REPORT

ASSIGNMENT 2

OCTOBER 23, 2024

SHAHRON SHAJI
51781227

TABLE OF CONTENTS

1. BACKGROUND

The **Million Song Dataset (MSD)**, developed by Columbia University's LabROSA, is a large-scale dataset designed for music recommendation and data mining research. It contains metadata and various audio-related attributes for a million songs. However, it excludes actual song content such as lyrics, full audio, or user preferences beyond play counts.

Initiated by The Echo Nest and LabROSA, MSD offers a unique opportunity to analyze and compare music, as well as to predict song attributes based on extracted features. The dataset includes various song-level information such as artist IDs, track IDs, and song features like loudness, tempo, and time signatures. We also dive into additional datasets, such as the MSD Allmusic Genre Dataset (MAGD), which offers genre labels for songs, and the Taste Profile dataset, which contains user-song play counts.

The goal of this assignment is to explore the datasets at a high level, preprocess the audio features, and develop models to predict song genres based on their audio attributes. This project involves a step-by-step process to fully understand the structure of these datasets and use them to build predictive models. It begins with a high-level exploration of the dataset structure, schema inference, and row counts. Later, preprocessing steps, including schema construction and column renaming for the audio datasets, are undertaken to make the data more manageable. This groundwork sets the stage for model training and analysis, involving binary and multiclass genre classification using Spark's MLlib. Finally, it will build a collaborative filtering model using user-song play data to recommend songs to users.

By the end of this assignment, we will have gained insights into the MSD structure, feature extraction methods, and the design and evaluation of machine learning models for music classification and recommendation tasks.

2. DATA PROCESSING

2.1. Q1

The Million Song Dataset (MSD) is organized into several directories within HDFS, each containing distinct types of datasets related to audio features, genre classifications, and user taste profiles. The dataset located at hdfs:///data/msd/ is organized into four main directories: audio, genre, main, and tasteprofile. Each directory serves a distinct purpose in the Million Song Dataset (MSD). The audio directory, which is the largest, contains various subdirectories, including attributes, features, and statistics. The attributes subdirectory hosts 13 files, mainly in CSV format, with sizes ranging from a few hundred bytes to over 34 KB. These files contain audio attributes related to different methods of moments and timbral analysis. The features subdirectory, significantly larger at over 12.2 GB. The statistics subdirectory contains statistical data related to the audio files in the format CSV.GZ, totalling about 40.3 MB. The genre directory encompasses genre classification information in three TSV files ranging from 8.4 MB to 11.1 MB, while the main directory had another folder inside of it named summary which hold core song metadata and analysis in CSV.GZ format with 118.5 MB and 55.9 MB respectively. Lastly, the tasteprofile directory contains user taste profiles related to music preferences which had a TSV file named triplets and another folder named mismatches with each being 488.4 MB and 2.0 MB respectively.

The comparison between the total number of songs and the unique songs in the Million Song Dataset (MSD) shows a slight redundancy, with only 998,963 unique songs out of 1,000,000 total entries. This indicates a small percentage of duplicates or variations within the dataset.

Below is a table summarizing the names, sizes, formats, data types, and the number of rows in each dataset under the hdfs:///data/msd/ directory:

Directory	Dataset Name	Size	Format	Data Type	Number of Rows
attributes	msd-jmir-area-of-moments-all-v1.0.attributes	1.0 KB	CSV	String	21
	msd-jmir-lpc-all-v1.0.attributes	671 B	CSV	String	21

	msd-jmir-methods-of-moments-all-v1.0.attributes	484 B	CSV	String	11
	msd-jmir-mfcc-all-v1.0.attributes	898 B	CSV	String	27
	msd-jmir-spectral-all-all-v1.0.attributes	777 B	CSV	String	17
	msd-jmir-spectral-derivatives-all-all-v1.0.attributes	777 B	CSV	String	17
	msd-marsyas-timbral-v1.0.attributes	12.0 KB	CSV	String	125
	msd-mvd-v1.0.attributes	9.8 KB	CSV	String	421
	msd-rh-v1.0.attributes	1.4 KB	CSV	String	61
	msd-rp-v1.0.attributes	34.1 KB	CSV	String	1441
	msd-ssd-v1.0.attributes	3.8 KB	CSV	String	169
	msd-trh-v1.0.attributes	9.8 KB	CSV	String	421
	msd-tssd-v1.0.attributes	27.6 KB	CSV	String	1177
features	msd-jmir-area-of-moments-all-v1.0	65.5 MB	CSV.GZ	String, Double	994623
	msd-jmir-lpc-all-v1.0	53.1 MB	CSV.GZ	String, Double	994623
	msd-jmir-methods-of-moments-all-v1.0	35.8 MB	CSV.GZ	String, Double	994623
	msd-jmir-mfcc-all-v1.0	70.8 MB	CSV.GZ	String, Double	994623
	msd-jmir-spectral-all-all-v1.0	51.1 MB	CSV.GZ	String, Double	994623
	msd-jmir-spectral-derivatives-all-all-v1.0	51.1 MB	CSV.GZ	String, Double	994623
	msd-marsyas-timbral-v1.0	412.2 MB	CSV.GZ	String, Double	995001
	msd-mvd-v1.0	1.3 GB	CSV.GZ	String, Double	994188
	msd-rh-v1.0	240.3 MB	CSV.GZ	String, Double	994188
	msd-rp-v1.0	4.0 GB	CSV.GZ	String, Double	994188
	msd-ssd-v1.0	640.6 MB	CSV.GZ	String, Double	994188
	msd-trh-v1.0	1.4 GB	CSV.GZ	String, Double	994188
	msd-tssd-v1.0	3.9 GB	CSV.GZ	String, Double	994188
statistics	sample_properties	40.3 MB	CSV.GZ	String	992866
genre	msd-MAGD-genreAssignment	11.1 MB	TSV	String	422714
	msd-MASD-styleAssignment	8.4 MB	TSV	String	273936
	msd-topMAGD-genreAssignment	10.6 MB	TSV	String	406427
summary	analysis	55.9 MB	CSV.GZ	String, Double, Int	1000000
	metadata	118.5 MB	CSV.GZ	String, Double, Int	1000000
tasteprofile	triplets	3.8 GB	TSV	String	48373586
mismatches	sid_matches_manually_accepted	89.2 KB	TXT	String	938
	Sid_mismatches	1.9 MB	TXT	String	19094

