6/9/2023

The Million Song Dataset

Data420 Assignment 2

Rayne Davidson
12444974

Contents

Data Processing

Data Structure

The data used here was data acquired from the Million Song Dataset (msd), which was accessed from hdfs:///data/msd/ directory. There are four folders within the msd: audio, genre, main, and tasteprofile. Inside of the audio folder, there are three files pertaining to audio features for each track, as measured and classified by Vienna University of Technology from <60 second song samples. The genre folder contains three files pertaining to the genre and style classifications for each track, as classified by Vienna University of Technology. The main folder has another folder within it, summary, that has two files, analysis and metadata, which have the baseline metadata for each track. Finally, tasteprofile has a file for triplets, which is where a user, song, and plays have all been correctly matched, and a folder for mismatches.

Ultimately, the data is pre-partitioned into eight files of reasonable sizes, with no sizes much larger than 150 Mb. All files are either .csv or .csv.gz, gzipped for compression Since the data is already pre-partitioned into reasonably sized files, and gzip files are not splitable, repartitioning the data in Spark was not necessary. Table 1 shows the count of rows within each of the audio datasets and how they compare to the total count of unique songs. This was meant to be all dataset but was misinterpreted.

Table 1 Counts of total rows in each dataset compared to counts of unique songs in each audio dataset

DATASET	ROWS	UNIQUE COUNT	ROWS DIFF FROM UNIQUE
AREA OF MOMENTS	994623	994604	-19
LPC	994623	994623	0
METHODS OF MOMENTS	994623	994623	0
MFCC	994623	994623	0
SPECTRAL	994623	994623	0
SPECTRAL DERIVATIVES	994623	994623	0
MARSYAS TIMBRAL	995001	995000	1
MVD	994188	994175	13
RH	994188	994175	13
RP	994188	994175	13
SSD	994188	994175	13
TRH	994188	994175	13
TSSD	994188	994175	13

Mismatched Data

Using code provided by James Williams the songs that were mismatched in the taste profile dataset were removed. Figure 1 shows a summary of the process. Additionally, each feature dataset was given its appropriate column names using the attributes dataset.

```
matches_manually_accepted = 488
mismatches                = 19094
triplets                  = 48373586
triplets_not_mismatched   = 45795111
```

Figure 1 Summary of taste profile dataset mismatches

Audio Similarity

Methods of Moment

Based on the summary of Methods of Moments (Appendix A) that showed a wide range of feature attribute values, the feature values were scaled to reduce the range of intensities. The standardized features were then checked for correlation with each other. The columns of “Methods of Moment Average 4” and “Methods of Moment Average 5” were found to be highly correlated, with a normalized correlation value of 0.98. “Methods of Moment Standard Deviation 4” and “Methods of Moment Standard Deviation 5” also exhibited strong correlation, with a normalized correlation value of 0.94.

The “MSD All Music Genre Dataset” contains a table with the track ID’s of all of the songs available in the dataset, and what genre they are categorized as, out of twenty genres, if available. Using the genre dataset, the total number of songs in each genre was compiled to create the distribution of genres, as seen in figure 2. The top plot in figure shows the raw distribution of genres, where pop-rock dominates in popularity, and the lower half of genres are indistinguishable on the scale needed to show pop-rock. The bottom figure has been scaled by log base 2, to allow for the full range of values to be displayed.

