The Million Song Dataset (MSD) Analysis

Data processing

In the processing stage, we get a better understanding of the data structure and remove the songs that are not matched from the dataset for future use.

Data structure

The data is structured like the tree directory as shown below

```
 |---audio
 |-----attributes
 |-------msd-jmir-area-of-moments-all-v1.0.attributes.csv
 |-------msd-jmir-lpc-all-v1.0.attributes.csv
 |-------msd-jmir-methods-of-moments-all-v1.0.attributes.csv
 |-------msd-jmir-mfcc-all-v1.0.attributes.csv
 |-------msd-jmir-spectral-all-all-v1.0.attributes.csv
 |-------msd-jmir-spectral-derivatives-all-all-v1.0.attributes.csv
 |-------msd-marsyas-timbral-v1.0.attributes.csv
 |-------msd-mvd-v1.0.attributes.csv
 |-------msd-rh-v1.0.attributes.csv
 |-------msd-rp-v1.0.attributes.csv
 |-------msd-ssd-v1.0.attributes.csv
 |-------msd-trh-v1.0.attributes.csv
 |-------msd-tssd-v1.0.attributes.csv
 |-----features
 |-------msd-jmir-area-of-moments-all-v1.0.csv
 |---------part-00000.csv.gz
…
 |---------part-00007.csv.gz
 |-------msd-jmir-lpc-all-v1.0.csv
 |---------part-00000.csv.gz
 …
 |---------part-00006.csv.gz
 |---------part-00007.csv.gz
 |-------msd-jmir-methods-of-moments-all-v1.0.csv
 |---------part-00000.csv.gz
…
 |---------part-00007.csv.gz
 |-------msd-jmir-mfcc-all-v1.0.csv
 |---------part-00000.csv.gz
…
 |---------part-00007.csv.gz
 |-------msd-jmir-spectral-all-all-v1.0.csv
 |---------part-00000.csv.gz
 …
 |---------part-00007.csv.gz
 |-------msd-jmir-spectral-derivatives-all-all-v1.0.csv
 |---------part-00000.csv.gz
…
 |---------part-00007.csv.gz
 |-------msd-marsyas-timbral-v1.0.csv
 |---------part-00000.csv.gz
…
 |---------part-00007.csv.gz
 |-------msd-mvd-v1.0.csv
 |---------part-00000.csv.gz
 …
 |---------part-00007.csv.gz
 |-------msd-rh-v1.0.csv
 |---------part-00000.csv.gz
…
 |---------part-00007.csv.gz
 |-------msd-rp-v1.0.csv
 |---------part-00000.csv.gz
…
 |---------part-00006.csv.gz
 |---------part-00007.csv.gz
 |-------msd-ssd-v1.0.csv
 |---------part-00000.csv.gz
 …
 |---------part-00007.csv.gz
 |-------msd-trh-v1.0.csv
 |---------part-00000.csv.gz
…
 |---------part-00007.csv.gz
 |-------msd-tssd-v1.0.csv
 |---------part-00000.csv.gz
 …
 |---------part-00007.csv.gz
 |-----statistics
 |-------sample_properties.csv.gz
 |---genre
 |-----msd-MAGD-genreAssignment.tsv
 |-----msd-MASD-styleAssignment.tsv
 |-----msd-topMAGD-genreAssignment.tsv
 |---main
 |-----summary
 |-------analysis.csv.gz
 |-------metadata.csv.gz
 |---tasteprofile
 |-----mismatches
 |-------sid_matches_manually_accepted.txt
 |-------sid_mismatches.txt
 |-----triplets.tsv
 |-------part-00000.tsv.gz
 …
 |-------part-00007.tsv.gz
```

As can be seen, four subgroups are contained in this dataset, of which the total size is 12.9G. The table below gives the information of the subgroups, size and format of each subgroup.

Supergroups	subgroups	Size	Format
audio	Attributes, features, statistics	12.3G	Csv.gz
genre	genreAssignment, styleAssignment	30.1M	Tsv
main	summary	174.4M	Csv.gz
tasteprofile	Mismatches, triplets	490.4M	Txt, tsv.gz

Parallelism
When one single file is loaded into spark, it creates one partition. When serval files are loaded and input files that are gzip compressed, as they cannot be splitted, the number of partitions is determined by the number of files and the file size. However, repartition() function can increase the level of parallelism by using a shuffle to redistribute data(Pyspark Package). This is a useful method but might cost a lot because of latency.

