The Million Song Dataset (MSD)

Blake List (31/05/2019)

For this task, we will be exploring a collection of datasets known as the Million Song Dataset (MSD), which started as a collaborative project between The Echo Nest and LabROSA.

The main dataset contains the song ID, the track ID, the artist ID, and 51 other fields, such as the year, title, artist tags, and various audio properties such as loudness, beat, tempo, and time signature. Note that track ID and song ID are not the same concepts - the track ID corresponds to a particular recording of a song, and there may be multiple (almost identical) tracks for the same song. Tracks are the fundamental identifier and are matched to songs. Songs are then matched to artists as well.

The Million Song Dataset also contains other datasets contributed by organisations and the community,

- SecondHandSongs (cover songs)
- musiXmatch dataset (song lyrics)
- Last.fm dataset (song-level tags and similarity)
- Taste Profile subset (user-song plays)
- thisismyjam-to-MSD mapping (user-song plays, imperfectly joined)
- tagtraum genre annotations (genre labels)
- Top MAGD dataset (more genre labels)

During this task, we will be focusing on Taste Profile, Audio Features and MSD AllMusic Genre Dataset (MAGD).

The Taste Profile dataset contains real user-song play counts from undisclosed organisations. All songs have been matched to identifiers in the main million song dataset and can be joined with this dataset to retrieve additional song attributes. The dataset has an issue with the matching between the Taste Profile tracks and the million song dataset tracks. Some tracks were matched to the wrong songs, as the user data needed to be matched to song metadata, not track metadata.

The Audio Features dataset contains a multitude of features extracted from audio samples for 994,960 songs to allow comparison between the songs and prediction of song attributes. Some example features include ‘Statistical Spectrum Descriptors’, ‘Timbral features’, and ‘jMir Area of Moments’.

The MSD Allmusic Genre Dataset (MAGD) contains all genre labels collected from Allmusic.com. This includes generic genres like Religious or Christmas, non-musical content like Comedy/Spoken and significantly small genres. The dataset has a size of 422714 labels.

Processing

Question 1

We have four datasets: audio (12.3 GB), genre (30.1 MB), main (174.4 MB), and tasteprofile (490.4 MB) replicated over HDFS four times. We will explore the structure of each one in more detail.

```
12.3 G   49.1 G   /data/msd/audio
30.1 M   120.5 M  /data/msd/genre
174.4 M  697.6 M  /data/msd/main
490.4 M  1.9 G    /data/msd/tasteprofile
```

Figure 1: Overview of the MSD dataset on HDFS.

The audio dataset contains three directories: attributes (103.0 MB), features (12.2 GB), and statistics (40.3 MB). The attributes contain 3 CSV files pertaining to each of the audio features (Rhythm Patterns, Marsyas, and jMir). The features folder contains 13 subfolders each containing some number of compressed CSV part files for each audio feature. The attributes CSV files range from just under 1 KB to just under 35 KB, while the features CSV files range from 35 MB to nearly 4 GB in size. The statistics folder contains just one gzipped CSV file for the sample properties compressed to 40 MB.

```
103.0 K  412.2 K  /data/msd/audio/attributes
12.2 G   48.9 G   /data/msd/audio/features
40.3 M   161.1 M  /data/msd/audio/statistics
```

Figure 2: Overview of the audio features data.

Looking at each file more closely, each of the audio feature CSV's in attribute contains various numerical string ID data pertaining to each feature. The level of parallelism we can hope to achieve with loading in the data is dependent on the number of replications. In this case, we have four replications of the data on HDFS so we can parallelise loading, however, as the statistics and features data is gzipped, we cannot parallelise applying transformations as we will only ever have one core working on one partition. Once the data is read in, we can parallelise actions on the data based on the number of partitions (from the number of compressed files) and the number of workers. As the attribute data is not compressed, we do not have this limitation.

```
MFCC_Overall_Standard_Deviation_1,real
MFCC_Overall_Standard_Deviation_2,real
MFCC_Overall_Standard_Deviation_3,real
MFCC_Overall_Standard_Deviation_4,real
MFCC_Overall_Standard_Deviation_5,real
MFCC_Overall_Standard_Deviation_6,real
MFCC_Overall_Standard_Deviation_7,real
MFCC_Overall_Standard_Deviation_8,real
MFCC_Overall_Standard_Deviation_9,real
MFCC_Overall_Standard_Deviation_10,real
MFCC_Overall_Standard_Deviation_11,real
MFCC_Overall_Standard_Deviation_12,real
MFCC_Overall_Standard_Deviation_13,real
MFCC_Overall_Average_1,real
MFCC_Overall_Average_2,real
MFCC_Overall_Average_3,real
MFCC_Overall_Average_4,real
MFCC_Overall_Average_5,real
MFCC_Overall_Average_6,real
MFCC_Overall_Average_7,real
MFCC_Overall_Average_8,real
MFCC_Overall_Average_9,real
MFCC_Overall_Average_10,real
MFCC_Overall_Average_11,real
MFCC_Overall_Average_12,real
MFCC_Overall_Average_13,real
MSD_TRACKID,string
```

Figure 3: Column names and data types for an attributes CSV.

The genre dataset contains three tab-separated (TSV) files pertaining to the MSD Allmusic genre and style assignments. The three files are between 8 and 11 MB in size. Each TSV contains a song ID and a genre or style category such as 'Pop_Rock' or 'Metal_alternative', respectively. As none of the files in the genre folder are compressed, we can expect to parallelise the loading of the data by the number of replications on HDFS, and the transforming of the data dependent on the number of workers.

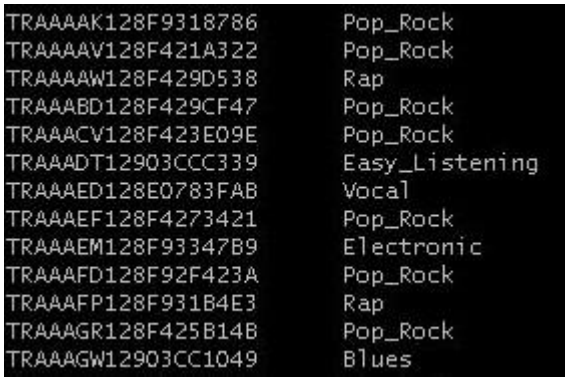


Figure 4: Head of the MSD genres data.

The main dataset contains the song ID, the track ID, the artist ID, and 51 other fields, such as the year, title, artist tags, and various audio properties such as loudness, beat, tempo, and time signature. The main folder contains a summary subfolder which has two gzipped CSV files - analysis (55.9 MB) and metadata (118.5 MB). As these files are both compressed, we will be able to parallelise the loading of the data up to the number of repetitions on HDFS, but we will only have one worker applying transformations until they are unzipped.

Figure 5: Main dataset compressed files.

The tasteprofile data contains two folders - mismatches (2.0 MB), and triplets.tsv (488.4 MB). The mismatches folder contains two txt files, one for the identifiers and names of the mismatched songs, and the other for the manually added matches. The triplets.tsv folder contains eight gzipped part files for the implicit feedback user data. We would be able to parallelise the loading up to the number of replication, as previously mentioned, and the transformations would only be performed by one worker until the data is unzipped.

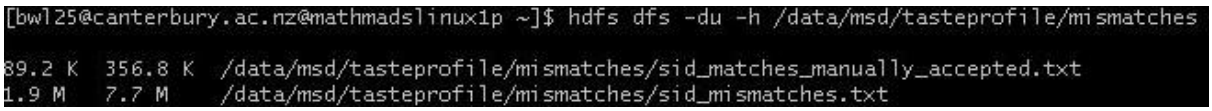


Figure 6: Mismatched song files in the Taste Profile dataset.

The repartition algorithm does a full shuffle and creates new partitions with data that is distributed evenly. As Spark can only run one concurrent task for every partition of an RDD, up to the number of cores used in the cluster, we would want to repartition the data to have about 2-4 times the number of cores. Repartitioning the data can be useful if the goal is to create a larger number of partitions for processing. As a full data shuffle is an expensive process, we must decide whether it is worth doing if there are a large number of rows in a particular dataset.

Within the audio dataset, the attributes files contain small numbers of rows ranging from 11 to 1177 depending on the type of attribute. The audio features and statistics files have around 994188 and 992866 rows, respectively, just less than the total number of unique songs in the dataset. The three

genre datasets, MAGD genre, MASD style, and topMAGD genre each have 422714, 273936, and 406427 rows, respectively. Lastly, the Taste Profile triplets dataset has 48373586 lines.