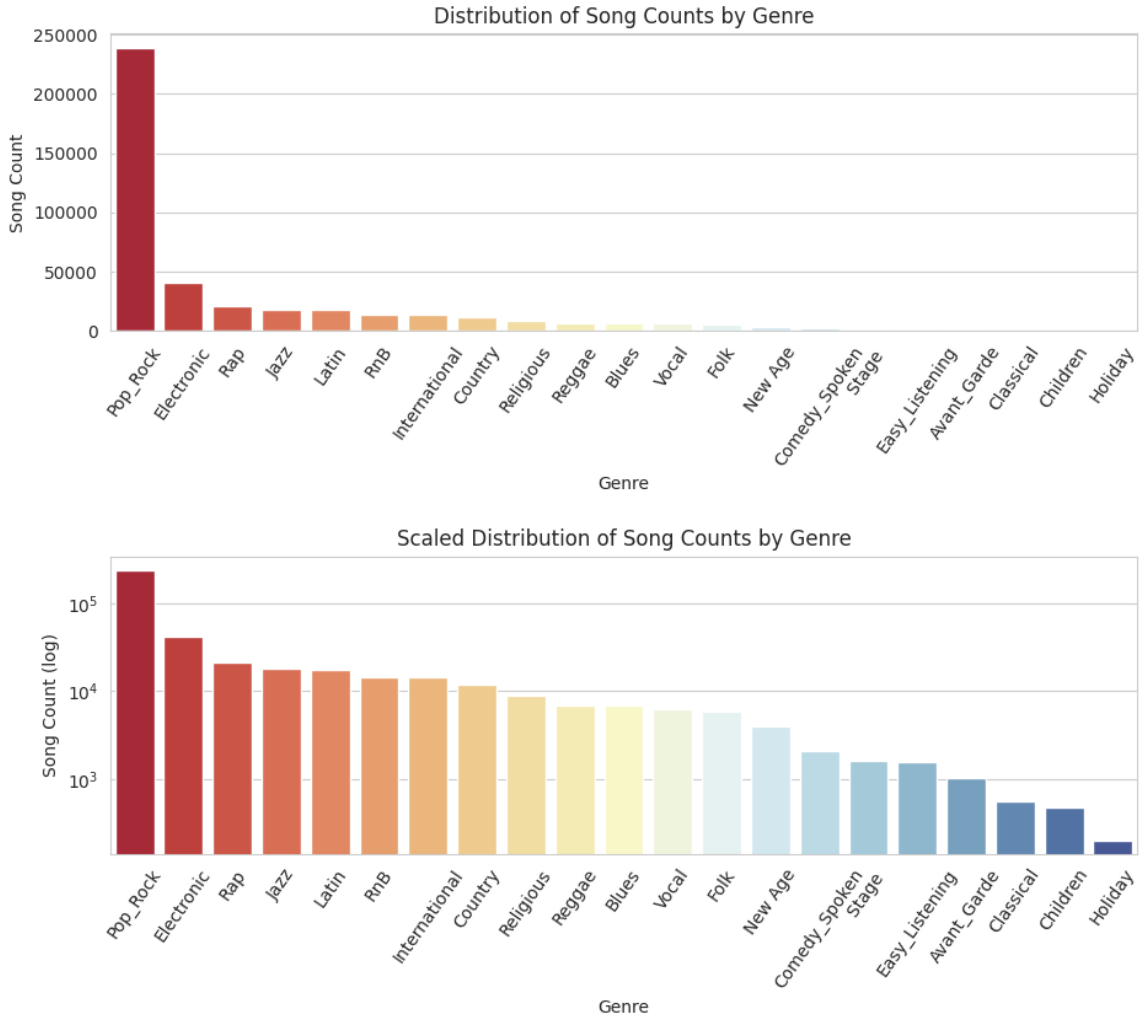

Figure 2 Distribution of Genres in MSD dataset by song count (plot title is incorrect)

The genres were also given an “index” number, that was essentially a rank number, with pop-rock music receiving a “1” being the genre with the most songs, and holiday music receiving a “21”, being the genre with the least songs. The genre categorization was combined with the observed dataset, Methods of Moment, to label a genre to each track, if available.

Binary Classification of Electronic Music

To isolate Electronic tracks, all of the songs that were categorized as Electronic by the genre dataset were labelled with a "1", and all other songs were labelled with a "0", independent from the genre index mentioned previously. Three classification models were applied to Methods of Moment, utilizing the binary categorization of 1 for known electronic music, and 0 for not known to be electronic. The Methods of Moment dataset contained a class balance of roughly 1:10 of electronic music to "not electronic music". Given that some of the track IDs did not receive an genre tag, it is possible that the class balance is different in reality, if some of the tracks that were not categorized with a genre were in fact electronic music.

Methods of Moment was split into two smaller datasets: training and test, both of which contained matching class balances to the dataset as a whole. Despite splitting the test and training data as close to the true class balance as possible, the training dataset contained a slightly higher percentage of the not electronic class, as seen in table 2, which may have affected the results.

Table 2 Ratio Distribution of electronic and not electronic music in master, training and test datasets

DATASET	ELECTRONIC (1)	NOT ELECTRONIC (0)
METHODS OF MOMENT	9.6681 %	90.3319%
TRAINING	9.6679%	90.3321%
TEST	9.6689%	90.3311%

The training dataset is used to teach the model what an electronic song "looks" like, based on the relationships between audio features of the tracks known to be electronic. The model uses those relationships to create, essentially, a criteria of audio features that electronic music "looks" like. The model is then applied to the test dataset, which is made up of tracks that were not present in the training dataset, to evaluate the new tracks and determine if it meets the electronic criteria (1), or does not (0). The 1 and 0 labels that the model predicts for the test dataset is compared against the known labels for test to evaluate the models ability to correctly and consistently identify electronic music. Three algorithms were used for the binary classification of electronic music. A summary of the results from all three binary classification models can be found in table 3.

Table 3 Performance metrics (precision, recall, accuracy, AUROC) for the three classification models trained

BINARY CLASSIFICATION MODEL	PRECISION	RECALL	ACCURACY	AUROC
LOGISTIC REGRESSION	0.4856	0.0290	0.9031	0.7606
RANDOM FOREST	0.0000	0.0000	0.9033	0.7429
GRADIENT-BOOSTED TREES	0.5812	0.1078	0.9062	0.7951

The first model was a logistic regression model. Logistic regression was chosen primarily due to its prominence. There is a lot of documentation and reference material for logistic regression and its application to classification models. A user with minimal statistical knowledge would find plenty of tutorials on how it can be applied and interpreted successfully. Logistic regression models also have versatility in the parameters specified in them, which can be applied to both binary and multiclass classifications. The binary logistic regression model for electronic music resulted in 7,898 false negatives, and 250 false positives. While the model was able to very accurately predict when electronic music was, in fact, electronic, it was not very successful at distinguishing what music was not electronic. Overall, the logistic regression model had an AUROC of 0.7606, accuracy of 0.9031, and a recall of 0.0290.

The second model was a random forest model. The random forest model was, similarly to logistic regression, chosen primarily due to the wide availability of documentation and tutorials relating to it. The random forest model was likely a poor choice of model, given that it is a very strong model for reducing overfitting, which was not exactly a concern for this binary classification. The random forest model resulted in an AUROC of 0.7429 which may appear to be fairly decent, being only slightly lower than the AUROC of the logistic regression, however the random forest model did not

correctly identify any electronic tracks; the model did not predict *any* tracks to be electronic at all, true positive or false positive. The random forest model had an accuracy of 0.9033, and recall of 0.0000.

Finally, a Gradient-Boosted Trees classification model was used. Gradient-Boosted Trees was selected as a classification model after consulting a *Towards Data* Science article. According to the Spark documentation on Gradient-Boosted Trees, the model iteratively trains sequences of decision trees, essentially weighting importance on models that preformed poorly. The gradient-boosted tree model was, generally, the most successful of the binary classification models. It produced 632 false positives and 7,257 false negatives. The AUROC of the Gradient-Boosted Trees was 0.7951, with an accuracy of 0.9062 and a recall of 0.1078. The Gradient-Boosted Trees mode, while with slightly better metrics than the previous two models, still did not perform very well at its basic application.

None of the first three models used any resampling methods on the training dataset to altar the class balance. According to James Williams™, when a class balance is 1:10, resampling is generally not necessary, so no up-sampling, down-sampling or weighting was performed on the training dataset. With the unaltered class balance, on all of the models trained, the accuracy metric was very similar to the class ratio of non-electronic music, as shown in table 4. On the best preforming model, the gradient-boosted trees model, the metric values of both accuracy and recall were extremely similar to the class balance.