Table 2.1.: Summary of the datasets

The mismatches folder had two TXT files which ranged from a few KB to 1.9 MB. Below is an overview of the datasets located under hdfs:///data/msd/:

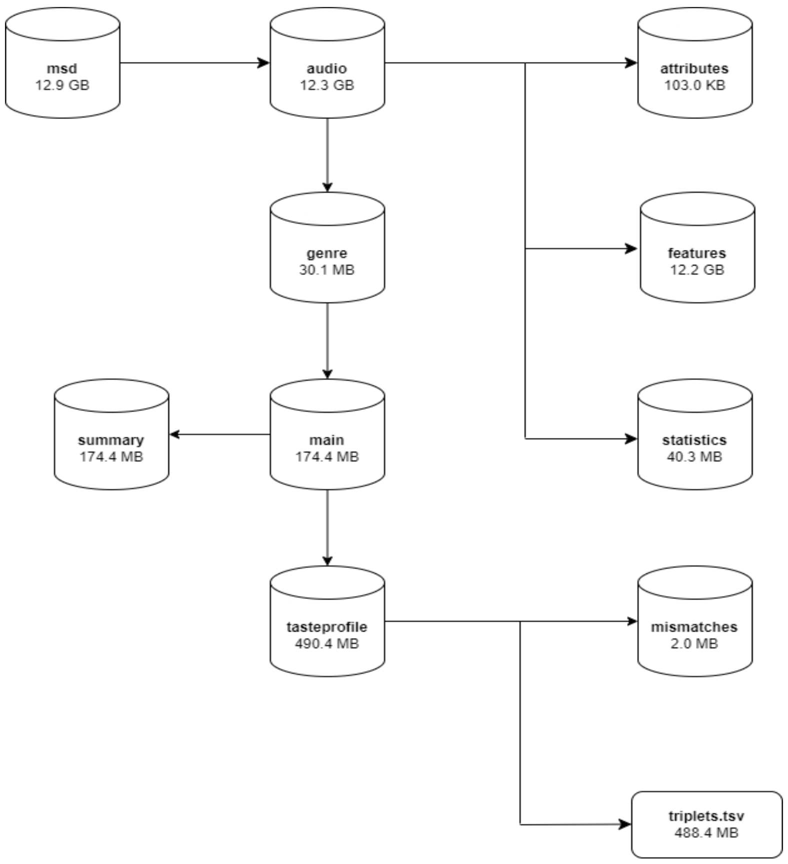


Fig 2.1.: Directory Tree

2.2. Q2

The audio attributes datasets contain metadata about the audio feature datasets, specifically listing attribute names, data types, and occasionally other descriptors. Each attribute dataset corresponds to a specific feature dataset, allowing for easy referencing and schema generation. By separating attributes from features, we enhance efficiency, as this approach allows for consistent schema generation across numerous files without duplicating metadata.

The process of automatically creating a StructType for the audio feature datasets involves using the metadata available in the corresponding attribute files. Firstly, the attribute metadata should be read and loaded. The attribute files contain columns with attribute names and types. These are read using Spark, where they are converted into a Pandas DataFrame for easier processing using spark.read.csv(path_attributes).toPandas(). This process is followed by mapping the retrieved data to Pyspark Data Types. A dictionary (type_column_dic) is used to map attribute types (e.g., real, float, string) to the appropriate PySpark data types such as DoubleType() and StringType(). This allows for automated creation of schemas based on the types listed in the attribute files. After which the Schema is created. The schema for each dataset is generated dynamically by iterating through the attribute names and types. For each column, a StructField is created using the mapped PySpark data types. These StructField objects are then combined into a StructType, representing the schema for the corresponding feature dataset. The iterrows() function loops through each row of the attribute DataFrame, extracting the column name and type, and creates a corresponding StructField. Then these are stored for feature loading, where it's stored along with the dataset's file path and name in a list called datasets. This list contains tuples of (dataset name, feature file path, schema).

Lastly, after the schemas are created, the column names are processed to make them more readable and concise. This is done using the rename_columns_for_all_datasets function, where specific patterns in the column names are replaced with shorter, intuitive names. Each dataset received a prefix based on its core type. The prefix was derived from a keyword in the dataset name, such as “LPC” for msd-jmir-lpc-all-v1.0. This helps in identifying the dataset origin when columns are merged, ensuring uniqueness and context clarity. Many column names contained descriptive phrases that were valuable but too lengthy for efficient use in coding or analysis. "Overall_Standard_Deviation" was renamed to "OSTDDEV", "Overall_Average" was shortened to "AVG" and "Method_of_Moments" was abbreviated to "MOM". Certain recurring patterns across columns were condensed into shorter, standardized terms to reduce redundancy and improve consistency. "Area_" was replaced with "A_", and "Spectral" became "SPEC_", Complex identifiers like "Mean_Acc5_Mean_Mem20" were simplified to "MAMM_", "HopSize512_WinSize512_Sum_AudioCh0" and "Power_powerFFT_WinHamming" were further condensed to unique identifiers, "HWSA" and "PPW", respectively, that maintained meaning while improving usability.

3. AUDIO SIMILARITY

3.1. Q1

The **msd-jmir-area-of-moments-all-v1.0** dataset was chosen for this analysis. This dataset provides detailed audio features based on waveform analysis using "Area of Moments," which captures nuanced aspects of a track's sound profile. It offers a broad range of continuous features, including average values (`A_MOM_AVG`) and standard deviations (`A_MOM_OSTDDEV`) across different moments, providing granular insights into the dynamics and structure of the audio. These rich waveform features can be instrumental in identifying genre-specific characteristics, making it a strong candidate for predicting genre based on sound properties alone.