Question 2

The first part of pre-processing the data involved loading the Taste Profile dataset and filtering out the songs that were mismatched. The schema of the triplets dataset was determined from the description of the data and confirmed when the dataset was initially loaded in. The two mismatch text files pertaining to song ID mismatches and manually accepted mismatches were both loaded in as fixed-width text format and the song and track IDs were extracted. The manually accepted song IDs were then anti-joined from the normal mismatches, which were then anti-joined from the original Taste Profile dataset. The result was a dataset containing Song IDs, User IDs, and song play counts making up 45,795,111 matches out of a total 48,373,586 rows.

In order to define the schemas for each of the audio features datasets, we needed to extract the column names and their types from the attributes datasets. These would then have to be mapped to the correct and consistent types such that individual schemas could be created for each audio feature. We needed to convert each row in the attributes data frame into JSON format so that we could create a type structure from the JSON data. First, we initialised a default schema to load the attributes data in and defined a dictionary for the correct data types that we would map to and a list of each of the feature and attribute names. We then iterated through each of the names, loading the data with the default schema, then used JSON to extract the structure for each row. A schema was then created and applied to the features datasets.

Audio Similarity

Question 1

In this section, we would investigate how to use numerical representations of a song's audio waveform to predict its genre. To do this, the audio feature dataset, "jMir Area of Moments" would be used. As the dataset contained 20 columns, 10 for the standard deviation and 10 for the mean of each area of moments, a subset of the data was created to do the descriptive statistics and correlation between variables.

```
In [11]: # Create a subset of the features dataset for describing.
    ...: features_subset = features_jmir_area_moments.select(features_jmir_area_moments.columns[:5])
    ...: features_subset.describe().show()
    ...:
+-------+-----------------------------------------------------+-----------------------------------------------------+
|summary|Area_Method_of_Moments_Overall_Standard_Deviation_1|Area_Method_of_Moments_Overall_Standard_Deviation_2|
+-------+-----------------------------------------------------+-----------------------------------------------------+
|  count|                                               994604|                                               994604|
|   mean|                                1.228921240269988|                                5500.56838524418|
| stddev|                                0.5282405038550929|                                2366.029716955828|
|    min|                                                0.0|                                                0.0|
|    max|                                                9.346|                                               46860.0|
+-------+-----------------------------------------------------+-----------------------------------------------------+

+-----------------------------------------------------+-----------------------------------------------------+-----------------------------------------------------+
|Area_Method_of_Moments_Overall_Standard_Deviation_3|Area_Method_of_Moments_Overall_Standard_Deviation_4|Area_Method_of_Moments_Overall_Standard_Deviation_5|
+-----------------------------------------------------+-----------------------------------------------------+-----------------------------------------------------+
|                                               994604|                                               994604|                                               994604|
|                                33817.8867026857|                                1.2766992610567312E8|                                7.834826067815083E8|
|                                18228.646843590064|                                2.396385081368785E8|                                1.582591707067279E9|
|                                                0.0|                                                0.0|                                                0.0|
|                                               699400.0|                                               7.864E9|                                               8.124E10|
+-----------------------------------------------------+-----------------------------------------------------+-----------------------------------------------------+
```

Figure 7: Descriptive statistics for a subset of the features data.

As we can see from the data, many columns have different scales. For example, Overall_Standard_Deviation_1 has a mean value of 1.23, whereas Overall_Standard_Deviation_2 has a mean value of 5500.59. This large difference in the scale of variables indicates that we may need to normalise the dataset before training any machine learning models on the data.

Furthermore, we can see from the correlation matrix below that some variables are highly correlated. For example, variables “Area_Method_of_Moments_Overall_Standard_Deviation_5” is highly correlated with “Area_Method_of_Moments_Overall_Standard_Deviation_6” with a correlation coefficient of 0.96 (2dp), and “Area_Method_of_Moments_Overall_Average_5” is highly correlated with almost all other features. This implies that principal component analysis may need to be performed to reduce the number of highly correlating features and to reduce the dimensionality of the data before we begin fitting models.

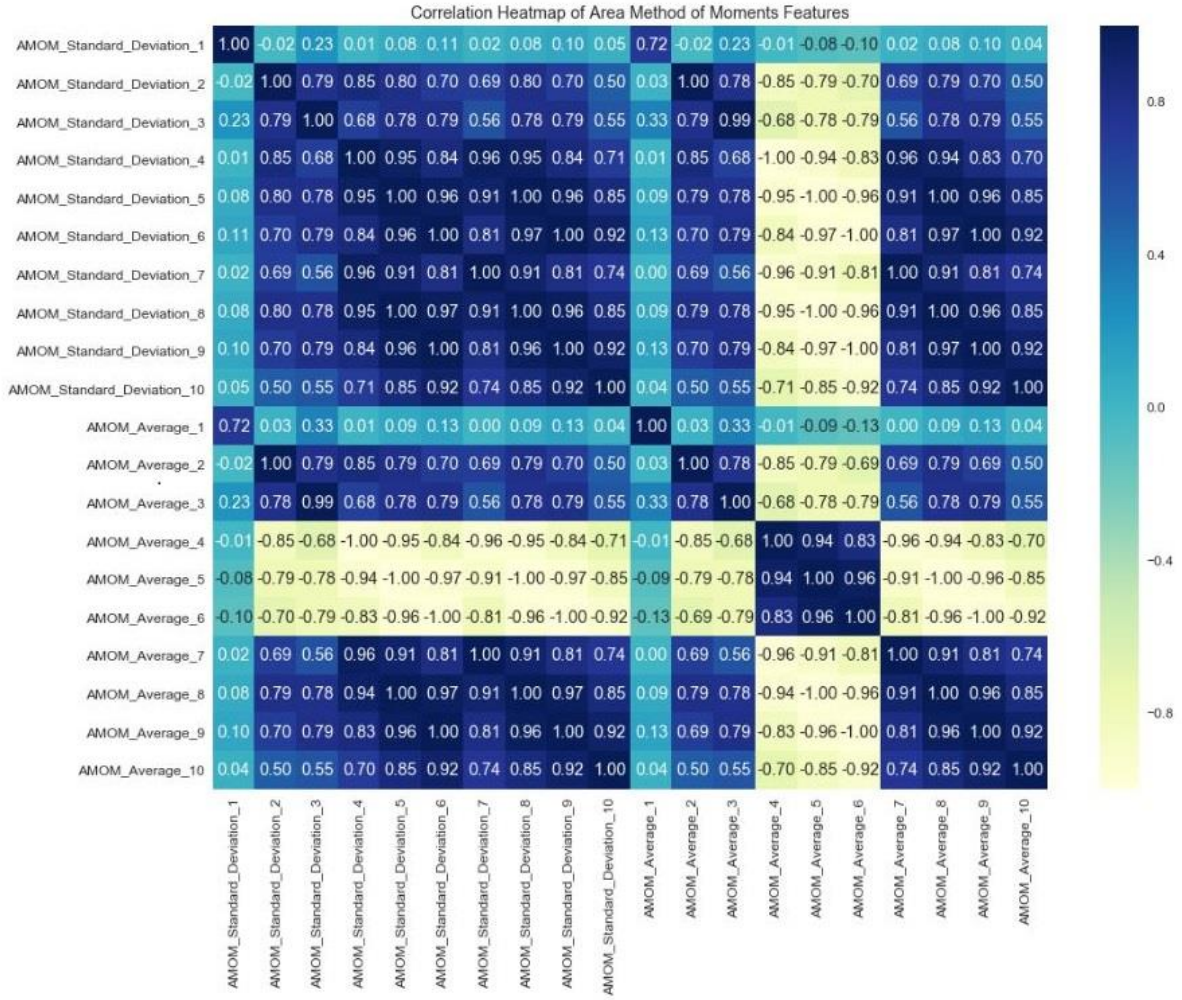


Figure 8: Correlation heatmap of the jMir Area Method of Moments dataset.

Moving on to the MSD All Music Genre Dataset (MAGD), we can visualise the distribution of the genres for the match songs as follows. A log scale plot of the genre distributions has also been included to show the smaller differences between genre counts.