Table 4 Comparison of the performance metrics for logistic regression (LR), random forest (RF), and Gradient-Boosted Trees (GBT) models and class balance ratios

	LR	RF	GBT
NON-ELECTRONIC (0) CLASS RATIO	0.9033	0.9033	0.9033
ACCURACY	0.9031	0.9033	0.9062
DIFFERENCE (RATIO-ACCURACY)	**0.0002**	**0.0000**	**-0.0029**
ELECTRONIC CLASS (1) RATIO	0.0967	0.0967	0.0967
RECALL	0.0290	0.0000	0.1078
DIFFERENCE (RATIO-RECALL)	**0.0677**	**0.0967**	**-0.0111**

In an effort to improve the metrics of the binary classification model, a second Gradient-Boosted Trees model was trained using cross-validation and changes to the hyperparameters of the model. The model was modified to increase how many 'splits', or depth of the decision trees within the model that were analyzed. Additionally, the parameter pertaining to the categorization of the audio features was increased, allowing for more 'bins' of audio characteristics. Finally, the number of decision trees inside of the model was increased. The model was trained with these new hyperparameters, and cross-validated, meaning the model was evaluated multiple times as it was trained and then averaged, which can more accurately capture the performance of the model. This second Gradient-Boosted Trees model increased the metrics overall, with an AUROC of 0.8098, accuracy of 0.9077, and recall of 0.1302. The cross-validated model produced 690 false positives and 7,075 false negatives. A comparison of metrics between the first Gradient-Boosted Trees model and the cross-validated Gradient-Boosted Trees with tuned hyperparameters can be seen in table 5.

*Table 5 Comparison of performance metrics for simple GBT model against GBT model that was cross-validated and *with tuned hyperparameters*

BINARY CLASSIFICATION MODEL	PRECISION	RECALL	ACCURACY	AUROC
GRADIENT-BOOSTED TREES	0.5812	0.1078	0.9062	0.7951
CROSS-VALIDATED GRADIENT-BOOSTED TREES*	0.6055	0.1302	0.9077	0.8098

Logistic Regression Multiclass Classification

Logistic Regression classification models can be used for both binary classification and multiclass classifications. When performing multiclass classifications, a logistic regression model uses the one-vs-all method, which essentially preforms a binary classification for each class, repeating the binary classification for each class, according to Sucky (2020). It builds a criteria for what class 1 'looks' like, checks all tracks to see if their attributes meet that criteria, and assign it as either "class 1" or "not class 1", then repeats for class 2, and so on.

Using the genre index added previously that labelled each genre from 1 to 21 based on its popularity, the dataset was split into training and test datasets again, this time with the class balances of all 20 genres taken into consideration. Due to the extreme differences between classes, primarily the extreme over-representation of class 1 (pop-rock) making up over 50% of the data, some resampling was performed to attempt to balance the classes. The resampling in this case was fairly extreme, with up-sampling of the under-represented classes to 5,000 and down-sampling of the over-represented class to 5,000. Table 6 shows the multiclass class balance of the Methods of Moment multiclass training dataset, the resampled training set, and the resulting class balance of the logistic regression model using the resampled training data. The exact cause for the logistic regression class balance being nearly identical to the class balance of the original training set is not exactly known or understood, and may have resulted from an error in the code.

Table 6 Class balance of the multiclass training data, the resampled multiclass training data, and the resulting logistic regression model (LR) using the resampled training data

CLASS (GENRE INDEX)	TRAINING	RESAMPLED TRAINING	LR WITH RESAMPLED TRAINING
POP-ROCK 1	0.5650	0.0468	0.5649
ELECTRONIC 2	0.0967	0.0478	0.0967
RAP 3	0.0497	0.0466	0.0497
JAZZ 4	0.0423	0.0477	0.0423
LATIN 5	0.0416	0.0470	0.0416
RNB 6	0.0340	0.0483	0.0340
INTERNATIONAL 7	0.0337	0.0473	0.0337
COUNTRY 8	0.0278	0.0478	0.0278
RELIGIOUS 9	0.0209	0.0480	0.0209
RAGGAE 10	0.0165	0.0478	0.0165
BLUES 11	0.0162	0.0476	0.0162
VOCAL 12	0.0147	0.0475	0.0147
FOLK 13	0.0138	0.0473	0.0138
NEW AGE 14	0.0095	0.0479	0.0095
COMEDY SPOKEN 15	0.0049	0.0474	0.0049
STAGE 16	0.0038	0.0465	0.0038
EASY LISTENING 17	0.0036	0.0475	0.0037
AVANT GARDE 18	0.0024	0.0476	0.0024
CLASSICAL 19	0.0013	0.0474	0.0013
CHILDREN 20	0.0011	0.0473	0.0011
HOLIDAY 21	0.0005	0.0474	0.0005

While this resampling is extreme, all trials with less extreme resampling methods resulted in the lower classes having 0.00 representation in the model metrics, as displayed in appendix B. It was deemed to be more appropriate to ensure all classes had *some* representation, even if it is not very well represented. The logistic regression model was trained using the resampled multiclass training dataset. The logistic regression model performed with an AUROC of 0.3354, and an accuracy of 0.2679, and precision, recall and F-1 score (that combines precision and recall) metrics are shown in table 7.

Table 7 Precision, Recall and F-1 Scores per class from the multiclass logistic regression model using the resampled training data

CLASS (GENRE INDEX)	PRECISION	RECALL	F-1
POP-ROCK 1	0.8861	0.3164	0.4663
ELECTRONIC 2	0.2923	0.2305	0.2577
RAP 3	0.2568	0.4555	0.3284
JAZZ 4	0.2038	0.2334	0.2176
LATIN 5	0.1099	0.0531	0.0716
RNB 6	0.1270	0.1834	0.1501
INTERNATIONAL 7	0.0539	0.0166	0.0253
COUNTRY 8	0.0882	0.1603	0.1138
RELIGIOUS 9	0.0475	0.1201	0.0681
RAGGAE 10	0.0713	0.2199	0.1077
BLUES 11	0.0531	0.1852	0.0826
VOCAL 12	0.0921	0.1067	0.0989
FOLK 13	0.0574	0.0674	0.0620
NEW AGE 14	0.0844	0.3521	0.1361
COMEDY SPOKEN 15	0.0866	0.6570	0.1530
STAGE 16	0.0411	0.3437	0.0733
EASY LISTENING 17	0.0230	0.1104	0.0380
AVANT GARDE 18	0.0103	0.2069	0.0197
CLASSICAL 19	0.0197	0.2768	0.0367
CHILDREN 20	0.0023	0.0430	0.0043
HOLIDAY 21	0.0015	0.1463	0.0030