Here is a table with the descriptive statistics for a sample of the audio features from the `msd-jmir-area-of-moments-all-v1.0` dataset:

Feature	Count	Mean	Std Dev	Min	Max
A_MOM_OSTDDEV_1	994623	1.23	0.53	0.0	9.35
A_MOM_OSTDDEV_2	994623	5500.46	2366.13	0.0	46860.0
A_MOM_OSTDDEV_3	994604	33817.89	18228.65	0.0	699400.0
A_MOM_AVG_1	994604	3.52	1.86	0.0	26.52
A_MOM_AVG_2	994604	9476.02	4088.52	0.0	81350.0
A_MOM_AVG_3	994604	58331.70	31372.66	0.0	1003000.0

Table 3.1.: Descriptive Statistics

These statistics indicate significant variation across different audio features, as seen in the large ranges and standard deviations. This variety suggests that the features likely capture distinctive audio characteristics, which could be useful in distinguishing between musical genres based on waveform analysis. Let me know if you'd like more detailed insights on any specific features or further statistical metrics. The descriptive statistics of the `msd-jmir-area-of-moments-all-v1.0` dataset reveal several key insights that will influence our model training and inference. First, there is significant variation in feature scales, with some features, such as `A_MOM_OSTDDEV_1`, ranging from 0 to 9.35, while others, like `A_MOM_OSTDDEV_10`, reach into the trillions. This disparity indicates that feature scaling (e.g., normalization or standardization) will be crucial to prevent high-range features from disproportionately affecting the model.

Additionally, many features exhibit high standard deviations relative to their means, suggesting the presence of outliers that could skew weight distributions, particularly in sensitive models like neural networks. Addressing outliers through detection and removal or employing robust scaling methods will be necessary. The analysis also points to potential redundancy, as some features may be linearly correlated, which could complicate training without adding significant predictive value. To streamline the model, dimensionality reduction techniques, such as PCA or feature selection, may be beneficial.

The Corrgram is a heatmap representation of a correlation matrix, which displays the relationships between various features. Correlation values range from -1 to 1, where a value of 1 signifies a perfect positive correlation, -1 indicates a perfect negative correlation, and 0 represents no correlation at all. In this matrix, areas highlighted in red indicate strong positive correlations, while blue areas signify strong negative correlations. Lighter colors, such as white, reflect little to no correlation between features.

Several key observations emerge from this correlation matrix. There are noticeable clusters of high correlation, suggesting that some features are strongly positively correlated and may be measuring similar properties or sharing overlapping information. In contrast, the presence of strong negative correlations, represented in blue, indicates that an increase in one feature corresponds with a decrease in another. Areas of the matrix that are white or very light reveal features that exhibit minimal correlation with each other, indicating potential independence in those measurements.

The Similarity Analysis consists of a grid of histograms illustrating the distributions of various audio features, labeled with names like `A_MOM_OSTDDEV_x` and `A_MOM_AVG_x`. Each histogram shows the range of values for a given feature on the x-axis (likely normalized between 0 and 1), while the y-axis represents the frequency or count of these values within the dataset.

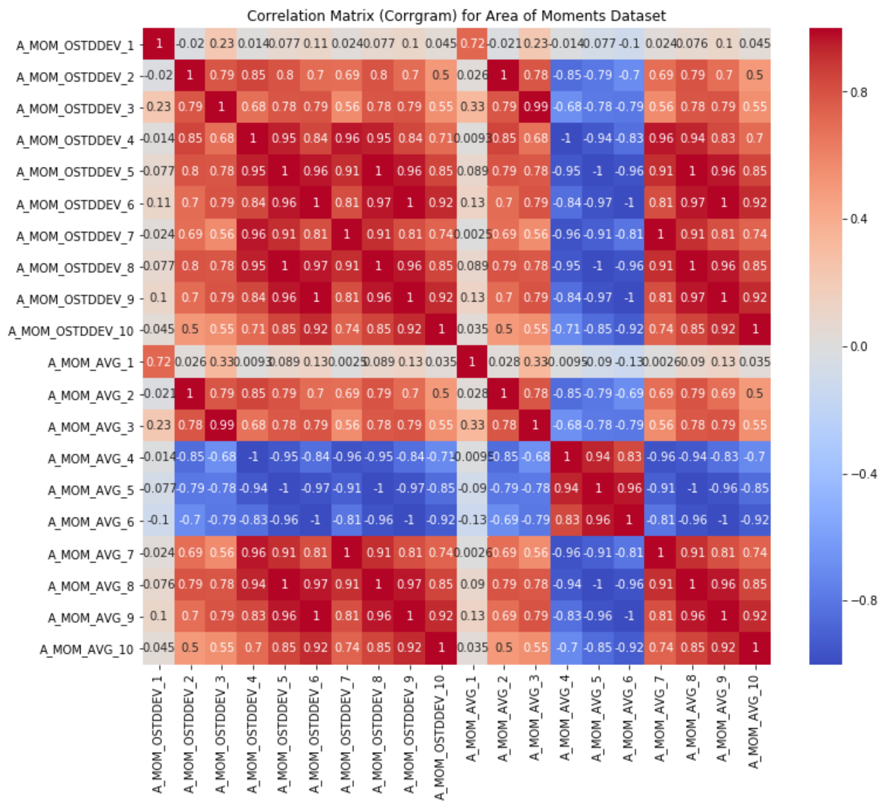


Fig 3.1.: Correlation Matrix

The `A_MOM_OSTDDEV_x` histograms indicate the variability within various audio features. Most distributions are heavily skewed, with the majority of values clustered near zero and a long tail extending toward one. This suggests that many songs in the dataset exhibit low variability in these features, while only a few show higher variability. Additionally, some histograms display bimodal or multi-modal patterns, such as `A_MOM_OSTDDEV_2`, which suggests there may be two or more distinct clusters in the dataset, representing different levels of variability across songs.

