution and assess its features. This initial research helped in laying the foundation for more sophisticated modelling methods. I used the combined dataset to perform binary and multiple classifications based on the genre column in the Audio Similarity section. These categories improved our comprehension of the traits unique to each genre by illuminating the linkages and patterns seen in the data.

After eliminating less useful data points and transforming song and user IDs into numerical codes, I used the Alternating Least Squares (ALS) algorithm for the collaborative filtering model. The dataset was divided into training and test sets to ensure that users in the test set had interactions in the training set for accurate predictions. I generated suggestions and tried to compare them with the user's real play history, but I ran into problems because the recommended songs' metadata was missing. This made it difficult to compare the songs that were played and those that were suggested in detail.

Metrics like Precision @ 10, Mean Average Precision (MAP), and NDCG @ 10 were used to assess the model's performance, and the results showed that it was effective at ranking relevant song recommendations. Though helpful, these measurements also brought drawbacks, including data sparsity and the cold start issue. To enhance the assessment and efficacy of the recommendation system, I proposed substitute techniques such as user surveys, A/B testing, and metrics related to diversity and originality. Metrics like repeat play and retention rates may also offer more information about how well users interact with the recommendations.

This assignment gave me important insights into the possibilities and constraints of collaborative filtering models in music recommendation systems, even if it was difficult for me to thoroughly compare the songs that were played and those that were recommended. Future work will concentrate on enhancing the completeness of the data and investigating sophisticated evaluation methods to improve the model's functionality and user experience.

Reference

1. *Welcome! | Million Song Dataset*. (n.d.). http://millionsongdataset.com/
2. *Apache Spark™ - Unified Engine for large-scale data analytics*. (n.d.). https://spark.apache.org/
3. DataRobot, & DataRobot. (2023, July 31). *Multiclass classification in machine learning*. DataRobot AI Platform. https://www.datarobot.com/blog/multiclass-classification-in-machine-learning/
4. *chatGPT4*. (n.d.). chatGPT. https://chat.openai.com
5. *Grammarly: free AI writing assistance*. (n.d.). https://www.grammarly.com/

Appendix

Appendix A: Data

Data – hdfs:///data/msd

Data is collected from The Million Song Dataset

Appendix B: Data Preprocessing and Audio Similarity – Notebook

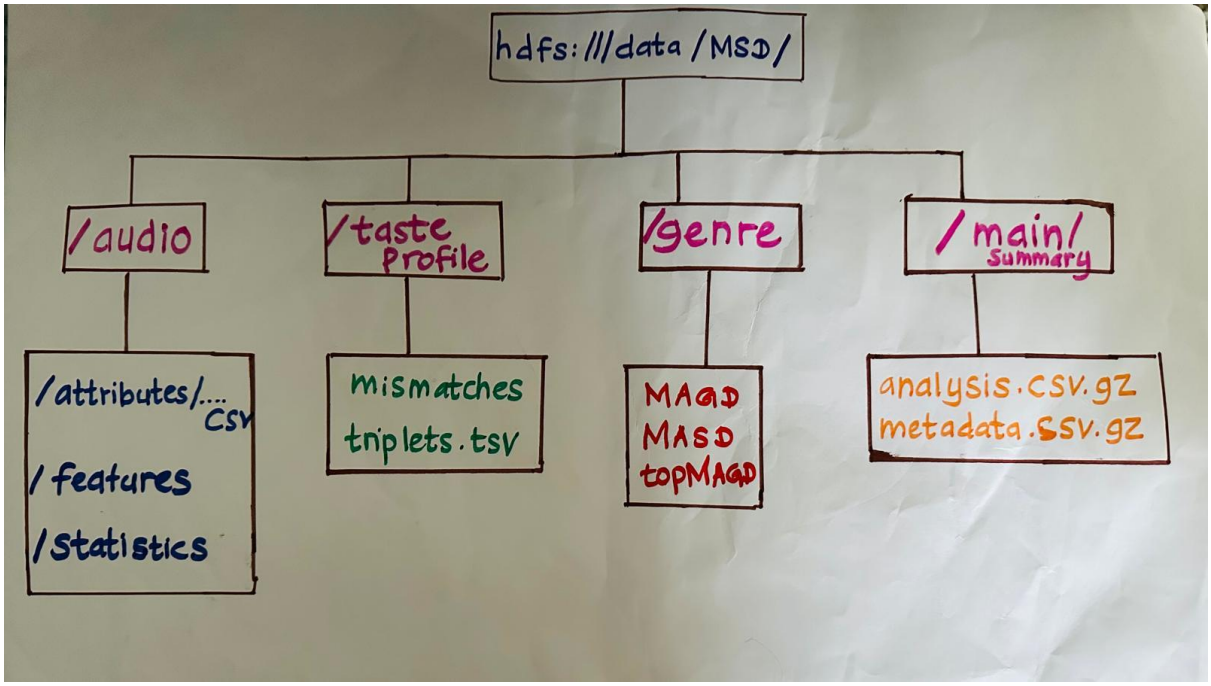


Figure1 – MSD Structure

Appendix C: Song recommendation

This notebook consists of collaborative filtering and song recommendation