The closer an F-1 score is to 1, the better the model is considered to have performed overall – it is successful at both identifying a class correctly, and at identifying all the class correctly. As seen in the metrics of the multiclass logistic regression model, the highest F-1 score achieved was, unsurprisingly, the pop-rock genre. A handful of lower classes, such as Country (8), New Age (14) and Comedy (15) had F-1 scores higher than the surrounding lower classes, due to higher recall metrics, which may be an indication that the audio features of those genres are distinct enough to consistently be correctly categorized, though with extra false positives. Overall, the metrics are not very high, indicating a poor model performance, however with multiclass classification this is not entirely unusual.

Song Recommendations

Taste Profile

The taste profile dataset contains three columns: user ID, song ID, and plays. This results in a very long dataset, as each user has as many rows as different songs they have played. In total, the taste profile dataset contains 384,543 different songs, and the interactions between those songs and 1,019,318 different users. Compiling each user ID with how many different songs they have interacted with can identify the user who has listened to the widest variety of songs within the dataset, with that user being "ec6dfcf19485cb011e0b22637075037aae34cf26" having listened to 4,400 or 1.1442% of the variety of songs within the data.

There was a wide range of values pertaining to both how many times a particular song was played in total across all users, as well as how many total plays a user played across all songs. The vast majority of both cases was the lower end of plays, generally under 50-100 in both scenarios, however the extremes were quite extreme, resulting in, similarly to the distribution difference between pop-rock and every other genre, an extreme gap between the highest plays and lowest. To make the distribution of plays for users and songs more readable, both were normalized on a log base 2 scale to create figure 3. The top plot shows the distribution of total play counts per user to highlight the average play count per user, although with a long trailing tail showing that the distribution is skewed to the right. The bottom plot shows the distribution of total play counts per song, and has a similar distribution and skew as the user activity, though with two hiccups of a high number of songs having *very* low play counts before the distribution is more normalized.

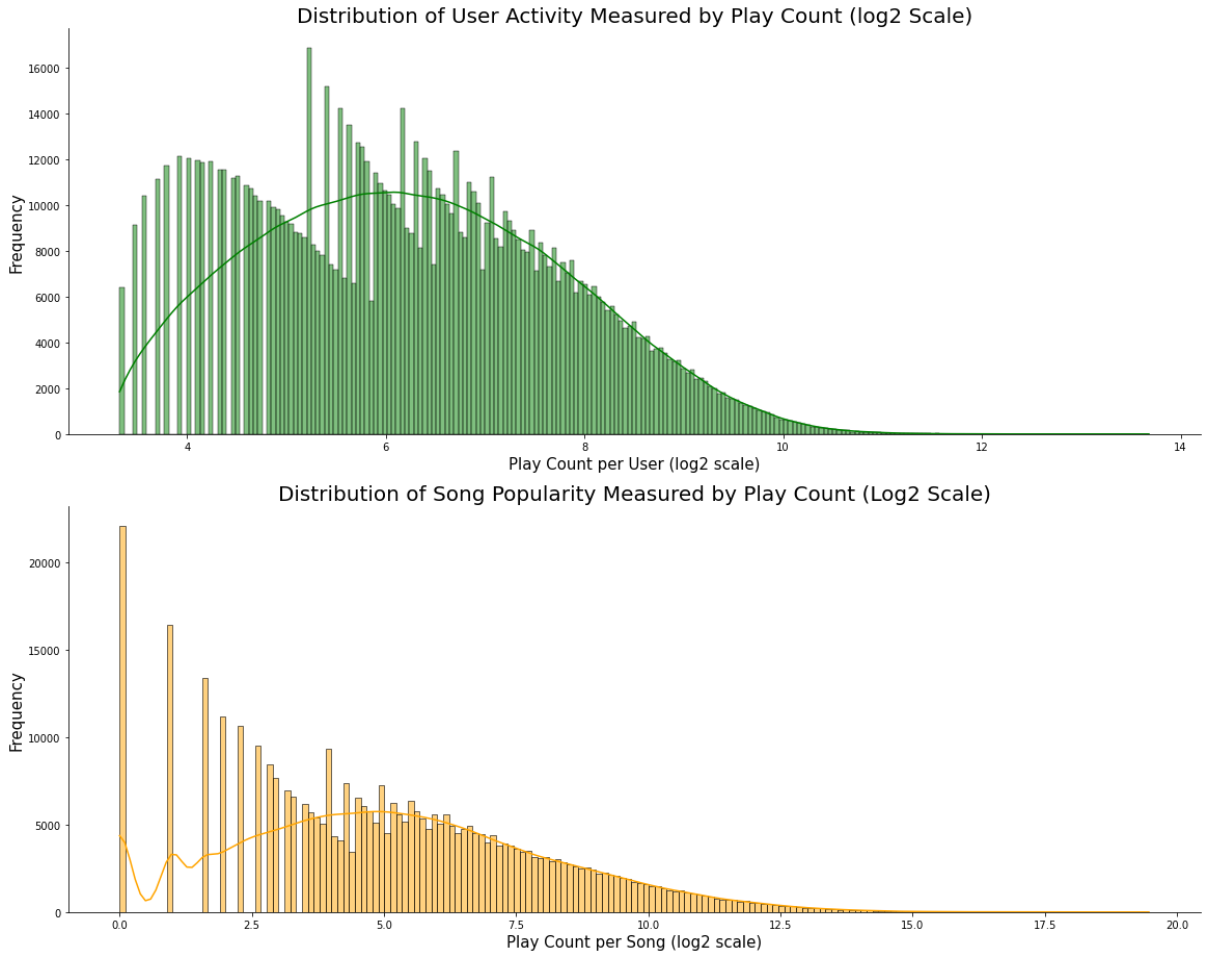

Figure 3 Distribution of user activity (top) and song popularity (bottom) as measured by play counts, normalized on a log2 scale