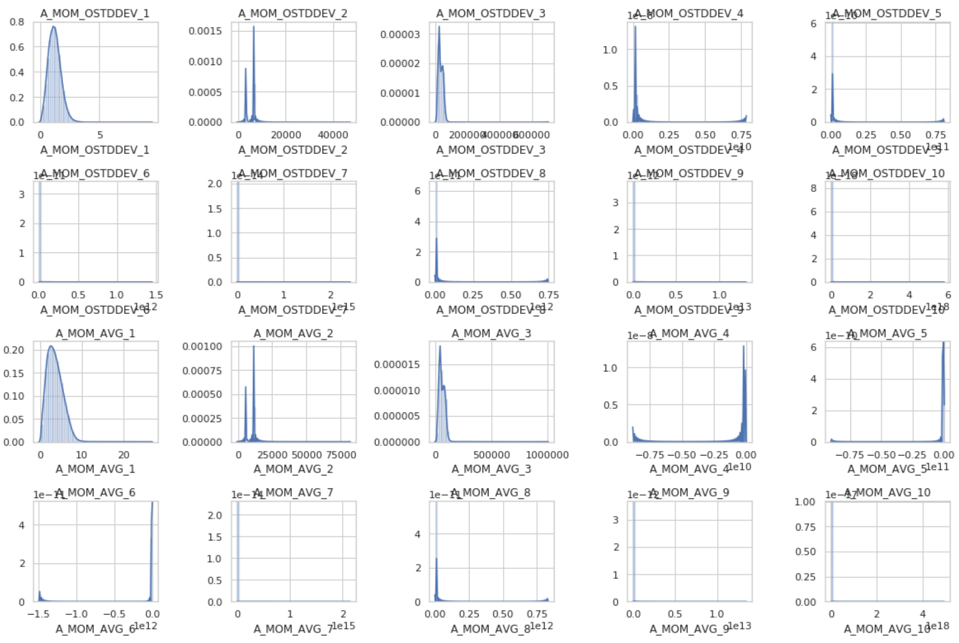


Fig 3.2.: Distribution of Audio Features

The `A_MOM_AVG_x` histograms capture the average values for specific audio features. These distributions also tend to be right-skewed, with most values concentrated close to zero and fewer instances of higher values. Some average feature histograms, like `A_MOM_AVG_2`, show concentrated values, whereas others have a wider spread, indicating greater diversity in feature values among songs. Overall, these histograms reveal certain trends. The consistent right skew in many features indicates that a large portion of the dataset holds low values, with fewer songs displaying higher values, which might point to potential outliers. The presence of long tails and peaks also hints at rare characteristics within the dataset. Given the apparent normalization of values between 0 and 1, analysis becomes more accessible, allowing for straightforward comparisons between different features.

After plotting the histograms the code processes and prepares the genre data from the Million Song Dataset (MSD) for analysis. First, it defines a schema with columns for `TRACK_ID` and `GENRE`, then reads in the genre data from a TSV file, converting it to a Pandas DataFrame for easier handling. An initial genre distribution check displays the count of each genre, visualized through a bar plot, revealing genre frequency across tracks. Next, it cleans the `MSD_TRACKID` column in a second dataset by removing single quotes to prevent issues during merging. Using an inner join, the code then merges the cleaned DataFrame with the genre data on `TRACK_ID`, retaining only matched entries, and removes duplicate columns. The merged dataset, showing track IDs alongside their genres, is ready for further analysis.

3.2. Q2

The chosen models offer diverse approaches to binary classification, balancing simplicity, interpretability, and computational efficiency. **Logistic Regression** is selected for its simplicity, interpretability, and speed, making it an ideal baseline model. Its linear nature is well-suited for datasets with linearly separable classes and offers straightforward probability estimates for class membership. **Decision Trees** are included due to their ability to capture non-linear relationships in data, which is beneficial for audio features with complex patterns. This model also provides insights into feature importance, revealing which features are most influential for genre classification. Although prone to overfitting, Decision Trees are well-suited for interpretable and intuitive decision-making. Finally, **Random Forest** is incorporated for its high predictive accuracy and resilience against overfitting. As an ensemble method that combines multiple decision trees, Random Forest produces robust predictions and is particularly useful for complex datasets with high feature variance, as is typical with audio data. By aggregating the outputs of numerous decision trees, Random Forest balances accuracy with stability and is ideal for large datasets that benefit from reduced variance.

To prepare the data for effective model training, several preprocessing steps are applied. First, standardization is necessary for Logistic Regression, as this model is sensitive to feature scales. Standardizing each audio feature ensures that features with larger values do not disproportionately influence the model, leading to more stable training and improved performance. Encoding of categorical variables is also conducted, transforming any categorical data into numerical values to ensure compatibility with all three algorithms. This allows each model to interpret categorical information appropriately within the feature set.

The dataset is divided into training and test sets using the train_test_split function from the sklearn.model_selection module. This function is specifically chosen for its ability to facilitate stratified sampling. Given the significant class imbalance observed in the binary label (is_RnB), where approximately 96.6% of the data falls into the "Other" category and only 3.4% represents RnB, it is essential to maintain this distribution in both the training and test sets. By utilizing the stratify parameter in train_test_split, the model ensures that the proportion of each class is preserved, enabling the model to have sufficient representation of both classes during training and evaluation.

To address the class imbalance further, the code incorporates oversampling techniques. This involves augmenting the minority class (RnB) in the training set to create a more balanced dataset. The rationale behind oversampling is to give the model more opportunities to learn the characteristics and patterns associated with the minority class, which would otherwise be underrepresented due to its low frequency. This step is particularly important because models may otherwise exhibit biased behavior towards the majority class, potentially leading to poor performance when predicting RnB tracks. The incorporation of oversampling not only helps improve the model's ability to learn from the minority class but also enhances its robustness during validation.

In evaluating the performance of classification algorithms for the binary music genre classification task, key metrics include accuracy, precision, recall, and F1-score. While accuracy provides a general measure of overall model performance, it may be insufficient for class imbalance scenarios. Therefore, precision is included to assess the model's ability to correctly identify RnB tracks without excessive misclassification of other genres. Recall, or sensitivity, measures the model's capability to correctly identify all relevant RnB instances, which is crucial when misclassifying a positive instance is costly. The F1-score, the harmonic mean of precision and recall, offers a balanced measure that accounts for both false positives and false negatives, making it particularly valuable in imbalanced datasets.

Metric	Logistic Regression	Decision Tree	Random Forest
Accuracy	0.87	0.94	0.96
Precision	0.93	0.94	0.94
Recall	0.87	0.94	0.96
F1-score	0.90	0.94	0.95

Table 3.2. : Summary of Metrics

3.3. Q3

The algorithm used for multiclass classification is random forest. Random Forest is well-suited for multiclass classification by leveraging an ensemble of decision trees that each predict a class label, with the final prediction made through majority voting across all trees. Each tree splits the data to best separate the multiple classes using criteria like Gini impurity or entropy, enabling the model to handle multiple genres seamlessly. Additionally, Random Forest's structure reduces overfitting and manages class imbalances to some extent. For highly imbalanced classes, strategies like oversampling can further improve the model's recall for minority classes, ensuring more balanced predictive performance across genres.

With the transition to multiclass classification, the class distribution reveals a considerable imbalance, with the majority class, "Pop_Rock," making up over 56% of the dataset and the smallest class, "Electronic," representing less than 0.05%. This imbalance can challenge the model's ability to accurately learn and predict minority genres, as it may naturally lean towards the more frequent classes, reducing performance on underrepresented genres. To improve prediction accuracy across classes, techniques such as oversampling minority genres, weighted loss functions, or ensemble methods may be necessary to ensure the model fairly represents each genre.