Counts of rows

Dataset	Count of rows
the metadata for the main million song dataset(summary in main)	100,000
user-song play counts (tasteprofile)	48373586
msd-topMAGD-genreAssignment (Genre)	406427
Audio features	994188
Additional track statistics (Audio statistics)	992866

Data processing

To filter the Taste Profile dataset to remove the songs which were mismatched, we first load the mismatches that were identified and matches that were manually into spark, extract the SONG_IDs and exclude the matches that were manually accepted from those that were identified. We then do another left-anti join to exclude all the mismatches from the play counts. This gives us a data frame with 45,795,111 rows.

The audio feature attribute names and types from the audio/attributes directory are STRING, string, NUMERIC and real.

Audio similarity

In this section, we predict a song’s genres by its audio features with four different classifiers and compare the performances.

Pick one small dataset and produce statistics

The dataset we are using in this assignment is ‘msd-jmir-method-of-moments-all-v1.0, we load it into pyspark with the scheme we created in processing stage, as it has the smallest size and would not take too long to run.

Assemble features

We assume those features with correlation coefficients bigger than 0.8 are strongly correlated. We can get 13 combinations and they are (1,2), (2,3), (3,4), (6,7), (8,9), this means we will later take this into consideration if we are implementing a method that does not tolerance linearity.

To visualize the distribution of genres for the songs that were matched, we load in the genre dataset first, remove the mismatches and group the genres by their counts. We can then plot a pie chart. The distribution of the genres is as follows.

Merge datasets

Since we have removed the mismatches from the genres dataset, we can now join the genres dataset and audio features dataset to make sure every song has a label unless there are null variables in the datasets.

Binary classification model selection

In order to perform a prediction, we will be using logistic regression, random forest, random forest and Gradient-boosted tree. Out of interest, we use one more method than required in this question. Before we fit the models, we compare the classifiers in terms of explainability, interpretability, predictive accuracy, training speed, hyperparameter tuning, dimensionality, and issues with scaling.

Logistic regression is one of the most popular algorithms used for classification problems. It is easy to interpret compared to some other classifiers. When it comes to hyperparameter tuning, we use parameter C as our regularization parameter. Parameter C = 1/λ. (joparga3) Lambda controls the model complexity so it can be used to find a model that is either overfitted nor underfitted. Logistic regression is not collinearity tolerant, we will need dimensionality reduction to remove collinearity. Scaling is not necessary for logistic regression method, but it does not hurt to do so, sometimes it might improve the model.
Binary logistic regression can be generalized into multinomial logistic regression to train and predict multiclass classification problems(Linear Methods - RDD-Based API).
Random forest, decision tree and gradient-boosted tree are all tree-based methods, which means it takes longer to train them and they are much more complex to interpret especially when we have

more branches. From what we have learnt from the last semester, tree-based methods can easily handle categorical predictors without having to create dummy variables and can deal with missing values and perform feature selection and complexity reduction automatically. Generally, trees have poor predictive accuracy when compared with other methods. However, by aggregating trees, their predictive performance can be greatly improved. That's why random forest is chosen, it is an ensembled method based on trees and attempt to de-correlate trees. Hyperparameters to be tuned in a tree-based method are the maximum depth of the trees and maximum number of features considered at each split. When performing random forest, we also need to take the number of decision trees to be combined and whether bagging/bootstrapping is performed with or without replacement into consideration(Elsinghorst, Shirin).
While random forest is based of fully grown trees, gradient-boosted is based on weak leaners, it is thus less prone to overfit the data and in addition to the hyperparameters in a random forest, there are even more hypermeters to be tuned such as the number of features, the number of the minimum samples.

Data processing & encoding & class imbalance

Since there appears to be some features with strong correlations to each other, we first implement a dimensional reduction to generate those predictors that can better explain the response variable. As we are performing a binary model on "Rap"songs, we first convert the features of rap songs into ones, while non-rap songs are 0s. As can be seen in the table below, this is a class imbalance problem. We have more (roughly 95%) non-rap songs in the dataset. This would make our model harder to judge and a null model perform quite well.