Collaborative Filtering

Collaborative filtering functions as, essentially, a matrix of users as columns, and songs as rows. Not every user has listened to every song – even the user who has listened to the widest variety of songs only interacted with 1.144% of all of the songs. To generate recommendations, collaborative filtering pairs users who have listened to similar songs together – the more overlap the better. For example, if User A and User D have both listened to songs 9, 13 and 17, but user D has also listened to song 3, collaborative filtering would recommend song 3 to User A, given that the two users seem to have similar tastes and preferences in songs. In addition to its normal algorithms and collaborative filtering, Spotify currently does a very upfront explicit model of this, with the "blend" playlists, that allow two users to "blend" their music together into a playlist that contains songs that both users like, and songs that each user likes that the other one either doesn't, or hasn't encountered or interacted with previously.

Observing the general distribution of both plots, a new dataset was created using a threshold of 50 minimum plays, for both users and songs. 50 plays lands just below the median number of plays per song, allowing for the majority of songs to be included in the dataset, and excluding songs with minimal plays which would not contribute much to the filtering model. 50 plays includes more than half of users, but lands around where the distribution starts to thin out, and with collaborative filtering, the users to compare against each other, the better.

The new dataset, the "threshold data", was split using a random split of 75% to training and 25% to test, and then the test data was matched to the training data to ensure that each user in test was represented at least once in the training data. The collaborative filtering model cannot make recommendations to users that it has never encountered before, which is why it is important to ensure all users are represented at least once. The test dataset was verified to contain at least 25% of the total plays in threshold data, as seen in figure 4.

```
# training rows = 21501010
# unmatched test rows      = 7167309
# matched test rows      = 7167309
test lost 0 rows in the matching join
test has 25.0% of all plays in threshold_data
```

Figure 4 Verification of 25% of plays represented in the test dataset

It is worth noting that, if a time component of the data was available, the train/test split would be created using a time marker, with training data being data acquired before the split date, and test data being the data acquired after the split date. This is done to assess the models ability to predict future events based on patterns and behavior, which is often needed in recommendation algorithms.

Recommendations

Using the collaborative filtering model Alternating Least Squares, trained by the training data that met the threshold criteria, a set of 10 recommendations per user for the top 15 users by play count was created. The recommendations were given in a relative order of assumed user preference; that is the order in which the items should be recommended, not of preference of user probability. The recommendations were compared against known user interactions to evaluate the quality of the recommendations.

Using the basic ALS model and threshold of 50 plays for users and songs, nine of the top 15 users had any recommendations overlap with a song known to be relevant to them (Appendix C). One user had 3 recommendations overlap, two users had two recommendations overlap, and the remaining six users had only one overlapping recommendation. When the threshold for songs was lowered to a minimum of 10 plays (Appendix D), eight of the top 15 users were given a recommendation that overlapped their known relevant songs, all of which was only one overlapping recommendation.

When the hyperparameters of the model were tuned (Appendix E), increasing the number of iterations of the model from 5 to 10, and the regularization from 0.01 to 0.05, eight of the top 15 users had overlapping recommendations. In the hyperparameter model, two users had three recommendations overlap with their known relevant songs, and in

the case of user (index) 4, one of the three recommendations was the user's 4th most played song. Two users had two recommended songs overlap, and the remaining four had one recommendation overlap, and for one of those users, user (index) 2, the recommended song was their 5th most played song.

The quality of a recommendation model can be quantified using several metrics, such as Precision @ K, Mean Average Precision (MAP) @ K, and Normalized, Discounted Cumulative Gain (NDCG) @ K. Precision @ K checks the top K, in this case 10, songs recommended and compares them against all of the songs the user is known to have interacted with. One "point" is given to each correct (as in, exists in the list of knowns) song, regardless of how far down the list of known interactions (which is in order of song plays) the recommendation appears. Precision @ K also cannot account for how far down a recommended song is, if it should have been recommended higher.

MAP @ K functions very similarly, but does take into consideration the position of where the recommended song appears in the list of known interactions. If song A is the 5th recommended song and is on the user's list of known interactions, it does not receive as high of a "point" as it would have with Precision @ K, where a "point" is won for any recommendation existing on the user's interaction list, regardless of how highly it was recommended. In MAP @ K, the "point" received by any recommendation is a function of its recommendation location, where the first several recommendations are "worth" more, and lower recommendations are "worth" less.

NDCG @ K takes into account not only the position a recommendation, but also the position that recommendation holds on the list of known interactions, essentially allowing for "penalty" for poor recommendations, and weighting correct recommendations. There is a lot of math behind the NDGC @ K metric, but it mostly boils down to the first several recommendations being "worth" a lot, and the value of recommendations after that point is relatively insignificant, regardless of if that recommendation was in fact on the user's known interactions list.

Table 8 shows the metrics for each collaborative filtering model trialed. Reducing the threshold for song plays down to 10 plays had a negative impact on the model's performance, confirming that songs with low play counts do not contribute to the model. Alteration of the hyperparameters had the greatest impact on performance metrics overall, particularly with increasing iterations through the model.

Table 8 Performance metrics of four collaborative filtering models for song recommendations.

MODEL	PRECISION @ 10	MAP @ 10	NDCG @ 10
BASE ALS	0.06667	0.04744	0.08588
REDUCED THRESHOLD ALS	0.5714	0.01619	0.05743
HYPERPARAMETERS* ALS	0.9333	0.04034	0.10127
HYPERPARAMETERS ALS**	0.10667	0.05563	0.12876

**Iterations increased from 5 to 10 and regularization increased from 0.01 to 0.05*
***Iterations increased from 5 to 15 and regularization increased from 0.01 to 0.05*

Models and metrics can only go so far in an offline and implicit environment. An offline model can be repeatedly trained until it is perfect, but still perform poorly when it is released, because things change, and people change. A recommendation model is only as good as the age of the data it is trained on; the taste profile dataset was released over 10 years ago, and any models trained on it would almost certainly do a very poor job at generating accurate predictions for the same users. If a model is online and live, the users actual interaction with recommendations would be the best measure of a models performance in an implicit environment. The interactions can be used to re-train the model to make new predictions and recommendations and grow with the additional feedback.