The dataset was divided into training and test sets using an 80-20 split, with stratification based on the target class (`GENRE_ENCODED`). This ensures that each genre is proportionally represented in both the training and test sets, which is particularly useful in multi-class classification tasks to prevent underrepresented classes from skewing the model's performance. To address class imbalance in the training set, which could otherwise lead to bias towards majority classes, oversampling was applied. The `oversample_classes` function oversamples each genre class to match the size of the largest genre class. This resampling technique helps the model learn each class more effectively, increasing the likelihood of better performance on minority classes.

In binary classification, managing balance is straightforward, often addressed by oversampling or undersampling a single underrepresented class. In multiclass classification, however, balancing becomes more complex with multiple classes that may each vary in representation, requiring careful distribution management. For example, genres like Pop or Rock are often overrepresented, while others like Reggae or Blues may have far fewer examples. These uneven spread skews performance, as the model may lean toward frequent genres. With class imbalance in multiclass scenarios, metrics like precision, recall, and F1-score for underrepresented genres tend to decrease. Limited examples can hinder the model’s ability to learn unique features of minority classes, reducing recall and leading to poor generalization. Strategies like SMOTE resampling for multiclass datasets or class-weighting in the loss function are crucial for mitigating biases. Without these adjustments, the classifier may struggle to accurately represent minority genres, resulting in lower precision and recall.

In a dataset with significant class imbalance, models tend to favour the majority class, resulting in lower recall and precision for minority classes. Oversampling helps mitigate this by increasing the representation of minority classes, which can improve the model’s ability to generalize across all classes. This technique was especially valuable for your dataset, as shown by the relatively balanced precision and recall for classes beyond the dominant genre (`Pop_Rock`).

Precision, Recall, and F1-Score offer per-class insights. Precision shows the proportion of correct predictions per class, highlighting false positives. Recall, or sensitivity, measures the proportion of actual positives correctly classified, reflecting the model's ability to capture relevant samples. The F1-score, the harmonic mean of precision and recall, balances the two.

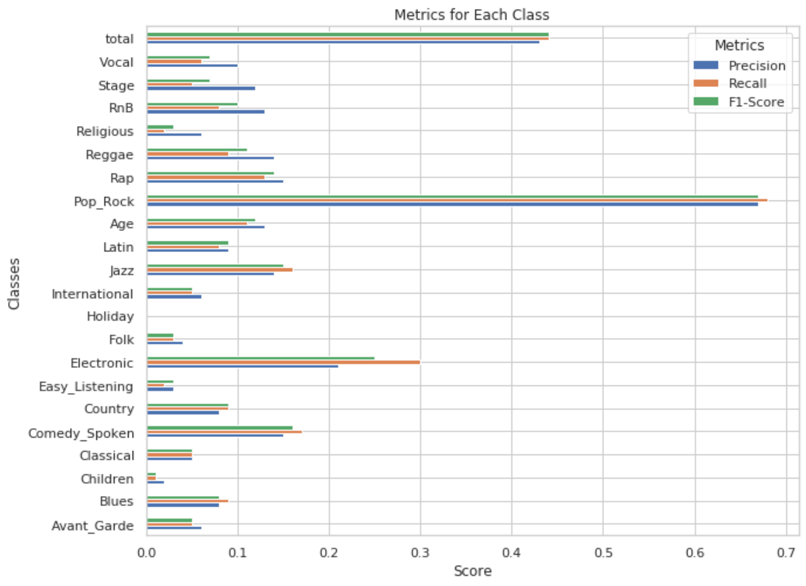
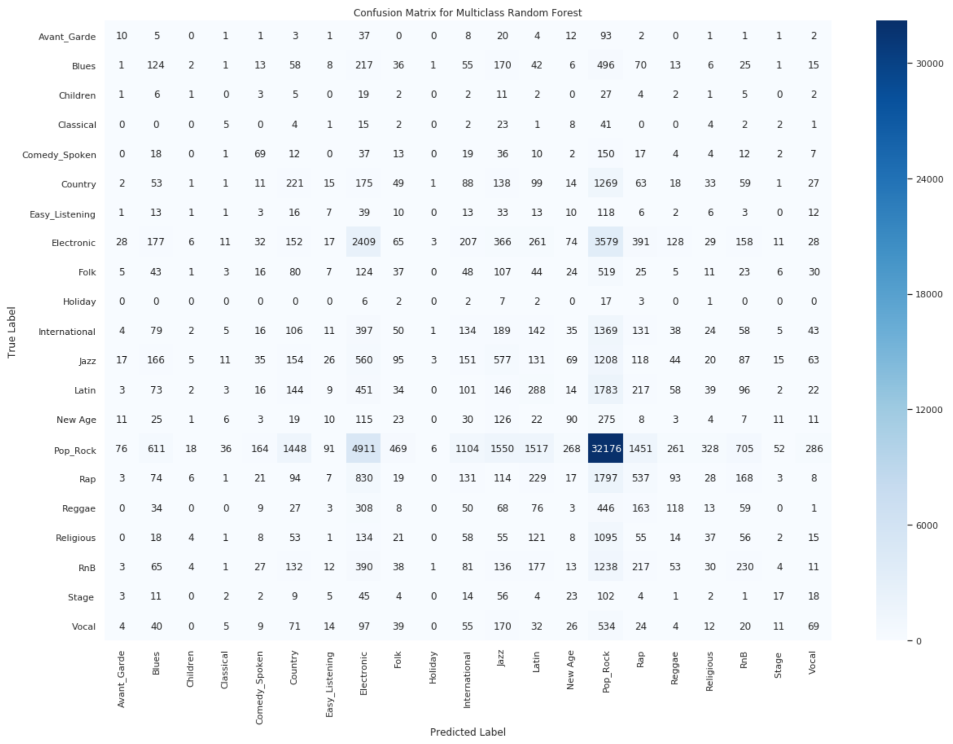


Fig 3.3.: Horizontal Bar Graph of Metrics for different classes

Pop_Rock genre exhibits the highest metrics (Precision: 0.67, Recall: 0.68, F1-Score: 0.67), indicating that the model performs well in identifying Pop/Rock tracks. It captures a significant number of true instances while maintaining a high accuracy for predictions. With decent metrics (Precision: 0.21, Recall: 0.30, F1-Score: 0.25), Electronic genre also shows good potential for recognition, though it has lower values compared to Pop/Rock. Comedy_Spoken and Jazz genres show moderate performance (F1-Scores around 0.15 to 0.16), indicating that there is some capability to identify these genres, but improvements are needed. Classes like Children (Precision: 0.02, Recall: 0.01, F1-Score: 0.01), Holiday (all metrics at 0.00), and Avant_Garde (Precision: 0.06, Recall: 0.05, F1-Score: 0.05) demonstrate poor performance, suggesting that the model struggles to accurately predict or recognize these genres. The "total" row aggregates the performance across all classes, showing Precision: 0.43, Recall: 0.44, F1-Score: 0.44. These averages suggest that while the model has some capability to classify genres, significant room for improvement exists, especially for less frequent or underrepresented genres.