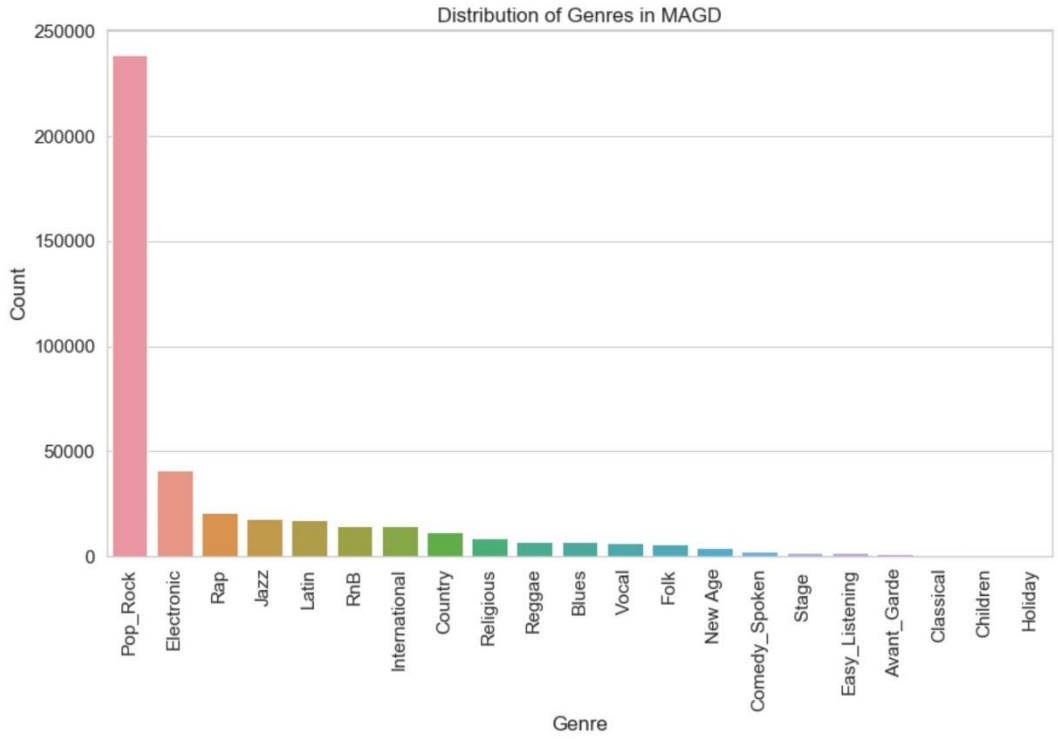


Figure 9: Distribution of the MAGD genres.

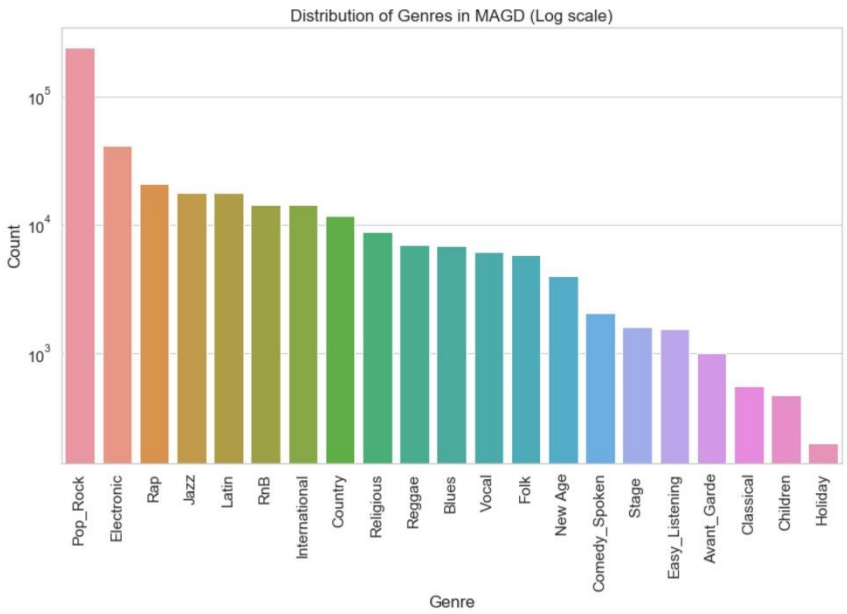


Figure 10a: Distribution of the MAGD genres in log scale.

Genre	Count
Pop_Rock	238786
Electronic	41075
Rap	20939
Jazz	17836
Latin	17590
RnB	14335
International	14242
Country	11772
Religious	8814
Reggae	6946
Blues	6836
Vocal	6195
Folk	5865
New Age	4010
Comedy_Spoken	2067
Stage	1614
Easy_Listening	1545
Avant_Garde	1014
Classical	556
Children	477
Holiday	200

Figure 10b: Genre counts in the MAGD dataset.

As we can see from the distribution of each genre in the MSD All Music Dataset, the genre “Pop Rock” far out-weighs the rest of the genres. We can also see in the log scale distribution plot that there is a large difference between the genres with medium and low counts. For example, the genre “Holiday” only has 200 counts, while others like “Reggae” have almost 7000. This means that we have a heavily imbalanced dataset which we will need to address with resampling methods like oversampling and undersampling when we perform binary and multiclass classification.

Question 2

After merging the genres dataset with the audio features dataset such that every song has a label, we are now reading to perform classification on the data. We will first focus on binary classification. The three classification algorithms that will be implemented for this section are logistic regression, random forest, and support vector machine. Explanations about their explainability, interpretability, predictive accuracy, training speed, hyper-parameter tuning, dimensionality, and issues with scaling follow.

- Logistic Regression – This is a first choice when dealing with binary classification problems and although the name contains 'regression', we can turn it into a classification problem by setting a threshold (e.g. 0.5) for the label class.
    - Logistic regression is a simple algorithm that can be used as a baseline for performance as it is easy to implement and explain. It is also very efficient and highly interpretable.
    - Logistic regression doesn't require input features to be scaled, it doesn't require any tuning, it's easy to regularise, and it outputs well-calibrated predicted probabilities.
    - In terms of its training speed, it is among the faster algorithms which means that it is also efficient to tune the hyper-parameters of with cross-validation.
    - For big data and high dimensional classification, we can use some form of regularisation or apply principal component analysis to the data before training.

- Random Forest – an ensemble of decision trees which are merged together to create a more accurate prediction.
    - Random forest has an advantage over other models with its ease of interpretability and explainability. They can be visualised to see exactly where the splits are being made and what features are contributing to this splits. Its hyper-parameters also produce a good prediction result and depending on their values, the training speed can be relatively efficient.
    - Random forest has many hyper-parameters which can be tuned. These are either used to increase the predictive power of the model or to make the model faster. The number of estimators is the number of trees the algorithm builds before taking the maximum voting. The maximum number of features is the number of features to consider to split a node. The minimum sample leaf is the number of leaves that are required to split an internal node.
    - Random forests also deal well with scaling and high dimensional data but can suffer with computational time depending on the number of estimators. We may want to decrease this to trade prediction accuracy for training efficiency.

- Support Vector Machine – a discriminative classifier formally defined by a separating hyperplane. In other words, given labelled training data the algorithm outputs an optimal hyperplane which categorizes new examples. In two dimensional space, this hyperplane is a line dividing a plane into two parts wherein each class lay in either side.
    - Support vector machines are very accurate and generalise well for a range of applications, especially binary classification, however, they can be more difficult than other algorithms to explain when dealing with high dimensionality. This means that the model produced does not naturally provide any useful intuitive reasons about why a particular point is classified in one class rather than another.

- As dimensionality of the data increases, it becomes more difficult to explain the fit of the support vector machine. But as we can change the degree of the kernel from linear to polynomial and exponential, we get what is called the kernel trick – where the kernel maps the non-linearly separable data into a higher dimensional space to find a hyperplane that can separate the data points.
- The hyper-parameters involved in the SVM model are the kernel (linear, polynomial, etc.), regularisation (how much to avoid misclassifying, at the cost of overfitting and training time), gamma (the distance a point must be at to have some influence on the separation line), and the margin (the separation line between the closest class points).
- The training time of the SVM can be arbitrarily long, depending on the hyper-parameters. The greater the regularisation parameter, the slower the training. A more complicated kernel also means a slower training time. For example, a radial kernel is much more complex than a linear kernel. Lastly, the size of the data and the level of dimensionality plays a large part in the time taken to train the model. We can perhaps reduce the training time by reducing the dimensionality with principal components, reducing the level of regularisation, or training on a subset to determine ideal hyper-parameters.

As previously mentioned, many columns in the data differ by orders of magnitude. Normalising the data will help to restructure the values over the same range to improve stability and help to find some relationship. Furthermore, many columns were shown to be highly correlating. So, principal component analysis will help to reduce the number of dimensions and also collapse similar columns.