Features(labels)	Count
1(Rap songs)	20566
0(Non-rap songs)	392727

Data splitting

After randomly splitting the data into training and test sets (20% held out for testing), we get 330831 observations in the training set and 82441 observations. Below is a screenshot that displays the class balance of each dataset after splitting. This does not look good， we will need to do a stratified sampling and down-sample the training dataset. The models we build will be much better at predicting non-rap songs because there are many more non-rap songs in the training data. It's usually easier to model the majority class because we have more training observations to learn from. Potential remedies can be up-sample the minority class and/or down-sample the majority class, we will down-sample the non-rap songs class in this case, if more time is given, we can try down-sampling or up sampling or their combination or observation weighting and compare the model performance. After down-sampling, there are 32907 observations in the training dataset, of which there is approximately 50% of rap songs. We can build models with the training data before down-sampling to make a comparison. PCA is introduced to reduce the dimensionality in the logistic regression problem to guarantee the impendence of variables.

Model comparison

Compared to logistic regression, random forest, decision tree and Gradient-boosted tree are more robust to collinearity, so we will not need to do a dimensional reduction before using these methods.

The models' performance metrics before down-sampling can be seen in the table below.

Model	Accuracy(4dp)	Precision(4dp)	Recall(4dp)
Logistic regression	0.9477	0.2582	0.02678
Random forest	0.9502	-	0
Decision tree	0.9502	-	0
Gradient-boosted tree	0.9502	-	0

The models' performance **after down-sampling** is as follows.

Model	Accuracy(4dp)	Precision(4dp)	Recall (4dp)
Logistic regression	0.7761	0.1564	0.7948
Random forest	0.7489	0.1444	0.8202
Decision tree	0.7637	0.1487	0.8202
Gradient-boosted tree	0.7637	0.1487	0.7917

As can be seen, the accuracies of the models before down-sampling are looking great. However, accuracy is not a good measure for a highly imbalanced problem, as the null model tends to perform well. The precisions do not exist because the total numbers of the predicted positives are zero, recalls are zero means the numbers of true positives are zero.

While after down-sampling is performed, precisions and recalls for all the methods are better, especially recalls, but neither precision nor recall tells the whole story, we are getting good recalls but still not quite good precision. We can calculate the F1 scores, which are about 0.25. This means there is still room for improvement.

Hyperparameter tuning - Cross validation

Below is a table that demonstrates how cross validation changes the auroc(area under the curve). As can be seen in the table, cross validation in this case does not improve the models much. Cross validation does not necessarily improve the model, it helps more accurately access the model performance.

Methods	Auroc before cross validation(4dp)	Auroc before cross validation(4dp)
Logistic regression	0.8378	0.8305
Random forest	0.8193	0.8167
Decision tree	0.8294	0.8351
Gradient-boosted tree	0.8294	0.8369

Multiclass classification

One vs one means we are dealing with a binary classification problem, in this case, we treat one class as positive class and the other class as negative class. When we have more than two classes, we will need a one vs. all strategy, one class is treated as positive class while the rest of the classes are treated as negative class. At prediction time, a voting scheme is applied: all C(C−1)/2C(C−1)/2 classifiers are applied to an unseen sample where C is the number of classes and the class that got the highest number of "+1" predictions gets predicted by the combined classifier. All-versus-all tends to be superior to one-versus-all. However, a problem with the schemes is that binary classifiers are sensitive to errors. If any classifier makes an error, it can affect the vote count. (Mma)

Method selection, encoding, training and evaluation

We will be performing a logistic regression method to train the model, as it takes care of multi-class problems. This is again a highly imbalanced problem with the smallest class accounts for 4.04% while the biggest class for 56%, but since we are expecting almost the same class priors for the future data, but oversampling or down-sampling can change the distribution of the data we use to build models. Taken time into consideration, we will skip this and use a randomized sampling to take 80% of the data to be used for training. Cross validation will be done and used for comparison to see if it can improve the model performance.

We access the model performance by getting its accuracy of 0.5635 before cross validation is performed, which is almost a null model(56%) and it remains the same after cross validation.

Song recommendation

In this section we use the Taste Profile dataset to develop a song recommendation system based on collaborative filtering.