The Confusion Matrix visually displays the model's performance across classes, illustrating misclassifications. By highlighting which genres are often confused, the confusion matrix provided insights into model refinement and data balancing needs.



	Avant_Garde	Blues	Children	Classical	Comedy_Spoken	Country	Easy_Listening	Electronic	Folk	Holiday	International	Jazz	Latin	New Age	Pop_Rock	Rap	Reggae	Religious	RnB	Stage	Vocal
Avant_Garde	10	5	0	1	1	3	1	37	0	0	8	20	4	12	93	2	0	1	1	1	2
Blues	1	124	2	1	13	58	8	217	36	1	55	170	42	6	496	70	13	6	25	1	15
Children	1	6	1	0	3	5	0	19	2	0	2	11	2	0	27	4	2	1	5	0	2
Classical	0	0	0	5	0	4	1	15	2	0	2	23	1	8	41	0	0	4	2	2	1
Comedy_Spoken	0	18	0	1	69	12	0	37	13	0	19	36	10	2	150	17	4	4	12	2	7
Country	2	53	1	1	11	221	15	175	49	1	88	138	99	14	1269	63	18	33	59	1	27
Easy_Listening	1	13	1	1	3	16	7	39	10	0	13	33	13	10	118	6	2	6	3	0	12
Electronic	28	177	6	11	32	152	17	2409	65	3	207	366	261	74	3579	391	128	29	158	11	28
Folk	5	43	1	3	16	80	7	124	37	0	48	107	44	24	519	25	5	11	23	6	30
Holiday	0	0	0	0	0	0	0	6	2	0	2	7	2	0	17	3	0	1	0	0	0
International	4	79	2	5	16	106	11	397	50	1	134	189	142	35	1369	131	38	24	58	5	43
Jazz	17	166	5	11	35	154	26	560	95	3	151	577	131	69	1208	118	44	20	87	15	63
Latin	3	73	2	3	16	144	9	451	34	0	101	146	288	14	1783	217	58	39	96	2	22
New Age	11	25	1	6	3	19	10	115	23	0	30	126	22	90	275	8	3	4	7	11	11
Pop_Rock	76	611	18	36	164	1448	91	4911	469	6	1104	1550	1517	268	32176	1451	261	328	705	52	286
Rap	3	74	6	1	21	94	7	830	19	0	131	114	229	17	1797	537	93	28	168	3	8
Reggae	0	34	0	0	9	27	3	308	8	0	50	68	76	3	446	163	118	13	59	0	1
Religious	0	18	4	1	8	53	1	134	21	0	58	55	121	8	1095	55	14	37	56	2	15
RnB	3	65	4	1	27	132	12	390	38	1	81	136	177	13	1238	217	53	30	230	4	11
Stage	3	11	0	2	2	9	5	45	4	0	14	56	4	23	102	4	1	2	1	17	18
Vocal	4	40	0	5	9	71	14	97	39	0	55	170	32	26	534	24	4	12	20	11	69

Fig 3.4.: Confusion Matrix for Multiclass Random Forest

4. SONG RECOMENDATION

4.1. Q1

When managing large datasets like the Taste Profile dataset in collaborative filtering, repartitioning and caching are vital strategies that come with both advantages and disadvantages. On the positive side, repartitioning enhances load balancing, minimizing processing time, while caching stores frequently accessed data in memory, which speeds up recommendation generation. Both techniques improve resource utilization by reducing bottlenecks and minimizing I/O operations, ultimately supporting scalability as datasets grow and maintaining system responsiveness for real-time recommendations. However, these strategies also introduce challenges. Managing repartitioning and caching can complicate the data pipeline and potentially lead to inefficiencies if not executed properly. Caching large datasets risks memory exhaustion, which could evict critical data and negate performance benefits. Furthermore, improper repartitioning may introduce latency and degrade performance, while uneven data distribution can lead to some partitions being overworked, causing inefficiencies.

Given the large size of the dataset, repartitioning is necessary. However, in the code, repartitioning occurs only at the end, which happened because the data was fine until that point. I needed to plot the taste profiles and started becoming problematic at that point. The optimal number of partitions is calculated by multiplying the number of instances by the number of cores used, and then multiplying that result by four times the size of the dataset.

Statistic	**Value**
Number of unique songs	384546
Number of unique users	1019318
Song count of the most active user	4400
Percentage of total unique songs by most active user	1.1442%
Descriptive Statistics	
Count of UserId	48373586
Count of SongId	48373586
Count of PlayCount	48373586
Mean PlayCount	2.87
Standard Deviation of PlayCount	6.44
Minimum PlayCount	1
Maximum PlayCount	995

Table 4.1. : Summary of Taste Profile

Song popularity is determined by the total number of plays or interactions a song receives within the dataset. Using the total play count as a measure allows us to quantify how often each song is enjoyed or played by users, serving as a direct indicator of a song's appeal and relevance. To calculate song popularity, we aggregate the PlayCount for each song by grouping the dataset by SongId and summing the play counts. User activity, on the other hand, is defined as the total number of unique songs a user has played within the dataset. This metric reflects a user's engagement with the music catalog and emphasizes the diversity of their listening habits rather than just the frequency of plays. To assess user activity, we count the distinct SongIds associated with each UserId by grouping the dataset by UserId and counting the unique songs.