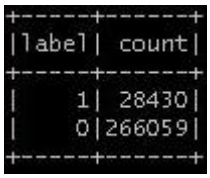
Track_ID	features	Genre	pca_features	label
TRAAABD128F429CF47	[-0.3490032877377...	Pop_Rock	[1.62137127430899...	0
TRAABPK128F424CFDB	[-0.6805307210020...	Pop_Rock	[1.06356249772653...	0
TRAACER128F4290F96	[2.64098457302018...	Pop_Rock	[-0.9642943351214...	0
TRAADYB128F92D7E73	[-0.2695728701843...	Jazz	[1.94681694919172...	0
TRAAGHM128EF35CF8E	[-0.2506608660050...	Electronic	[1.56064910499270...	1
TRAAGRV128F93526C0	[-0.8938581281452...	Pop_Rock	[-1.0192791745399...	0
TRAAGTO128F1497E3C	[-0.5502270122061...	Pop_Rock	[1.77949596780140...	0
TRAAHAU128F9313A3D	[-1.1467116240234...	Pop_Rock	[1.35810654894883...	0
TRAAHEG128E07861C3	[-0.0274992166884...	Rap	[-0.7901702143511...	0
TRAAHZP129O3CA25F4	[0.18998883137424...	Rap	[1.19315318033674...	0
TRAAICW128F1496C68	[-0.6391134318492...	International	[-0.7614220038015...	0
TRAAJJW129O3CBDDCB	[0.76113135759111...	International	[-1.6994555841035...	0
TRAAJKJ128F92FB44F	[-1.7095328684013...	Folk	[2.00752635489092...	0
TRAAKLX128F934CEE4	[-0.0350640183602...	Electronic	[-0.6558118190802...	1
TRAAKWR128F931B29F	[-1.6516621356125...	Pop_Rock	[1.93838696957098...	0
TRAALQN128E07931A4	[0.29211365394282...	Electronic	[-1.2201847882170...	1
TRAAMFF129O3CE81O7	[-0.2393136634973...	Pop_Rock	[-0.3470005160931...	0
TRAAMHG128F92ED7B2	[-0.1163856363315...	International	[-1.1342478381331...	0
TRAAROH128F42604B0	[-0.8567905999537...	Electronic	[-1.0094249048233...	1
TRAARQN128E07894DF	[1.88072200500963...	Pop_Rock	[1.02354601592778...	0

only showing top 20 rows

Figure 11: Pre-processed data with features and labels.

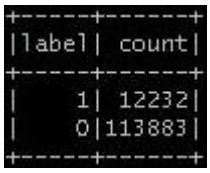
The class imbalance for this data was quite large, there were 40,662 observations with the genre “electronic” (class 1) and 379,942 observations with other genres (class 0). This meant that only about 11% of the data had the “electronic” label.

Dealing with imbalanced classes is a typical problem in machine learning. There are different methods to mitigate this problem. One way is to undersample the majority class or oversample the minority class to make the dataset more evenly balanced. Another way is to assign weights for each class to penalise the majority class by assigning less weight and boost the minority class by assigning a larger weight. For this problem, oversampling was implemented to resample with replacement from the data with "electronic" as the label. This was done by first splitting the data into train and test sets (we only wanted to oversample the train set), then splitting the data into the corresponding classes. The minority class was then resampled with replaced by the ratio of the two classes and the train sets were then joined back together. Below are the counts of the train sets before and after oversample.



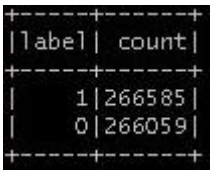
label	count
1	28430
0	266059

Figure 12a: Train set counts before oversampling.



label	count
1	12232
0	113883

Figure 12b: Train set counts after oversampling.



label	count
1	266585
0	266059

Figure 12c: Test set counts.

The logistic regression, random forest, and support vector machine models were then created using the Pyspark machine learning library and the data was fitted. Various functions in Pyspark were explored to find a way to conveniently compute the metrics of each model. We wanted to find the accuracy, precision, recall, and F1-score. The equations for each metric are as follows.

- True Positives – The number of correct classifications for the positive class.
- True Negatives – The number of correct classifications for the negative class.
- False Positives – The number of incorrect classifications for the positive class.
- False Negatives – The number of incorrect classifications for the negative class.
- Precision – The number of correct positive classification over the total number of positive classifications.
- Recall – The number of correct positive classifications over the number of positives.
- Accuracy – The number of correct classifications over the total number of observations.
- F1 Score – The harmonic average of precision and recall.

$$Precision = \frac{TP}{TP+FP} \qquad Recall = \frac{TP}{TP+FN}$$

$$Accuracy = \frac{TP+TN}{P+N} \qquad F_1\ Score = \frac{2*Precision*Recall}{Precision+Recall}$$

Figure 13: Metric Equations.

Although the multiclass classification evaluator has a form of weighted precision and recall, it is not appropriate nor well defined for this case. Documentation and formulas can be found on the older Pyspark MLLib docs, however, as we are not weighting towards a specific class, it is best to manually define and calculate the metrics for each label. We can do this by calculating the metrics true positive, true negative, false positive, false negative values using the above formulas. One function can be defined for both binary and multiclass classification where we compute the metrics for each

label. In the case of binary classification, the labels will be 1 (positive class) and 0 (negative class). The metrics for the three models are below.

Table 1: Binary classification test results for the three machine learning models.

Model	Accuracy (2dp)	Precision (2dp)	Recall (2dp)	F1 Score (2dp)
Logistic Regression	0.58	0.14	0.62	0.22
Random Forest	0.51	0.13	0.75	0.23
Linear SVM	0.54	0.13	0.68	0.22

As we can see, the logistic regression model had the highest accuracy, but also the lowest recall indicating a larger number of incorrectly classified positives. The random forest, however, had the highest recall and F1 score, which, for class-imbalanced data, are the metrics we would want to optimise.

Cross-validation can also be used to fine-tune the hyper-parameters of each model. It ensures that we are training and testing on every observation in the dataset. A parameter grid and validator are created to iterate through the range of different parameters and train and test models on each combination. In this case, a 5-fold cross-validator has been implemented. It is also worth noting that cross-validation takes a substantial amount of time to perform.

Table 2: Binary classification test results for the three machine learning models with cross-validation.

Model	Accuracy (2dp)	Precision (2dp)	Recall (2dp)	F1 Score (2dp)
Logistic Regression	0.58	0.14	0.63	0.23
Random Forest	0.68	0.17	0.58	0.26
Linear SVM	0.47	0.12	0.73	0.21

The most improvement in the evaluation of the models with cross-validation has occurred in the random forest model. We saw an increase in accuracy from 0.51 to 0.68 (2dp) and an increase in precision and F1 score of 0.13 to 0.17 and 0.23 to 0.26 (2dp), respectively. While the logistic regression model stayed more or less the same, indicating the hyper-parameters used initially we the most ideal, we did see a large decrease in accuracy for the linear support vector machine of 0.54 to 0.47 (2dp). Interestingly, the linear SVM exhibited an increase in recall from 0.68 to 0.73 (2dp) when cross-validation was applied which may signify that the model is classifying more observations as negatives at the cost of accuracy.

Overall, the linear support vector machine performed the worst out of the three models due to the class imbalance. Standard scaling and principal component analysis were used to normalise the scale of the variables and decrease the dimensionality of the data. The logistic regression model outperformed the others when cross-validation was not implemented, however, the cross-validated random forest model was the most adept at the classification of the binary class when considering accuracy and F1 score as the important metrics to optimise. The class imbalance definitely played a part in reducing the effectiveness of each model despite oversampling the minority class. In future work, it would be ideal to compare various resampling techniques involving a combination of undersampling and synthetic oversampling like SMOTE or ROSE with a variety of different cross-validated models.

Question 3

In the multiclass classification, we have two options to consider when training the data. These are known as One-versus-All (or One-versus-Rest) and One-versus-One.

- One-versus-All (OvA)

One-versus-All involves training a single classifier per class, with the samples of that class as positive samples and all other samples as negatives. In other words, we would train an individual classifier for each genre, where said genre (e.g. 'Pop-Rock') is the positive class and all other classes are grouped into the negative class. This effectively turns the classification into a binary problem. This method, however, suffers from several problems. Firstly, the scale of the confidence values may differ between the binary classifiers. Secondly, even if the class distribution is balanced in the training set, the binary classifiers will be dealing with unbalanced distributions because the size of the negative class will be much greater than the size of the positive class. In Pyspark, we can implement the One-versus-All method using the 'pyspark.ml.classification.OneVsRest' function. For logistic regression or random forest, this method would work particularly well as these models are able to handle class imbalance and outliers. The support vector machine will likely struggle with OvA due to this issue.

- One-versus-One (OvO)

One-versus-One involves training a binary classifier for every pair of class. For example, if we have three classes (0, 1, 2), we will treat each pair as a binary problem and train N(N-1)/2 classifiers for N classes accordingly. So, three models for classes 0 – 1, 0 – 2, and 1 – 2. This is much less sensitive to the problems of imbalanced datasets but is much more computationally expensive.