References (Report)

Sucky, R. N. (2020, November 3). Multiclass classification algorithm from scratch with a project in Python: Step by step guide. Towards Data Science. https://towardsdatascience.com/multiclass-classification-algorithm-from-scratch-with-a-project-in-python-step-by-step-guide-485a83c79992

References (Code)

Databricks. (n.d.). Getting Started with Spark MLlib - Databricks Notebook Guide - Databricks. Databricks. https://www.databricks.com/notebooks/gallery/GettingStartedWithSparkMLlib.html

Stack Overflow. (2014, November 19). Spark iterate HDFS directory. https://stackoverflow.com/questions/27023766/spark-iterate-hdfs-directory

Flateman, J. (2018, May 28). Building a recommender system in PySpark using ALS. Medium. https://medium.com/@jonahflateman/building-a-recommender-system-in-pyspark-using-als-18e1dd9e38e6

Stack Overflow. (2018, August 17). List out only the file name from folder using spark [Duplicate]. https://stackoverflow.com/questions/51939514/list-out-only-the-file-name-from-folder-using-spark

Saini, A. (2018, August 20). Machine Learning with PySpark and MLlib: Solving a Binary Classification Problem. Towards Data Science. https://towardsdatascience.com/machine-learning-with-pyspark-and-mllib-solving-a-binary-classification-problem-96396065d2aa

Patel, N. (2019, February 12). Recommendation system in python using ALS algorithm and Apache Spark. Medium. https://medium.com/@patelneha1495/recommendation-system-in-python-using-als-algorithm-and-apache-spark-27aca08eaab3

A whole lot of ChatGPT and your sample code

- Plotting help
- Hyperparameters and what they mean/do
- Very failed cross-validation of multiclass
- F-1 Score explanation

Appendix

Appendix A

Summary of Methods of Moments/Descriptive Statistics

	COUNT	MEAN	STDDEV	MIN	25%	50%	75%	MAX
ST_DEV_1	994623	0.154982	0.066462	0	0.1058	0.1524	0.1991	0.959
ST_DEV_2	994623	10.38455	3.868001	0	7.704	9.955	12.66	55.42
ST_DEV_3	994623	526.814	180.4378	0	403.5	515.1	632.8	2919
ST_DEV_4	994623	35071.98	12806.82	0	26170	34200	42490	407100
ST_DEV_5	994623	5297870	2089356	0	3845000	5042000	6383000	4.66E+07
AVG_1	994623	0.350844	0.18558	0	0.2081	0.3293	0.4727	2.647
AVG_2	994623	27.46387	8.352649	0	21.71	27.92	33.28	117
AVG_3	994623	1495.809	505.8938	0	1151	1549	1872	5834
AVG_4	994623	143165.5	50494.28	-146300	101800	159300	184300	452500
AVG_5	994623	2.40E+07	9307340	0	1.66E+07	2.74E+07	3.12E+07	9.48E+07

Appendix B

Example of 0.00 metrics in multiclass classification using “less extreme” resampling of training data:

```
Random Forest AUROC: 0.46864890616379357
Accuracy: 0.5110118021797783
Precision (label 1.0): 0.687288443518005
Precision (label 2.0): 0.22211549327607638
Precision (label 3.0): 0.3059456928838951
Precision (label 4.0): 0.19325445396436827
Precision (label 5.0): 0.04
Precision (label 6.0): 0.1406443618339529
Precision (label 7.0): 0.2
Precision (label 8.0): 0.019417475728155338
Precision (label 9.0): 0.0
Precision (label 10.0): 0.0
Precision (label 11.0): 0.2
Precision (label 12.0): 0.16
Precision (label 13.0): 0.0
Precision (label 14.0): 0.15436241610738255
Precision (label 15.0): 0.2793733681462141
Precision (label 16.0): 0.0
Precision (label 17.0): 0.0
Precision (label 18.0): 0.0
Precision (label 19.0): 0.0
Precision (label 20.0): 0.0
Precision (label 21.0): 0.0
Recall (label 1.0): 0.756132968651378
Recall (label 2.0): 0.4832800590115564
Recall (label 3.0): 0.31267942583732056
Recall (label 4.0): 0.39960629921259844
Recall (label 5.0): 0.00028563267637817766
Recall (label 6.0): 0.07928746070555362
Recall (label 7.0): 0.0003522367030644593
Recall (label 8.0): 0.0008550662676357417
Recall (label 9.0): 0.0
Recall (label 10.0): 0.0
Recall (label 11.0): 0.0029390154298310064
Recall (label 12.0): 0.025869037995149554
Recall (label 13.0): 0.0
Recall (label 14.0): 0.02871410736579276
Recall (label 15.0): 0.2584541062801932
Recall (label 16.0): 0.0
Recall (label 17.0): 0.0
Recall (label 18.0): 0.0
Recall (label 19.0): 0.0
Recall (label 20.0): 0.0
Recall (label 21.0): 0.0
F1-Score (label 1.0): 0.7200689233728375
F1-Score (label 2.0): 0.3043511923196036
F1-Score (label 3.0): 0.3092759110269758
F1-Score (label 4.0): 0.2605188376569804
F1-Score (label 5.0): 0.0005672149744753261
F1-Score (label 6.0): 0.10140719231628323
F1-Score (label 7.0): 0.0007032348804500703
F1-Score (label 8.0): 0.0016380016380016380
F1-Score (label 9.0): 0.0
F1-Score (label 10.0): 0.0
F1-Score (label 11.0): 0.005792903692976104
F1-Score (label 12.0): 0.04453723034098817
F1-Score (label 13.0): 0.0
F1-Score (label 14.0): 0.04842105263157895
F1-Score (label 15.0): 0.2685069008782936
F1-Score (label 16.0): 0.0
F1-Score (label 17.0): 0.0
F1-Score (label 18.0): 0.0
F1-Score (label 19.0): 0.0
F1-Score (label 20.0): 0.0
F1-Score (label 21.0): 0.0
```

