**DATA420-21S2 (C)**

**Assignment 2**

**The Million Song Dataset (MSD)**

Xin Gao (43044879)

October 29, 2021

**Foreword**

In this assignment we will investigate a collection of datasets referred to as the Million Song Dataset (MSD), a project initiated by 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.

The code used to solve each of the questions is provided separately and is commented well. This report describes the approach used, answers any questions, and describes anything unexpected along the way.

**Data Processing**

Q1:

1. The data is in four parts: audio, genre, main and tasteprofile.

***Audio***: with three folders.

1, attributes (13 different attribute files stored in csv format)

2, features (13 folders with names shared the same prefix with attributes consistently, stored in 8 distributed compressed csv files for each folder)

3, statistics (one compressed csv file)

***Genre***: three tsv files

***Main***: two different compressed csv files in summary folder

***Tasteprofile***: two txt files and 8 distributed compressed tsv files in triplets.tsv folder

The data is structured according to the directory tree below

hdfs:///data/msd/

├─ audio

│ ├─attributes

│ │ ├─msd-jmir-area-of-moments-all-v1.0.attributes.csv

│ │ ├─msd-jmir-lpc-all-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-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

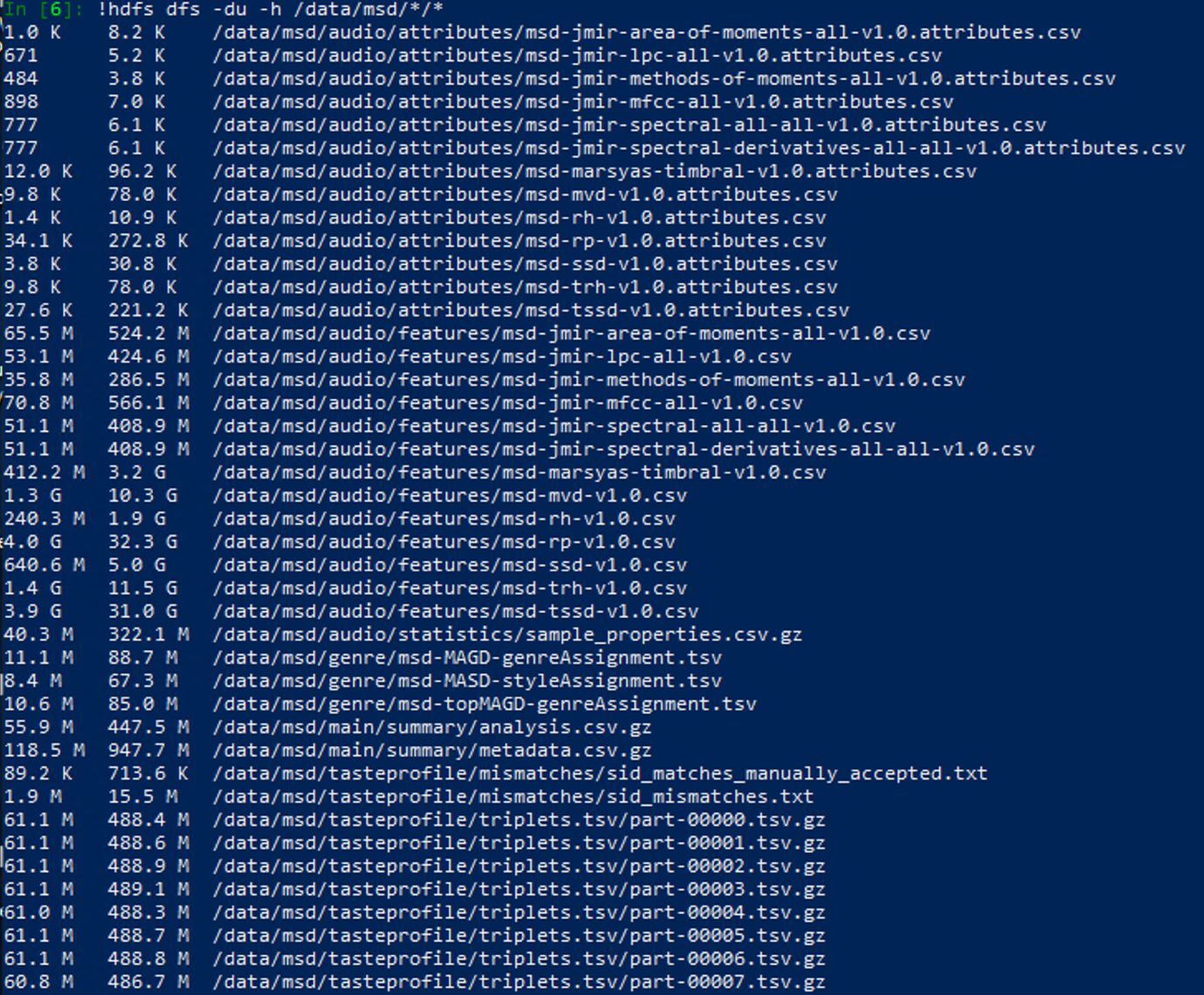
└─triplets.tsv

├─part-00000.tsv.gz

├─...