When you then want to classify images, you need to run each of these 45 classifiers and choose the best performing one. This strategy has one big advantage over the others and this is, that you only need to train it on a part of the training set for the 2 classes it distinguishes between. Algorithms like Support Vector Machine Classifiers don't scale well at large datasets, which is why in this case using a binary classification algorithm like Logistic Regression with the OvO strategy would do better, because it is faster to train a lot of classifiers on a small dataset than training just one at a large dataset. This could be implemented in Pyspark by creating a binary classifier (e.g. logistic regression or random forest) and training on the pairwise combinations of classes in the dataset. The problem here is that, for large numbers of classes, the computation time will grow exceedingly large. This means that classifiers such as the support vector machine will be far too slow to deal with the number of pairs. Ideally, one would use a logistic regression model or a random forest.

Following on with the multiclass classification processing, to encode each of the genre labels, we can use the string indexer function from Pyspark. This will ensure each unique value in the genres column is given as an integer.

When considering our options for resampling, we find we have four possibilities; no resampling, undersample to the lowest count, oversample to the highest count, or over and undersample to some average of the number of observations. As neither option seems to be the best path to take, we will implement the method that would hopefully lead to the least amount of unbalance: resample all classes to the average of the counts.

To resample each class for multiclass classification, we first split the data into train and test sets then calculate the mean value of the class counts in the train set. We then iterate through each class: if the positive class count is larger, undersample. If the class count is smaller, oversample. The ratio by which to oversample or undersample is calculated by the ratio of the mean count to the class count.

The result is a training dataset containing approximately the same number of samples for each class without needing to drastically over or undersample any class.

Genre	label	count
Pop_Rock	0.0	237641
Electronic	1.0	40662
Rap	2.0	20899
Jazz	3.0	17775
Latin	4.0	17504
RnB	5.0	14314
International	6.0	14194
Country	7.0	11691
Religious	8.0	8779
Reggae	9.0	6928
Blues	10.0	6801
Vocal	11.0	6182
Folk	12.0	5789
New Age	13.0	4000
Comedy_Spoken	14.0	2067
Stage	15.0	1613
Easy_Listening	16.0	1535
Avant_Garde	17.0	1012
Classical	18.0	555
Children	19.0	463
Holiday	20.0	200

Figure 14: Distribution and labels of each genre.

label	count
0.0	166387
1.0	28430
2.0	14777
3.0	12538
4.0	12175
5.0	9968
6.0	9953
7.0	8143
8.0	6138
9.0	4866
10.0	4708
11.0	4310
12.0	4033
13.0	2829
14.0	1444
15.0	1162
16.0	1074
17.0	705
18.0	376
19.0	332
20.0	141

Figure 15a: Train set counts before resampling.

label	count
0.0	14114
1.0	13942
2.0	13903
3.0	13986
4.0	13993
5.0	13996
6.0	14013
7.0	13954
8.0	14046
9.0	13845
10.0	13942
11.0	13864
12.0	13910
13.0	13911
14.0	13974
15.0	13881
16.0	13941
17.0	13973
18.0	13943
19.0	14133
20.0	14025

Figure 15b: Train set counts after resampling.

label	count
0.0	71254
1.0	12232
2.0	6122
3.0	5237
4.0	5329
5.0	4346
6.0	4241
7.0	3548
8.0	2641
9.0	2062
10.0	2093
11.0	1872
12.0	1756
13.0	1171
14.0	623
15.0	451
16.0	461
17.0	307
18.0	179
19.0	131
20.0	59

Figure 15c: Test set counts.

Three separate training cases were performed for the One-versus-All algorithm: random forest without resampling, random forest with resampling, and random forest with resampling and cross-

validation. The results of the One-versus-All random forest without resampling were not overly surprising. The model was able to classify the first and largest class, 'Pop Rock', with an accuracy and F1 score of 0.57 and 0.72 (2dp), respectively, and similarly was able to recognise the second largest genre 'Electronic' with an accuracy and F1 score of 0.90 and 0.03 (2dp), respectively, however, it failed to classify any other observation in the dataset. This lead to all other labels having an accuracy of over 95% (predicting only classes 0 and 1) while all other metrics were 0. The model essentially ignored the other labels due to the class imbalance.

The One-versus-All random forest model with resampling demonstrated a large improvement in the classification of nearly all other classes due to their being many more observations to train on. Although four out of the 21 genres were still ignored (F1 score = precision = recall = 0.0), the majority of the other labels were picked up with reasonable accuracy. This model would likely improve with some weighting towards the F1 Score metric which would optimise for the average of the precision and recall of the model. Overall, the class with the highest F1 Score but lowest accuracy was the 'Pop Rock' genre.

When the One-versus-All resampled random forest model was cross-validated, we saw a slight increase in the F1 score of some label classes, for example, the F1 score for labels 2 and 5 increased from 0.12 and 0.00 to 0.16 and 0.10 (2dp), respectively. In addition, only two out of the 21 genres were not classified whereas, without cross-validation, four labels were ignored. This, however, came as a small cost in accurately classifying other classes. Overall, cross-validation improved the One-versus-All model. Including multiple classes in the data means the model is trying to distinguish between more classes which leads to an overall decrease in the accuracy, F1 score, precision, and recall of the classification of each label.

Song Recommendations

Question 1

In this section, we will be working with the Taste Profile dataset to develop a song recommendation system based on collaborative filtering. Collaborative filtering describes algorithms that generate songs recommendations for specific users based on the combined user-song play information from all users. These song recommendations are generated by embedding users and songs as numerical vectors in the same vector space and selecting songs that are similar to the user based on cosine similarity.

Within the Taste Profile dataset, there are 378,310 and 1,019,318 unique songs and unique users, respectively. The most active user has listened to 195 unique songs with 13,074 total plays. We can

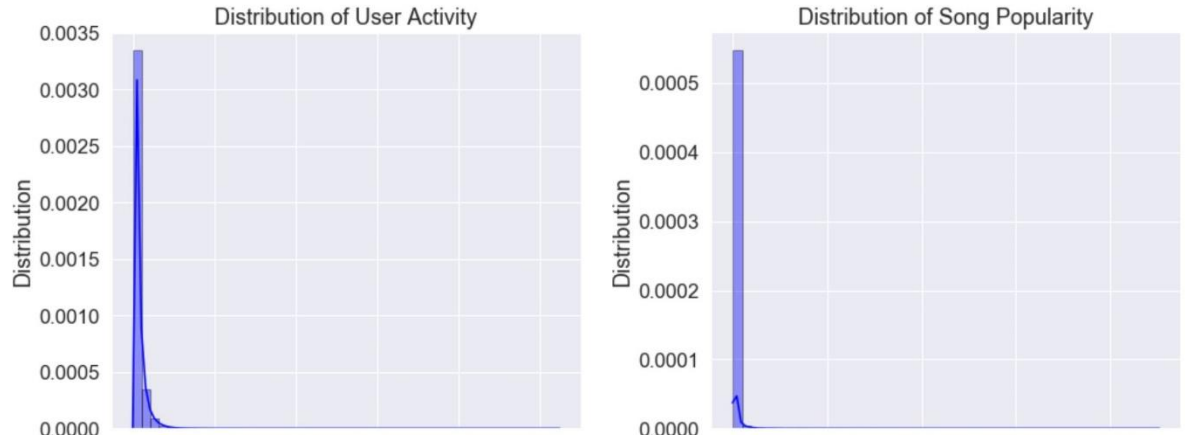


Figure 15: Distribution of user activity and song popularity.

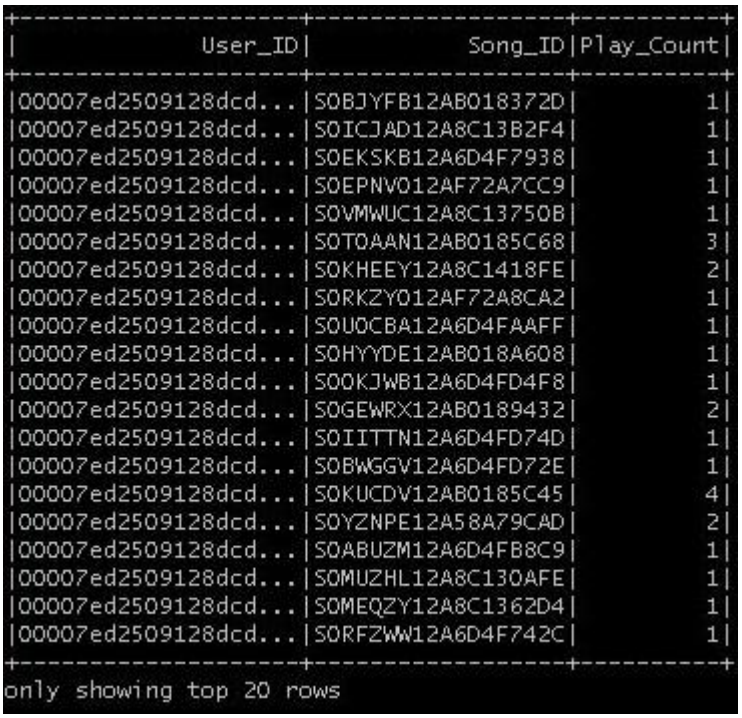
define song popularity as the number of unique users per song. We also could say that song popularity is the number of plays per song, but one person could have listened to a song one million times and that would not make it a popular song. Furthermore, we can define the user activity as the sum of the number of plays for each unique user. So, a user could listen to one song 100 times, or 100 songs once and it would give the same result. The distribution plots for the song popularity and user activity are below.