Appendix C

Basic ALS Recommendations Overlap Table:

user_id_index	recommendations	relevant	overlap
7	[1.0, 0.0, 14.0, 8.0, 16.0, 24.0, 7.0, 27.0, 1...	[804.0, 3358.0, 2052.0, 3102.0, 3431.0, 2058.0...	[]
4	[24.0, 28.0, 12.0, 0.0, 15.0, 62.0, 139.0, 8.0...	[284.0, 415.0, 6739.0, 10389.0, 6494.0, 1168.0...	[8.0]
9	[1.0, 14.0, 48.0, 47.0, 68.0, 33.0, 3.0, 97.0,...	[4418.0, 5027.0, 121.0, 19.0, 2889.0, 484.0, 6...	[47.0, 11.0]
3	[14.0, 24.0, 201.0, 28.0, 11.0, 97.0, 41.0, 62...	[31002.0, 4219.0, 2215.0, 9020.0, 338.0, 20915...	[14.0]
11	[16.0, 14.0, 42.0, 33.0, 47.0, 201.0, 217.0, 3...	[16314.0, 6171.0, 50.0, 9377.0, 333.0, 10739.0...	[]
12	[213.0, 42.0, 318.0, 70.0, 286.0, 402.0, 46.0,...	[13853.0, 10579.0, 5486.0, 7442.0, 23975.0, 85...	[]
1	[47.0, 48.0, 1.0, 14.0, 33.0, 97.0, 134.0, 136...	[7425.0, 3740.0, 18317.0, 4199.0, 8682.0, 112....	[11.0]
14	[41.0, 14.0, 28.0, 24.0, 9.0, 32.0, 76.0, 62.0...	[6270.0, 6368.0, 5961.0, 19060.0, 63835.0, 524...	[]
0	[14.0, 201.0, 97.0, 11.0, 163.0, 136.0, 297.0,...	[77699.0, 75994.0, 73323.0, 67826.0, 62652.0, ...	[]
2	[1.0, 48.0, 136.0, 97.0, 47.0, 68.0, 245.0, 35...	[2639.0, 11420.0, 1462.0, 164.0, 32959.0, 836....	[97.0]
6	[16.0, 15.0, 12.0, 10.0, 67.0, 205.0, 39.0, 40...	[5.0, 1136.0, 24188.0, 216.0, 1085.0, 1278.0, ...	[16.0, 15.0]
13	[1.0, 14.0, 97.0, 201.0, 11.0, 16.0, 217.0, 29...	[1816.0, 21.0, 248.0, 199.0, 901.0, 269.0, 551...	[11.0]
10	[24.0, 62.0, 28.0, 122.0, 139.0, 108.0, 105.0,...	[35761.0, 4095.0, 17822.0, 70216.0, 17429.0, 1...	[]
5	[14.0, 28.0, 42.0, 33.0, 1.0, 16.0, 0.0, 40.0,...	[1719.0, 13835.0, 1080.0, 1072.0, 27630.0, 191...	[0.0]
8	[47.0, 42.0, 213.0, 33.0, 402.0, 134.0, 335.0,...	[17865.0, 13356.0, 155.0, 97.0, 5.0, 23358.0, ...	[33.0, 402.0, 66.0]

Appendix B

Song threshold of 10 ALS Recommendations Overlap Table:

user_id_index	recommendations	relevant	overlap
7	[0.0, 16.0, 23.0, 1.0, 46.0]	[3687.0, 2271.0, 22731.0, 4487.0, 2938.0, 1225...	[]
4	[16.0, 66.0, 39.0, 23.0, 42.0]	[4345.0, 3015.0, 1348.0, 643.0, 250.0, 4822.0,...	[42.0]
9	[0.0, 67.0, 46.0, 33.0, 14.0]	[37909.0, 86011.0, 4345.0, 34489.0, 11640.0, 6...	[14.0]
3	[23.0, 22.0, 61.0, 1.0, 24.0]	[22876.0, 146228.0, 25167.0, 8995.0, 16203.0, ...	[]
11	[165.0, 317.0, 16.0, 177.0, 210.0]	[10048.0, 3257.0, 5317.0, 100277.0, 86420.0, 3...	[]
12	[16.0, 66.0, 42.0, 137.0, 319.0]	[27151.0, 21.0, 5894.0, 17029.0, 5606.0, 626.0...	[]
1	[47.0, 46.0, 0.0, 14.0, 33.0]	[7177.0, 17102.0, 1920.0, 1377.0, 12918.0, 292...	[14.0]
0	[51.0, 67.0, 138.0, 87.0, 33.0]	[178270.0, 177981.0, 135714.0, 132419.0, 11712...	[138.0]
2	[46.0, 47.0, 100.0, 0.0, 95.0]	[22221.0, 2860.0, 3429.0, 16779.0, 15726.0, 30...	[0.0]
6	[199.0, 202.0, 205.0, 14.0, 60.0]	[118862.0, 2555.0, 3864.0, 13496.0, 153.0, 38....	[205.0]
13	[199.0, 16.0, 14.0, 95.0, 46.0]	[49.0, 298.0, 28992.0, 1395.0, 283.0, 262.0, 1...	[]
10	[33.0, 16.0, 42.0, 43.0, 217.0]	[60791.0, 71871.0, 67125.0, 119806.0, 41378.0,...	[]
5	[42.0, 47.0, 215.0, 402.0, 110.0]	[26071.0, 222.0, 67.0, 1177.0, 464.0, 391.0, 1...	[47.0]
8	[29.0, 47.0, 43.0, 16.0, 33.0]	[8161.0, 6938.0, 111.0, 3087.0, 15.0, 6129.0, ...	[33.0]