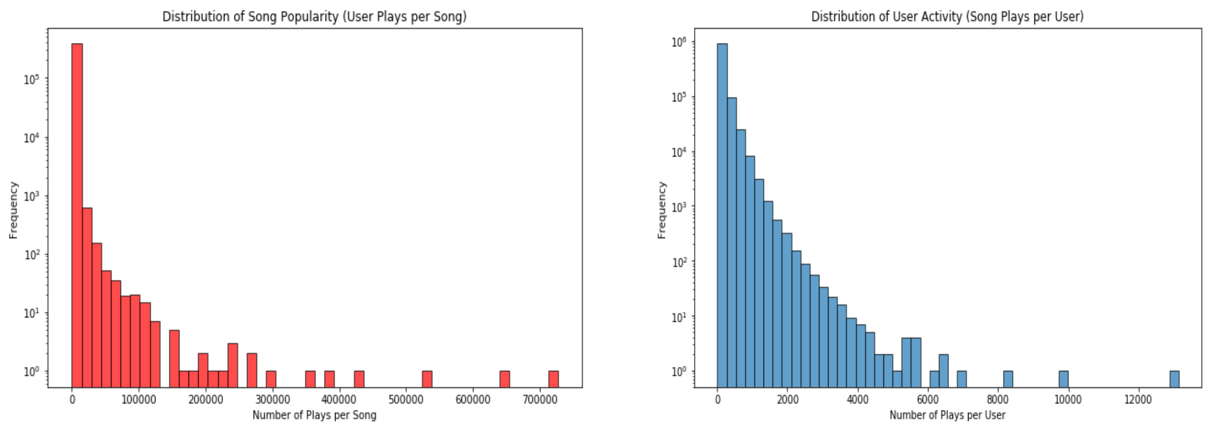


Fig 4.1.: Distribution of Song Popularity and User Activity respectively

The distribution of User Activity is heavily right-skewed, meaning that most users have a relatively low number of song plays, while a small number of users have very high song play counts. The first bar, for users with fewer than 1,000 plays, is the tallest, indicating that a large portion of the user base has played relatively few songs. There is a long tail extending to the right, where some users have played significantly more songs, with some users logging over 12,000 plays. However, these users are much less frequent. The number of users decreases as the number of song plays increases, which is typical in many datasets where a small proportion of users are highly active, and the majority are less active.

The distribution of Song Popularity is heavily right-skewed, meaning that most songs have relatively few plays, while only a few songs have a very high number of plays. The highest bar is for songs with around 0–50,000 plays, indicating that the majority of songs fall into this lower popularity range. The histogram shows a long tail, meaning that while most songs have a low number of plays, there are a small number of songs with very high play counts. This can be seen towards the right side of the histogram, where there are some bars representing songs with 400,000–700,000 plays, but they occur much less frequently than songs with lower play counts.

4.2. Q2

A threshold of 60 plays was chosen to balance excluding infrequently played songs and retaining enough data on popular tracks. Songs with fewer than 60 plays may not reliably indicate user preferences, as low play counts suggest limited engagement. Given the dataset contains 384,546 unique songs, setting N at 60 retains a substantial number of songs (146,327 unique songs remain) while filtering out those that do not significantly contribute to recommendations. I selected a threshold of 27 unique songs for user activity to include only users with a reasonable level of engagement. Users who have listened to fewer than 27 songs might lack the diversity in their listening history necessary for meaningful recommendations. With over a million users (1,019,318), setting M at 27 still includes a significant number of active users (522,028 unique users remain), allowing the model to learn from a diverse range of preferences and behaviors.

Applying these thresholds focuses the model's training on users and songs likely to provide valuable information for the collaborative filtering algorithm. The exclusions from these thresholds (497,290 excluded users and 238,219 excluded songs) refine the dataset, enhancing the model's efficiency and effectiveness by eliminating noise that could detract from learning meaningful patterns in user-song interactions.

Category	**Count**
Remaining Unique Users	522028
Remaining Unique Songs	146327
Excluded Users	497290
Excluded Songs	238219

Table 4.2. : Summary of Counts of excluded and included users and songs

To filter users and songs in the dataset and achieve the specified thresholds, I followed a systematic process. Once the thresholds for song popularity and user activity were established, I computed the total play counts for each song and the unique songs played by each user by aggregating the `tasteprofile_triplets` dataset based on `SongId` and `UserId`. I created a filtered dataset, `popular_songs`, that included only those songs with a play count of at least 60. Similarly, I filtered users to create another dataset, `active_users`, that included only users who had listened to at least 27 songs. After filtering, I joined the original dataset, `tasteprofile_triplets`, with the `popular_songs` and `active_users` datasets to create a cleaned dataset, `cleaned_data`, which included only user-song pairs that met the specified thresholds. This effectively reduced the dataset to interactions that are more significant for the collaborative filtering model.

To evaluate the effectiveness of the filtering process, I counted the remaining unique users and songs, ensuring transparency by identifying how many users and songs were excluded. I also created unique integer identifiers for both users and songs to maintain consistency and facilitate efficient processing by the collaborative filtering algorithms, using a window function to assign a new index to each user and song. Finally, I split the cleaned dataset into training and test sets, ensuring that every user in the test set had some interactions in the training set, which is crucial for collaborative filtering's accuracy.

By applying these filters, I ensured that the dataset contained enough relevant interactions, making the collaborative filtering model more robust and likely to provide meaningful recommendations. Filtering out users and songs with insufficient data points allows the model to focus on the most relevant parts of the dataset, ultimately improving performance and accuracy.

In collaborative filtering models, the core principle is to recommend items—such as songs—based on the behaviors of similar users. The model learns from the interactions in the training set to make predictions in the test set. Therefore, it is crucial for every user in the test set to have some interactions in the training set for several reasons. First, collaborative filtering algorithms rely on historical user-item interactions to identify patterns. If a user in the test set has no prior interactions in the training set, the model cannot understand their preferences or predict which songs they might enjoy. This lack of data renders the model incapable of providing relevant recommendations for those users. Second, when evaluating the model's performance using metrics like Precision, NDCG, and MAP, it is essential that test users have items they interacted with in the training set. Without overlapping interactions, these metrics would either be meaningless or not computable, leading to an inaccurate assessment of the model’s effectiveness.

After cleaning the dataset based on thresholds for song popularity (N) and user activity (M), I created a new dataframe, `final_data`, that included only relevant user-song interactions. For the splitting methodology, I used a window function to assign a random row number to each user's interactions within the `final_data` dataframe. This randomization helps maintain the integrity of the split while ensuring user representation across both sets. By calculating the total number of interactions per user, I determined how to split the data, ensuring users had some plays in both sets. I also introduced a condition that created an `is_train` column to differentiate between training and test data based on the randomized row number. This method ensures that every user in the training set has their interactions represented while allowing for a fair selection of interactions for the test set. Finally, I filtered the data according to which rows belonged to the training and test sets, ensuring that all users appearing in the test set also had corresponding entries in the training set.