The distribution of both datasets is heavily skewed to the right. This is because we have a few observations with high counts (e.g. number of plays or number of users) which pull the distribution of the data. As songs which have been played only a few times and users who have only listened to a few songs will not contribute much information to the overall dataset and are unlikely to be recommended, we can remove user-song plays which have been played less than N times and users who have listened to fewer than M songs. We will define N and M as the mean of the number of unique songs played by each user and the mean of the total plays for each song, respectively. So, we will filter out songs that have less than the mean total plays and users who have listened to less than the mean number of songs.

The mean of these grouped datasets was decided as the rejection region as it seemed to be a fair statistic for heavily skewed data. These datasets are then left-semi joined back onto the Taste Profile dataset sequentially. The left-semi join only returns observations from the left-hand data frame that match with the right-hand data frame so it does not result in extra columns and rows occurring from joining with a grouped dataset. However, if the two filtered data frames are both joined back on to Taste profile at the same time, we end up with users who may have listened to only one song as this song ID would have been brought over from the data frame of wanted songs. To avoid this, the data frame of songs with plays greater than the mean was joined on to the Taste Profile data frame, then this was used as the data frame we would be filtering to get the users who have listened to more than the mean number of unique songs. Finally, this was joined back on to the modified Taste Profile data frame from before.

Next, we will do the train test split for the collaborative filtering model. Due to the nature of the collaborative filtering model, we must ensure that every user in the test set has some user-song plays in the training set as well. This is because we want to train on the song plays for each user then try to predict which songs each user may want to listen to. To validate this, we need to test on song plays for the same users. Therefore, we need observations for song plays for each unique user in both the train and test splits. Creating this modified test split is a little more difficult. The approach taken is as follows:

Let *matches_clean* be the filtered Taste Profile matches data from the previous join.



User_ID	Song_ID	Play_Count
00007ed2509128dcd...	SOBJYFB12AB018372D	1
00007ed2509128dcd...	SOICJAD12A8C13B2F4	1
00007ed2509128dcd...	SOEKSKB12A6D4F7938	1
00007ed2509128dcd...	SOEPNVO12AF72A7CC9	1
00007ed2509128dcd...	SOVMWUC12A8C13750B	1
00007ed2509128dcd...	SOTOAAN12AB0185C68	3
00007ed2509128dcd...	SOKHEEY12A8C1418FE	2
00007ed2509128dcd...	SORKZYO12AF72A8CA2	1
00007ed2509128dcd...	SOUOCBA12A6D4FAAFF	1
00007ed2509128dcd...	SOHYYDE12AB018A608	1
00007ed2509128dcd...	SOOKJWB12A6D4FD4F8	1
00007ed2509128dcd...	SOGEWRX12AB0189432	2
00007ed2509128dcd...	SOIITTN12A6D4FD74D	1
00007ed2509128dcd...	SOBWGGV12A6D4FD72E	1
00007ed2509128dcd...	SOKUCDV12AB0185C45	4
00007ed2509128dcd...	SOYZNPE12A58A79CAD	2
00007ed2509128dcd...	SOABUZM12A6D4FB8C9	1
00007ed2509128dcd...	SOMUZHL12A8C130AFE	1
00007ed2509128dcd...	SOMEQZY12A8C1362D4	1
00007ed2509128dcd...	SORFZWW12A6D4F742C	1

only showing top 20 rows

Figure 16: Cleaned Taste Profile matches dataset.

- Create a new data frame called *train_collab* by replication *matches_clean* and dropping duplicate rows keyed on the User ID column. The *matches_clean* data frame has 308,225 unique User IDs so we expect this same number of rows in *train_collab* to begin with.
- Next, we remove the observations in *train_collab* from *matches_clean* and drop the duplicate rows of the User ID column to get the *test_collab* data frame. We now have a train and test data frame with exactly one observation for each user.
- We then remove the observations from both *train_collab* and *test_collab* from the *matches_clean* data frame to get the remaining observations that we will train test split.
- We perform the 80%-20% train test split on that previous data frame and union both of these splits with *train_collab* and *test_collab*. The result is a train and test split for the user-song plays dataset with 79% of the data in the train set and 21% in the test set, and at least one observation per unique User ID in both sets.

This method for train test splitting the data ensuring at least one user-song play observation in both sets is as equally random as the standard train test split function. This is because we are using the 'dropDuplicates' function which selects a pseudo-random instance of each unique key (in this case by User ID). So, we will select some random observation for each unique user ID when we initially create the *train_collab* and *test_collab* sets. We then do a random train test split on the remaining data to populate the train test splits.

User_ID	Song_ID	Play_Count	User_ID_encoded	Song_ID_encoded
00004fb90a86beb8b...	SOTEFFR12A8C144765	1	275161.0	345.0
00004fb90a86beb8b...	SOYEQLD12AB017C713	1	275161.0	1149.0
00004fb90a86beb8b...	SOGVKXX12A67ADA0B8	1	275161.0	301.0
00004fb90a86beb8b...	SOSXLTC12AF72A7F54	1	275161.0	2.0
00004fb90a86beb8b...	SOXLKNJ12A58A7E09A	3	275161.0	574.0
00004fb90a86beb8b...	SOBIMTY12A6D4F931F	1	275161.0	226.0
00004fb90a86beb8b...	SOWNIUS12A8C142815	2	275161.0	840.0
00004fb90a86beb8b...	SOWQLXP12AF72A08A2	2	275161.0	1663.0
00004fb90a86beb8b...	SOIGHWG12A8C136A37	2	275161.0	1210.0
00004fb90a86beb8b...	SOFFWDQ12A8C13B433	3	275161.0	1729.0
00004fb90a86beb8b...	SOUZBUD12A8C13FD8E	1	275161.0	949.0
00004fb90a86beb8b...	SOPKPUK12A8C13CAED	3	275161.0	1229.0
00004fb90a86beb8b...	SOPWKOX12A8C139D43	4	275161.0	1492.0
00004fb90a86beb8b...	SOUGLUN12A8C14282A	1	275161.0	4380.0
00004fb90a86beb8b...	SOJGSIO12A8C141DBF	1	275161.0	266.0
00004fb90a86beb8b...	SOWRMTT12A8C137064	1	275161.0	727.0
00004fb90a86beb8b...	SOWCKVR12A8C142411	1	275161.0	5.0
00004fb90a86beb8b...	SOGFFET12A58A7ECA9	1	275161.0	187.0
00004fb90a86beb8b...	SOPHBRE12A8C142825	2	275161.0	834.0
00004fb90a86beb8b...	SOOLKLP12AF729D959	2	275161.0	889.0

only showing top 20 rows

Figure 17: Final train data for the collaborative filtering model.

User_ID	Song_ID	Play_Count	User_ID_encoded	Song_ID_encoded
00004fb90a86beb8b...	SOLCSYN12AF72A049D	1	275161.0	1181.0
00004fb90a86beb8b...	SORRCNC12A8C13FDA9	1	275161.0	270.0
00004fb90a86beb8b...	SOBJCFV12A8AE469EE	1	275161.0	1677.0
00004fb90a86beb8b...	SOWGIBZ12A8C136A2E	3	275161.0	481.0
00004fb90a86beb8b...	SOGDQWF12A67AD954F	1	275161.0	1851.0
00004fb90a86beb8b...	SOGZCOB12A8C14280E	2	275161.0	365.0
00004fb90a86beb8b...	SOQFXDQ12AF72ADOEE	1	275161.0	2005.0
0000bb531aaa657c9...	SOCRUVF12A6D4F5906	1	122782.0	3501.0
0000bb531aaa657c9...	SOMOSXL12A6701E5DC	2	122782.0	4639.0
0000bb531aaa657c9...	SOLTRNQ12ABO181C9E	1	122782.0	13414.0
0000bb531aaa657c9...	SOKUKJB12A6701C31C	2	122782.0	11331.0
0000bb531aaa657c9...	SOCDRUZ12A8AE48614	1	122782.0	6357.0
0000bb531aaa657c9...	SOIDDVK12A6701C53E	1	122782.0	3084.0
0000bb531aaa657c9...	SODREUL12ABO18D6C3	2	122782.0	5413.0
0000bb531aaa657c9...	SOTLURY12ABO183C93	1	122782.0	3296.0
0000bb531aaa657c9...	SOLTRNQ12ABO181C9E	1	122782.0	13414.0
0000bb531aaa657c9...	SOMFLEZ12A6D4F907C	1	122782.0	12584.0
0000bb531aaa657c9...	SOKBHQN12A67ADBAE2	2	122782.0	12204.0
0000bb531aaa657c9...	SOYRVSP12A6D4F907A	1	122782.0	2919.0
0000d3c803e068cf1...	SOIVBNO12A58A75411	1	212211.0	2516.0

only showing top 20 rows

Figure 18: Final test data for the collaborative filtering model.