Properties of the dataset

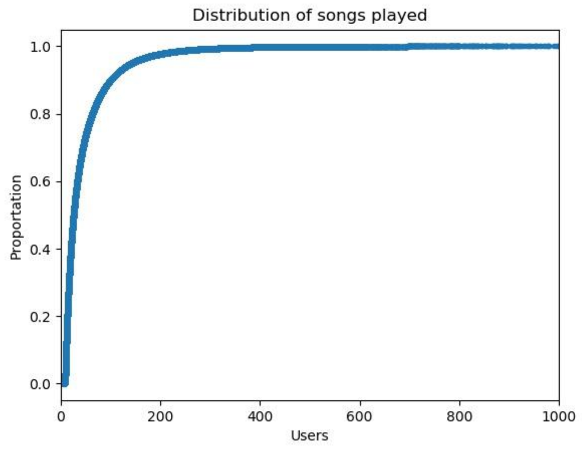
After filtering out the mismatches that are not manually accepted, we get the count of unique songs by dropping the duplicates in the song_id column, there are 378310 unique songs in the dataset. With the similar way, the number of unique users is 1019318.

We aggregate the data by user_id to get the count of songs each user played and the play times. The most active user has played 195 songs and the total play times is 13074, which means he has listened to 0.05% of all the unique songs.

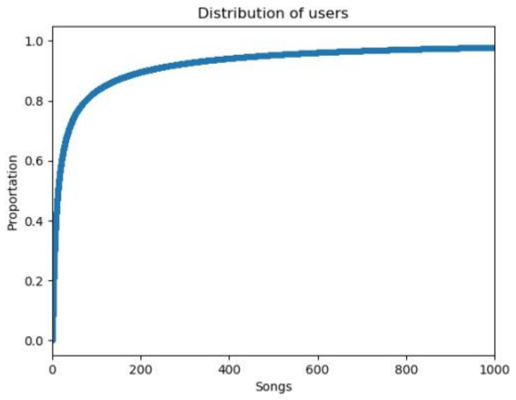
To get the song popularity, we aggregate by song_id to get the number of users of each song and how many times it has been played.

Distribution visualization

We then visualize the distribution of song popularity and the distribution of user activity.



As can be seen in the plot above, there are few songs that have not been played and there are nearly 90% of songs that have been played by less than 200 users.



There are about 40% of users who barely listen to any songs and about 80% of users have only listened to about 100 songs. These two distribution plots give us a better understanding of the percentage of "popular' songs in the dataset.

Threshold selection & data splitting

Having taken this into account, we remove songs which have been played less than 40 times and users who have listened to fewer than 55 songs in total, this gives us 106889 songs, which is 28.25% of the total unique songs and 441103 active users which is 23.45% of the total unique users.

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, as the main idea of collaborative filtering is to filter out the songs that one user might like based on his taste and other users with similar tastes, which means the songs the user has played before is always required.

After splitting the data, we get 5052998 observations in the test set, which is roughly 20% of the clean user-song dataset and 20688670 observations in the training, this satisfies the requirement that the test set contains at least 20% of the plays in total. This is done by extracting the row numbers of the dataset and choose those that can be divisible by 5 to be used in the testing set, while those that are not divisible by 5 are used for the training set. (This question is out of my depth and l am using someone's codes).

We also need to check if there are any users that are contained in test dataset but not in the training dataset, by "left-anti' joining the two datasets, we find out that there are no unique users in the test data. Also, the number of users of the training set is exactly the same as that of the test set.

Effectiveness of the collaborative filtering model

We select 5 users from the test set by hand and use the model to generate the songs that are recommended, we then get the songs that the users have actually played and join the two datasets to see if there are any mutual items.

User	Songs that are recommended	Song that are played.
6.0	13.0, 9.0, 6.0, 28.0, 48.0	34243.0, 14324.0, 3631.0, 3155.0, 824.0
7.0	236.0, 101.0, 372.0, 790.0, 356.0	3564.0, 991.0, 6549.0, 38675.0, 13053.0
12.0	358.0, 487.0, 1965.0, 2771.0, 494.0	2771.0, 363.0, 2124.0, 10095.0, 15680.0, 2117.0
13.0	2.0, 32.0, 58.0, 11.0, 90.0	5836.0, 10166.0, 12761.0, 676.0, 31167.0
55.0	201.0, 158.0, 352.0, 420.0, 115.0	158.0, 1906.0, 1496.0, 5721.0, 1335.0

As can be seen in the table above, there appears to be some duplicates in the two outputs, but not many. This is probably because we are only generating 5 items to be recommended and we are comparing the recommendations with all the songs that are actually played. We will need some other measures to access the model performance.

Ranking metrics

We then use the test set of user-song plays and recommendations from the collaborative filtering model to compute the following metrics, the outputs can be seen below.