Appendix C

Hyperparameters* ALS Recommendations Overlap Table:

user_id_index	recommendations	relevant	overlap
7	[214.0, 401.0, 379.0, 317.0, 42.0, 210.0, 136....	[44.0, 20.0, 35234.0, 8984.0, 3723.0, 5085.0, ...	[42.0]
4	[16.0, 209.0, 66.0, 317.0, 38.0, 15.0, 162.0, ...	[2609.0, 44.0, 20522.0, 38.0, 20.0, 13.0, 3552...	[66.0, 38.0, 30.0]
9	[0.0, 48.0, 47.0, 33.0, 14.0, 349.0, 64.0, 99....	[2041.0, 800.0, 22091.0, 13913.0, 1970.0, 3726...	[]
3	[203.0, 76.0, 201.0, 45.0, 233.0, 245.0, 253.0...	[24449.0, 9345.0, 20125.0, 27120.0, 133.0, 867...	[76.0]
11	[214.0, 284.0, 321.0, 69.0, 109.0, 401.0, 2330...	[8657.0, 7456.0, 328.0, 3625.0, 10980.0, 13056...	[]
12	[16.0, 76.0, 45.0, 33.0, 203.0, 42.0, 214.0, 1...	[4.0, 234.0, 10.0, 5.0, 44.0, 156.0, 86.0, 109...	[16.0, 203.0]
1	[14.0, 47.0, 48.0, 0.0, 202.0, 33.0, 95.0, 340...	[3351.0, 2018.0, 6234.0, 6258.0, 73742.0, 6968...	[215.0]
14	[202.0, 14.0, 16.0, 203.0, 201.0, 72.0, 95.0, ...	[23281.0, 20452.0, 1608.0, 23305.0, 25629.0, 2...	[]
0	[68.0, 348.0, 135.0, 153.0, 162.0, 261.0, 251....	[72850.0, 72548.0, 70022.0, 66559.0, 60486.0, ...	[]
2	[14.0, 0.0, 48.0, 202.0, 47.0, 95.0, 349.0, 29...	[832.0, 10027.0, 1141.0, 288.0, 407.0, 5106.0,...	[407.0]
6	[1.0, 16.0, 0.0, 22.0, 23.0, 24.0, 8.0, 27.0, ...	[1015.0, 14.0, 7236.0, 26695.0, 2816.0, 2920.0...	[1.0, 24.0]
13	[14.0, 16.0, 0.0, 47.0, 33.0, 42.0, 217.0, 202...	[13230.0, 2474.0, 23037.0, 47477.0, 34170.0, 1...	[]
10	[3.0, 123.0, 207.0, 1.0, 184.0, 339.0, 8.0, 19...	[36371.0, 32108.0, 68098.0, 32895.0, 15237.0, ...	[]
5	[1.0, 23.0, 22.0, 24.0, 12.0, 55.0, 59.0, 30.0...	[26781.0, 38271.0, 223.0, 5391.0, 4786.0, 4.0,...	[23.0, 12.0, 59.0]
8	[45.0, 14.0, 0.0, 1.0, 3.0, 11.0, 16.0, 76.0, ...	[7787.0, 11357.0, 1156.0, 3796.0, 12548.0, 680...	[]

Appendix D

Hyperparameters** Recommendations Overlap Table:

user_id_index	recommendations	relevant	overlap
7	[212.0, 393.0, 324.0, 136.0, 42.0, 311.0, 293....	[2683.0, 2.0, 18998.0, 16.0, 3.0, 44372.0, 402...	[212.0]
4	[24.0, 1.0, 22.0, 23.0, 12.0, 15.0, 29.0, 62.0...	[577.0, 5019.0, 1395.0, 1161.0, 22071.0, 1084....	[24.0, 23.0]
9	[14.0, 0.0, 3.0, 45.0, 5.0, 33.0, 10.0, 67.0, ...	[27207.0, 35332.0, 5209.0, 11315.0, 908.0, 63....	[]
3	[14.0, 203.0, 45.0, 98.0, 0.0, 353.0, 315.0, 1...	[3356.0, 14240.0, 20550.0, 12934.0, 51045.0, 4...	[]
11	[122.0, 3.0, 206.0, 185.0, 1.0, 342.0, 221.0, ...	[73640.0, 38562.0, 13886.0, 59866.0, 23884.0, ...	[]
12	[212.0, 311.0, 393.0, 293.0, 2526.0, 68.0, 111...	[2003.0, 2784.0, 11538.0, 6075.0, 16167.0, 776...	[68.0]
1	[14.0, 45.0, 203.0, 0.0, 98.0, 48.0, 100.0, 33...	[1450.0, 7617.0, 14399.0, 2045.0, 1067.0, 5590...	[14.0, 45.0]
14	[52.0, 86.0, 191.0, 67.0, 138.0, 229.0, 227.0,...	[1639.0, 10030.0, 9072.0, 5191.0, 1387.0, 5592...	[67.0]
0	[138.0, 347.0, 67.0, 151.0, 212.0, 14.0, 248.0...	[65570.0, 60719.0, 50527.0, 45821.0, 41309.0, ...	[]
2	[16.0, 66.0, 324.0, 210.0, 136.0, 29.0, 24.0, ...	[8128.0, 3318.0, 1608.0, 19.0, 3313.0, 643.0, ...	[16.0]
6	[1.0, 0.0, 16.0, 24.0, 15.0, 14.0, 12.0, 8.0, ...	[1314.0, 14942.0, 7743.0, 2864.0, 3118.0, 350....	[15.0, 8.0]
13	[9.0, 46.0, 32.0, 187.0, 2526.0, 212.0, 68.0, ...	[320.0, 29705.0, 13640.0, 27.0, 516.0, 20120.0...	[32.0, 68.0]
10	[16.0, 33.0, 42.0, 204.0, 212.0, 75.0, 143.0, ...	[1875.0, 1039.0, 247.0, 549.0, 3629.0, 52.0, 5...	[]
5	[204.0, 202.0, 1.0, 75.0, 230.0, 250.0, 245.0,...	[33723.0, 860.0, 60783.0, 9213.0, 7213.0, 292....	[75.0, 250.0, 203.0, 16.0]
8	[46.0, 0.0, 14.0, 1.0, 3.0, 10.0, 2.0, 9.0, 18...	[3790.0, 3203.0, 1946.0, 934.0, 29239.0, 38662...	[]

*I ran this pretty last minute out of curiosity and did not actually reference it directly.