Question 2

An implicit matrix factorisation model was then trained on the new collaborative filtering train data using alternating least squares (ALS). The model was then used to predict recommendations using the test set. The root mean squared error was calculated to simply evaluate the performance of the implicit collaborative filtering model and an RMSE of 5.629 (3dp) was found.

User_ID	Song_ID	Play_Count	User_ID_encoded	Song_ID_encoded	prediction
d8e6fa08d73821f30...	SOAOSDF12A58A779F1	1	57689.0	1584.0	2517.9683
6b36f65d2eb5579a8...	SOMEJEH12A6D4FD320	1646	52570.0	1667.0	1589.1185
11f613e16d2d89632...	SODUWZY12AB0183594	15	254606.0	51695.0	1066.44
f48fe9518cbe2e523...	SOJENYR12A6D4FD2A9	785	80263.0	51698.0	784.0228
c35a8573cd0eee934...	SOAGHUN12A58A7C884	752	26712.0	50841.0	746.5358
093cb74eb3c517c51...	SOFCXNF12A58A7CF16	2	19761.0	5848.0	690.8863
dcf6a11b2fea3af24...	SOYFYHE12A8C142082	686	262854.0	31341.0	674.27985
f3825861e5575cf42...	SOPURQO12A8C13EC4C	718	645.0	5910.0	661.994
174b880b7340e2921...	SOBSGZI12A8C13F8DF	716	22886.0	2424.0	659.2329
3f2049db6087a5ccf...	SOSJQZN12A8C14196C	550	123034.0	49618.0	558.30914
70caceccaa745b6f7...	SOPREHY12AB01815F9	676	300718.0	143.0	554.59674
0421d096b0c80ec28...	SOJUYSA12AB01837F8	526	180546.0	11870.0	529.9474
ea823a3da3b2d9801...	SOLLDMS12AB01859C0	531	305243.0	51554.0	527.59827
1c4275c8a7ebc62ce...	SOBIZDN12A6D4F86C7	506	148488.0	31973.0	491.90244
1c4275c8a7ebc62ce...	SOBIZDN12A6D4F86C7	506	148488.0	31973.0	491.90244
cb5c18801cda41033...	SOKNNOL12A8C13C564	510	120243.0	6944.0	466.58353
760e7f67b00171895...	SONSTND12AB018516E	33	222397.0	44233.0	461.59357
0421d096b0c80ec28...	SOUDLVN12AAFF43658	1	180546.0	142.0	444.1599
6407eeebb130e381e...	SODBTXH12AB01896E8	3	255228.0	1870.0	442.63586
ac662a5edc8d0f143...	SOWPFEC12A8C139607	472	270559.0	4190.0	439.31384

only showing top 20 rows

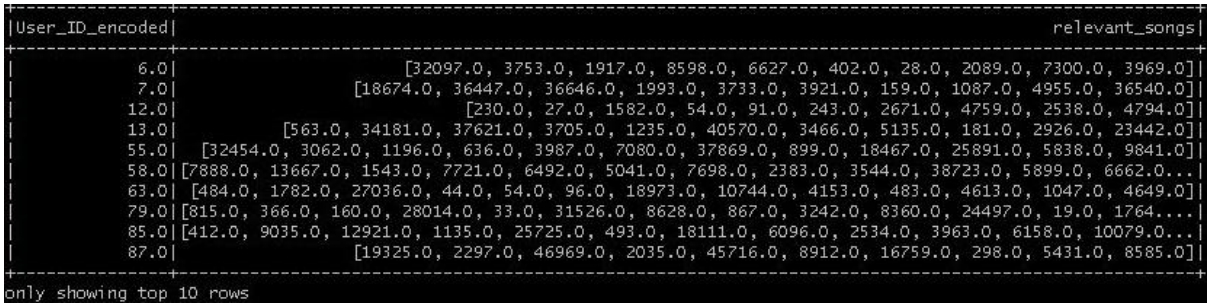
Figure 19: Data frame of user-song plays and predictions.

We then generated some song recommendations for all users and compared the recommendations with the songs actually listened to by the user. A subset of this was created to find the song recommendations for a given number of users.

User_ID_encoded	recommended_songs
6.0	[32097.0, 45271.0, 287.0, 37.0, 1917.0, 5740.0, 16704.0, 2432.0, 59.0, 24909.0]
7.0	[11.0, 1967.0, 1093.0, 108.0, 46084.0, 15064.0, 152.0, 86.0, 228.0, 159.0]
12.0	[333.0, 27.0, 11243.0, 247.0, 214.0, 7075.0, 14996.0, 91.0, 218.0, 243.0]
13.0	[40.0, 4502.0, 11786.0, 563.0, 34181.0, 228.0, 37621.0, 42307.0, 2926.0, 2283.0]
55.0	[46315.0, 1580.0, 32454.0, 1377.0, 3987.0, 6257.0, 703.0, 18467.0, 17898.0, 45714.0]
58.0	[2256.0, 20410.0, 3883.0, 488.0, 576.0, 16303.0, 19512.0, 2029.0, 26165.0, 41717.0]
63.0	[31082.0, 96.0, 313.0, 20486.0, 483.0, 25.0, 131.0, 484.0, 8560.0, 19.0]
79.0	[3.0, 11320.0, 45516.0, 3242.0, 19.0, 10073.0, 31526.0, 27588.0, 80.0, 863.0]
85.0	[147.0, 50.0, 43432.0, 4497.0, 50297.0, 44277.0, 20249.0, 12874.0, 16882.0, 4273.0]
87.0	[46969.0, 8912.0, 25645.0, 46296.0, 8085.0, 28835.0, 483.0, 2035.0, 45716.0, 19517.0]

only showing top 10 rows

Figure 20: Recommended songs per user.



User_ID_encoded	relevant_songs
6.0	[32097.0, 3753.0, 1917.0, 8598.0, 6627.0, 402.0, 28.0, 2089.0, 7300.0, 3969.0]
7.0	[18674.0, 36447.0, 36646.0, 1993.0, 3733.0, 3921.0, 159.0, 1087.0, 4955.0, 36540.0]
12.0	[230.0, 27.0, 1582.0, 54.0, 91.0, 243.0, 2671.0, 4759.0, 2538.0, 4794.0]
13.0	[563.0, 34181.0, 37621.0, 3705.0, 1235.0, 40570.0, 3466.0, 5135.0, 181.0, 2926.0, 23442.0]
55.0	[32454.0, 3062.0, 1196.0, 636.0, 3987.0, 7080.0, 37869.0, 899.0, 18467.0, 25891.0, 5838.0, 9841.0]
58.0	[7888.0, 13667.0, 1543.0, 7721.0, 6492.0, 5041.0, 7698.0, 2383.0, 3544.0, 38723.0, 5899.0, 6662.0...
63.0	[484.0, 1782.0, 27036.0, 44.0, 54.0, 96.0, 18973.0, 10744.0, 4153.0, 483.0, 4613.0, 1047.0, 4649.0]
79.0	[815.0, 366.0, 160.0, 28014.0, 33.0, 31526.0, 8628.0, 867.0, 3242.0, 8360.0, 24497.0, 19.0, 1764....
85.0	[412.0, 9035.0, 12921.0, 1135.0, 25725.0, 493.0, 18111.0, 6096.0, 2534.0, 3963.0, 6158.0, 10079.0...
87.0	[19325.0, 2297.0, 46969.0, 2035.0, 45716.0, 8912.0, 16759.0, 298.0, 5431.0, 8585.0]

only showing top 10 rows

Figure 21: Actual song plays per user.