Metrics	Values(4dp)
Precision @ 5	0.5617
NDCG @ 10	0.43912
Mean Average Precision (MAP)	0.2695

Precision@5 means the number of relevant items out of the top 5 items, this can be one of the criteria of how good the model is. one of its cons is that it does not take order into account.

Similar to precision, NDCG@10 means it is evaluated on the top 10 items. it takes care of non-binary relevance for allowing real numbers, but it does not work when we have incomplete ratings or users have no relevant documents, we need to figure out a way to set the values.

MAP refers to the mean average precision for all the users in the dataset, average precision actually means the percentage of the correct item at the correct order, it is useful as we always want to display the relevant (correct) items first in a recommendation system. However, the limitation of MAP is obvious because when it covers all the queries, it ignores the orders. In addition, it simply assumes binary relevance.

Root mean square (RMSE) and mean absolute error(MAE) can be used to evaluate these eight recommender system algorithms(Pradeep, Immidi Kali, and M. Jaya Bhaskar).

If we could measure future user-song plays based on the recommendations, MAR@k is also a great way to access the model performance, as it gives insight into how well the recommender is able to recall all the items has positively in the test set.

Limitations of collaborative filtering & recomdies

Collaborative filtering system has three different potential limitations, data sparsity, scalability and cold start problem. Data sparsity is caused by the big amount of data used in the system and the cold start problem. As collaborative filtering methods recommend items based on users' past preferences, new users will need to rate sufficient number of items to enable the system to capture their preferences accurately and thus provides reliable recommendations. Similarly, new items also have the same problem. When new items are added to system, they need to be rated by substantial number of users before they could be recommended to users who have similar tastes with the ones rated them. (Madhukar, Mani).

When a cold start problem occurs, the system is not able to provide any recommendations as the user's taste or is unknown.

As the main idea of collaborative filtering is to filter out the songs that one user might like based on his taste and other users with similar tastes, the algorithm improves when we know more about a user's taste, which means the more songs he plays. However, when we do not have enough information about the user's taste, this gives us a low quality recommendation.

Demographic filtering system can be a useful alternative algorithm for a brand new user,
as it is based on demographic characteristics of consumers and recommends a list of items that have good feedback from the consumers that are demographically similar to the target consumer(What Is Demographic Recommender Systems).

For the low quality problem, we can choose a hybrid approach, which combines a few recommendation methods, when it combines collaborative or content-based systems, it helps with the lack of information about the domain dependencies in collaborative filtering, and about the people's preferences in content-based system(Geetha, G., et al).

How to improve the system performance

Based on the song metadata provided in the million song dataset summary, l would say we can recommend songs based on the songs the user has played and the song's artist to improve the real world performance of the system.

Reference

1. "Pyspark Package¶." Pyspark Package - PySpark 2.4.5 Documentation, spark.apache.org/docs/latest/api/python/pyspark.html?highlight=repartition.
2. joparga3. "2. Tuning Parameters for Logistic Regression." Kaggle, Kaggle, 13 Dec. 2016, www.kaggle.com/joparga3/2-tuning-parameters-for-logistic-regression.
3."Linear Methods - RDD-Based API." Linear Methods - RDD-Based API - Spark 2.4.5 Documentation, spark.apache.org/docs/latest/mllib-linear-methods.html#logistic-regression.
4. Elsinghorst, Shirin. "Machine Learning Basics - Random Forest." *Shirin's PlaygRound*, 30 Oct. 2018, www.shirin-glander.de/2018/10/ml_basics_rf/.
5. Mma. "Multiclass Classification - One-vs-Rest / One-vs-One." *Mustafa Murat ARAT*, 2 Oct. 2019, mmuratarat.github.io/2019-10-02/multi-class-classification.
6. Pradeep, Immidi Kali, and M. Jaya Bhaskar. "Comparative analysis of recommender systems and its enhancements." *International Journal of Engineering & Technology* 7.3.29 (2018): 304-310.
7. Madhukar, Mani. "Challenges & limitation in recommender systems." *International Journal of Latest Trends in Engineering and Technology (IJLTET)* 4.3 (2014): 138-142.
8. "What Is Demographic Recommender Systems." *IGI Global*, www.igi-global.com/dictionary/recommender-systems-overview/7203.
9. Geetha, G., et al. "A hybrid approach using collaborative filtering and content based filtering for recommender system." *Journal of Physics: Conference Series*. Vol. 1000. No. 1. IOP Publishing, 2018.