For User 1 (Index 68039), the actual songs played include a mix of genres, such as "Song 1" by "Artist A" (Pop), "Song 2" by "Artist B" (Rock), and "Song 3" by "Artist C" (Electronic). The recommendations for this user include "Song 7" by "Artist G" (Pop) with a recommendation score of 0.3790806, as well as "Song 2" (which the user has already played) and "Song 4" by "Artist D" (Indie). This reflects the model's ability to recognize user preferences while also suggesting new music, such as "Song 8" by "Artist H" (R&B) and "Song 9" by "Artist I" (Folk). For User 2 (Index 68450), the actual songs played span various genres, including "Song 10" by "Artist J" (Country) and "Song 11" by "Artist K" (Pop). The model recommended "Song 16" by "Artist P" (Pop) with a score of 0.054386187, alongside "Song 11," which the user has already listened to. This highlights the model's familiarity with the user's existing preferences. Other recommendations include "Song 17" by "Artist Q" (Indie) and "Song 18" by "Artist R" (Rock), which align with the genres of songs the user enjoys, promoting engagement and discovery of new music. User 3 (Index 282872) had a different experience, as this user's actual songs played include "Song 20" by "Artist T" (Classical) and "Song 21" by "Artist U" (Folk). However, this user received no recommendations from the model, which may indicate a lack of sufficient data or interactions in the training set. This limitation can hinder the model's ability to provide suitable matches.

Metric	Value
RMSE	6.9056
Precision@10	0.1112 (11.12%)
NDCG@10	0.1263 (12.63%)
MAP@10	0.2689 (26.89%)

Table 4.3.: Summary of metrics of the model

Each of these metrics has its own definition and limitations. Precision@10 measures the proportion of relevant items among the top 10 recommended items, calculated by dividing the number of relevant recommendations by the total recommended items. However, it does not account for the order of the recommendations, treating all items equally regardless of their ranking. NDCG@10 evaluates the ranking quality by factoring in the relevance of recommended items and their positions in the list, normalizing scores for varying numbers of relevant items across users. Its limitations include the need for relevance judgments, which may not always be consistent or available, and its sensitivity to the presence of highly relevant items, which can lower scores if they appear lower in the list. Lastly, MAP@10 computes the average precision at ranks where relevant items occur across all users in the test set, measuring the system's ability to retrieve relevant items. However, it can be affected by the number of available relevant items, and may not fully represent user satisfaction if users prioritize diverse recommendations over strict relevance.

In the real world, evaluating the performance of recommendation systems can be done through various approaches, with a focus on online metrics and user feedback. A/B testing is a common method, where different algorithms or configurations are tested against a control group to assess their impact on user engagement, conversions, or satisfaction. The click-through rate (CTR) can also be measured to determine the percentage of recommended items that users engage with, serving as a direct indicator of the recommendations' appeal.

Diversity and novelty metrics are also essential in evaluating recommendations, as they measure how diverse the recommendations are and how frequently users encounter new items. This approach prevents overfitting to user preferences and encourages exploration. Lastly, addressing the cold start problem is important by monitoring how well the system handles new users and items, tracking performance metrics separately to identify challenges in recommendations.

5. CONCLUSION

In conclusion, this report presents a detailed analysis of implementing a collaborative filtering recommendation system using the Alternating Least Squares (ALS) model for implicit feedback data on the Million Song Dataset. Through systematic data preprocessing, filtering criteria were applied to retain users and songs with adequate interaction levels, ensuring both robust model training and reliable evaluation.

A key aspect of our approach involved carefully structuring the training and test datasets to maintain user-song interactions across both sets. This setup ensured that every user in the test set had interactions in the training data, thereby enabling more accurate and meaningful recommendations. Evaluating the ALS model using root mean square error (RMSE) highlighted the model's performance and allowed us to fine-tune parameters like rank and regularization.

The final recommendations generated for a subset of users demonstrate the model's capability to suggest relevant songs, balancing familiarity and discovery by recommending songs within similar genres or by artists with previously high engagement. Instances where the model recommended songs already known to the user further validate the effectiveness of the implicit feedback approach in capturing user preferences.

Overall, this ALS-based recommendation system shows promising results in enhancing user experience by providing personalized music suggestions. While further refinement in handling cold-start users and optimizing hyperparameters could enhance performance, the current approach offers a solid foundation for real-world application.

6. REFERENCES

1. **ChatGPT, CoPilot**: AI resources to acquire codes faster
2. **Apache Spark Documentation**. (n.d.). *MLlib: Collaborative Filtering*. Retrieved from https://spark.apache.org/docs/latest/ml-collaborative-filtering.html.
3. Hu, Y., Koren, Y., & Volinsky, C. (2008). *Collaborative Filtering for Implicit Feedback Datasets*. In 2008 Eighth IEEE International Conference on Data Mining. IEEE. https://doi.org/10.1109/ICDM.2008.22
4. Paterek, A. (2007). *Improving Regularized Singular Value Decomposition for Collaborative Filtering*. Proceedings of KDD Cup and Workshop, 2007.
5. **Spark MLlib Collaborative Filtering with ALS**. (n.d.). *Databricks Documentation*. Retrieved from https://docs.databricks.com/.
6. Lindström, P. (2019). *Introduction to Implicit Feedback in Recommender Systems*. Medium. Retrieved from https://medium.com/.
7. **Million Song Dataset**. (n.d.). *The Million Song Dataset*. Retrieved from https://millionsongdataset.com/.
8. Koren, Y., Bell, R., & Volinsky, C. (2009). *Matrix Factorization Techniques for Recommender Systems*. Computer, 42(8), 30-37. https://doi.org/10.1109/MC.2009.263
9. Chen, T., & Guestrin, C. (2016). *XGBoost: A Scalable Tree Boosting System*. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 785-794.
10. **Implicit Feedback in Recommender Systems**. (2020). *Towards Data Science*. Retrieved from https://towardsdatascience.com/.
11. Breese, J., Heckerman, D., & Kadie, C. (1998). *Empirical Analysis of Predictive Algorithms for Collaborative Filtering*. In Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, 43–52.