As we can from the above figures, the recommendation system has correctly predicted several user-song plays. Although the order is not perfect, for example, for user ID 6, song ID 1917 was predicted to be ranked 5th, however, was actually ranked 3rd, it does show that the model is performing appropriately. We can then evaluate its performance in more detail using the information retrieval metrics described below.

- Precision @ 5
    - Precision at k is a measure of how many of the first k recommended documents are in the set of true relevant documents averaged across all users. In this metric, the order of the recommendations is not taken into account.
    - As precision is a metric used to evaluate binary classification models, we need a way to translate the numerical problem of having some rating at rank, k, into a binary problem where we evaluate whether a recommended item is relevant or not, given a threshold. In the context of recommender systems, we compute the precision for the top N items recommended to the user. This means we define a k that corresponds to the top N items. Therefore, precision @ 5 is the proportion of recommended items in the top 5 set that are relevant
    - The limitations of the precision @ k metric are that the value of k is somewhat arbitrary to choose and has a large impact on the metric. Furthermore, the ranking of the value of k is inconsequential. Lastly, when computing the precision @ k, we are dividing the number of items recommended in the top k recommendations. So, in cases where no items are recommended, we cannot calculate this metric.

- NDCG @ 10
    - Normalised Discounted Cumulative Gain is a metric used to measure ranking quality. The idea is that highly relevant items appearing lower in a search result list should be penalised as the relevance value is reduced proportionally to the position of the result. As the discounted cumulative gain – the sum of the relevance of each recommendation relative to its position, as described above – is not directly comparable between users, so we must normalise them. Therefore, we arrange all the items in the test set in the ideal order, take first K items and compute DCG for them. Then we divide the raw DCG by this ideal DCG to get NDCG@K, a value between 0 and 1.
    - NDCG will consider multiple levels of relevance and focus of the top ranks. The main difficulty encountered in using NDCG is the unavailability of an ideal ordering of results when only partial relevance feedback is available. Furthermore, NDCG does not penalise for bad recommendations in the result. In other words, if a query

returns two results with scores (1, 1, 1) and (1, 1, 1, 0) respectively, both would be considered equally good even if the latter contains a bad document.

- Mean Average Precision (MAP)
    - Mean Average Precision is a measure of how many of the recommended documents are in the set of true relevant documents, where the order of the recommendations is taken into account (i.e. the penalty for highly relevant documents is higher). This is a useful metric for recommender systems as we want to evaluate the ranked ordering of the recommendations as being relevant or not. Furthermore, where the recommendations occurred relative to the actual rank of the items is important to consider. We want to penalise the model if incorrect guesses appear higher up the ranks.
    - One of the disadvantages of MAP is that it is among the least stable of the commonly used evaluation measures and that it does not average well since the total number of relevant documents for a query has a strong influence on precision at k.

Table 3: Ranking metrics for the collaborative filtering model.

Metric	Value (3dp)
Precision @ 5	0.907
NDCG @ 10	0.906
Mean Average Precision	0.699

As displayed in the table above, the collaborative filtering model has performed reasonably well on the test data. This is likely due to the aggressive filtering of songs and users made earlier and also due to correctly recommending high ranking songs for a number of users.

An alternative method for comparing two recommendation systems in the real world could be to measure coverage or diversity of the systems. This would allow us to evaluate how the models are considering items in the long tail of the frequency distribution. In other words, the songs that are new or hardly ever played as they are less known, while still keeping the recommendations as relevant as possible for each user.

- Coverage would measure the capacity of the system to recommend on the entire collection, for example, by trying to compute a score for every item and for every user. However, not only would this be computationally expensive for a collaborative filtering model, especially if there are a large number of users and items, it may also not be able to provide scores for items whose metadata does not provide enough match. Another, more practical measure, would be ‘effective coverage’, which is the share of items that come up in the top N results across all users.
- Diversity measures the capacity of the engine to go beyond the typically recommended items and recommend others that span the feature space of items, instead of being limited to a narrow subspace. This would require measuring item similarity and using it to compute the diversity between users, in other words, how different are recommendations across the Top-N results for each user.

Assuming we can measure future user-song plays based on recommendations, another metric that may be useful for evaluating the recommender system could be Mean Average Recall (MAR). This differs from Mean Average Precision in that MAP gives insight into how relevant the list of

recommended items is, whereas Mean Average Recall gives insight into how well the recommender is able to recall all the items the user has rated positively in the test set. This is equivalent to the number of relevant songs recommended by the model divided by the total number of existing relevant songs for that user, then averaged over all users. The difference between measuring future user-song plays based on the recommendations and evaluating the model using a test set, is that the user has now been suggested recommended songs to choose from, meaning their actual song plays will be closer to the recommendations as they have been given this new knowledge, whereas before we were trying to predict the songs the user may listen to without any feedback.

In the real world, however, we may wish to define a measure more relevant to the business objective, such as revenue, conversion, or bounce, etc. This could also be, for example, whether a user listened to a song for more than 30 seconds before changing to another song, or whether the user listened to a recommended song multiple times. One can use AB testing to compare how a recommendation system performs with respect to these metrics. For example, a new form of feedback added to the model took into consideration whether a user repeated a song before moving on to the next one. Songs that were repeated in this fashion had a higher probability of being liked the first time and a repeated listen ensured a user became 'hooked' on a song. Both the old recommendation system and the new model were AB tested and their respective revenues were compared. With that said, these metrics are just as important as the statistically defined ones mentioned above as they can provide real feedback for the recommender system and value for the company at a more heuristic level, while the statistically defined metrics such as NDCG@K can provide feedback as to how the recommendation system performs relative to each user's actual song plays at a micro level.

Question 3

Spotify is one of the largest music streaming services available with over 217 million customers worldwide. New users are joining and new music is being added daily. Due to the overwhelming sparsity in the data, the collaborative filtering model implemented in this task would struggle to deal with the new users who have low song-play counts and the new songs which have not been played many times. For a new user with zero plays, the collaborative model would be unable to produce any recommendations for that user. This is because the model has not received sufficient implicit feedback regarding which music the user may enjoy, so, we will have a sparse user-item matrix with no song values for the user. This issue is known as cold start computing and is when a recommendation system cannot make any inference about a user without having received enough information. The main strategy in dealing with new users who have had no interaction with the system is to ask them to provide some feedback or preferences to build up their initial profile. This may involve asking them to list some number of songs they have enjoyed listening to recently.

We would expect the collaborative filtering model to produce low-quality recommendations when the sparsity of the matrix is large. This is often caused by a low number of interactions between users and songs which leads to a difficulty in finding recommendations for users with unique tastes. Known as the popularity bias, the variation in user interest and sparsity in the data are overly large which results in low-quality recommendations.

A recommendation system that could handle new users, and essentially the 'cold start' problem, would be one that combined explicit and implicit techniques. At the moment a new user creates a profile, by default, they have had no interactions with the system, so it is unable to provide any personalised recommendations. If the user is requested to fill out a quick survey regarding a few items they may enjoy, such as their favourite music genres or songs, the recommendation system

can use this explicit feedback to create an initial basic profile about the user. In this case, a content-based model could be implemented. A user-item recommendation engine would compare items that the user rated positively with similar items that have not yet been seen by the user to make recommendations. Similarly, a user-user content-based approach could be used where the algorithm will rely on user's features (e.g. age, gender, location) to find similar users and recommend the items they interacted with in a positive way, therefore being robust to the new use case. This data must be acquired during the registration process, either by asking the user to input the data or by leveraging data already available, e.g. from social media accounts.

On the other hand, to deal with low-quality recommendations caused by large sparsity in the data, a hybrid filtering approach could be implemented. In this case, it would involve combining collaborative and content-based filtering models such that the user could be recommended songs either by comparing their listening and search habits (collaborative) or by suggesting songs that share characteristics with other songs that the user has listened to (content-based). There is still a lot of research into how to manage data sparsity in recommendation systems, however, another promising technique involves extracting pairwise association rules to build a collective model of preferences using interactions performed by users from a given population.

Based on the song metadata, various other columns could be included to improve the real-world performance of the recommendation system. For example, the columns related to song and artist popularity (hotness) could be used to give some insight into what music is currently trending. This may allow the system to rank certain artists or songs higher depending on the music charts. It may also be possible to validate the recommendations using the song popularity – if a song is trending, it is more likely that other users would want to listen to it, therefore, it should be recommended more. Furthermore, the column featuring similar artist indices would allow us to tag artists will others that are comparable. If a user likes a song from a particular artist, they are more likely to enjoy music from similar artists. Often users will click on the similar artists' tags in Spotify, so, having this automatically recommended to the user would save the user time in finding new music, enhance the experience of the service, and add more validated data to the model.