└─part-00007.tsv.gz

The size of data is shown below



1. For compressed data, as spark load them as a whole, “repartition” would be useful to break them down after loading. But for datasets that are small in size e.g. attributes, it is not necessary to do repartition.
2. In metadata.csv, the number of rows is 1 million like the name of the dataset. In audio features dataset, the number of rows is around 994.6 thousand which means 99.46% of the songs have been assigned audio features. In genre dataset, the number of rows is 422.7 thousand which means 42% of the songs are labelled with genres. In tasteprofile dataset, the number of rows is about 45.8 million. It is significantly larger than the number of unique songs because the dataset is a collection of user, song, play count triplets.

Q2:

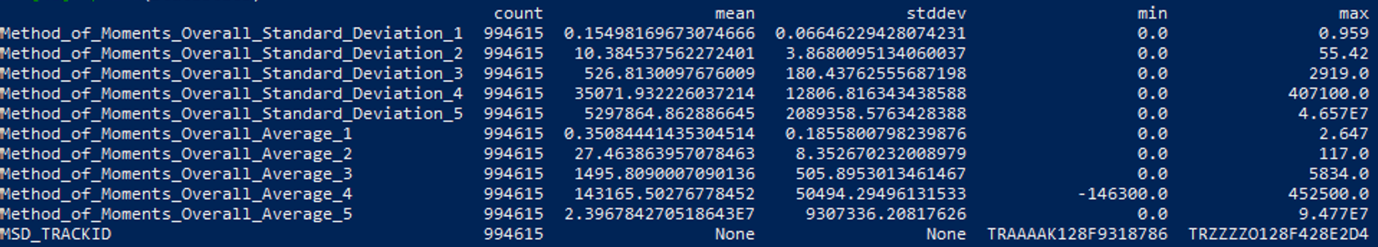
1. After extracting the song id and track id from the “sid\_mismatches.txt” file, there are 19094 mismatched songs in the dataset. Load the “sid\_matches\_manually\_accepted.txt” file as well. Note that the we need to filter out the extra rows with the startswith ("< ERROR: ") method and then extract the song id and track id. There are 488 manually accepted songs in the dataset. Remove the manually accepted songs from the mismatched songs, and get the count 19093. Load the triplets dataset, and there are 48373586 rows in total. Then remove the mismatched songs from the triplets dataset. The triplet dataset is reduced from 48373586 rows to 45795111 rows.
2. Did not use the codes provided on Learn because at first the file path format did not work for me (It worked afterward). Then I referred to the provided codes and wrote my own codes to carry out the feature-attributes function. The codes do not automatically create all the datasets with features and corresponding attributes. You need to choose a number in the “audio\_dataset\_names” list as a parameter and use the “features\_df” function to create the dataset.

**Audio similarity**

Q1:

1. Pick the smallest dataset “msd-jmir-methods-of-moments-all v1.0.csv” (35.8 M)

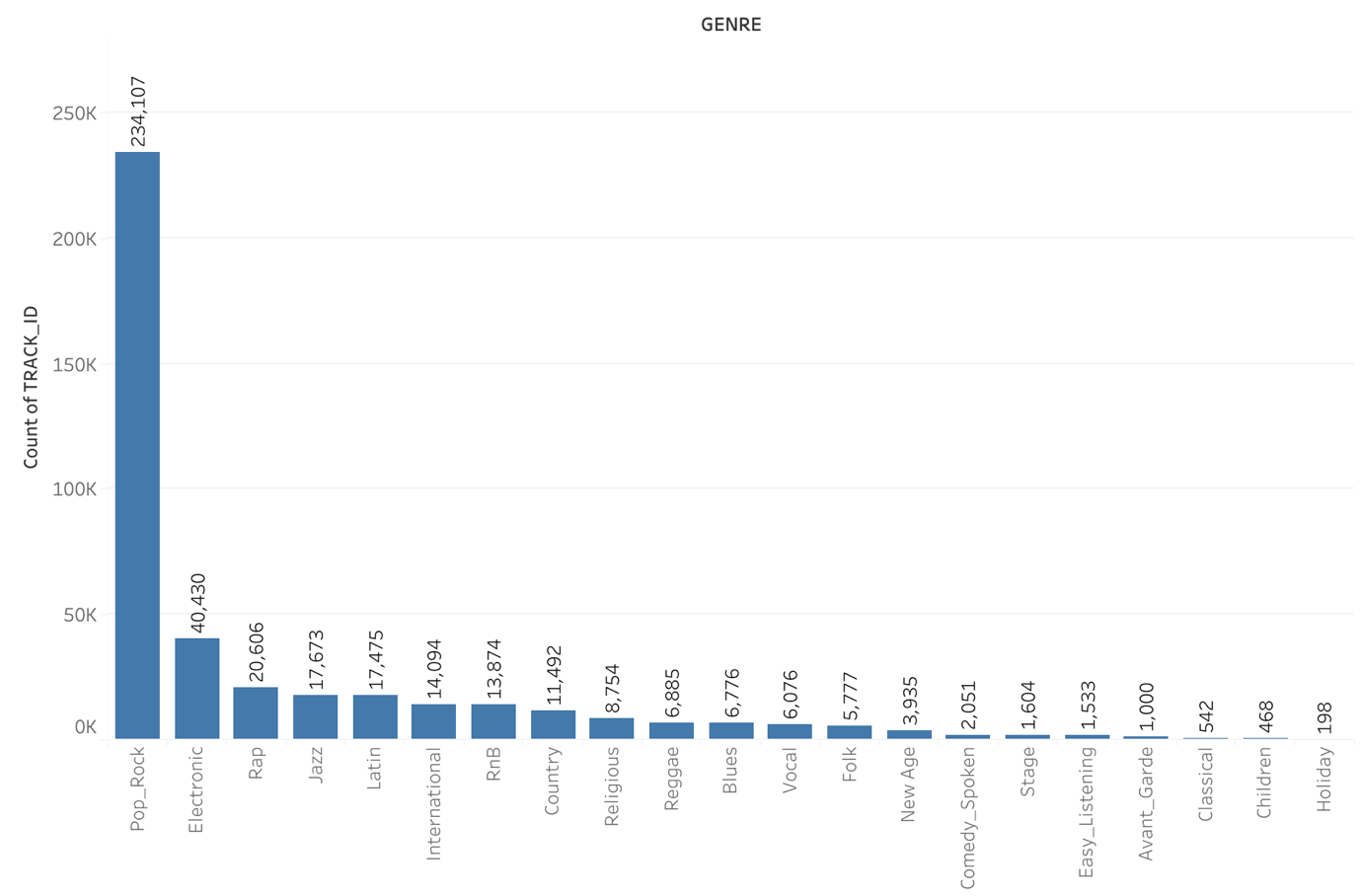
The descriptive statistics are below



The numeric features have a variety of scales. The means range from 0.15 to 143165 across the features, and the standard deviations range from 0.066 to 9307336. Centring and scaling is necessary due to the variety in means and variances of the numeric data.

According to the correlation matrix, I decided to set the correlation threshold to 0.8. The columns (1,2) (2,3) (3,4) (6,7) (8,9) are correlated. I removed 2, 3, 7, 9 columns (columns start from 0).

1. The distribution of genres for the songs that were matched (415350 in total) is shown below



We can see that “Pop\_Rock” is significantly more than other genres.

1. When merging the matched genres dataset and the audio features dataset, at first the genre column showed “null”. It took me a while to realize that the features dataset “TRACK\_ID” column values are with quotes. Then I went back to the codes where loading the features dataset and got rid of the quotes. To ensure that every song has a label, I chose the joining method to “inner” which means only the TRACK\_ID appears in both datasets will produce an output. The final count is 413289.

Q2:

1. Logistic regression is the first choice. It is a simple algorithm, very cheap to train, does not take long, very repeatable, not complicated to implement. It provides a base line and most of performance and not much tuning involved.

Second chosen algorithm is random forest. As if this problem is not linearly separable, a tree based model might be a good choice. See if it can perform better than the logistic regression model.

The last chosen algorithm is Gradient-boosted trees. It is a good pick for good performance especially for categorical although we have all the numeric predictors in this case.

Centring and scaling is necessary due to the variety in means and variances of the numeric data.

1. The class balance is

label count ratio

1 40026 0.096847

0 373263 0.903153

Label 1 is only near 10% and label 0 is just over 90%.

1. d) e)

I tried different sampling methods and applied them to logistic regression model, the metrics are shown below

|  |  |  |  |
| --- | --- | --- | --- |
|  | 8-2 stratified sampling  (1: 10%, 0:90%) | Down sampling  (1: 22%, 0:78%) | Up sampling  (1: 24%, 0:76%) |
| Precision | 0.490797 | 0.416324 | 0.393342 |
| Recall | 0.009992 | 0.114039 | 0.144641 |
| AUROC | 0.707867 | 0.707956 | 0.707998 |

The AUROC metrics are almost identical which means the sampling method did not change the overall performance. Looking at he precision and recall trade-off, up sampling did the best job. So I again tried different up sampling ratios with logistic regression model and recorded the metrics below

|  |  |  |  |
| --- | --- | --- | --- |
|  | Up sampling  (1: 24%, 0:76%) | Up sampling  (1: 30%, 0:70%) | Up sampling  (1: 35%, 0:65%) |
| Precision | 0.393342 | 0.315578 | 0.267771 |
| Recall | 0.144641 | 0.245690 | 0.333125 |
| AUROC | 0.707998 | 0.707601 | 0.707416 |

The AUROC metrics are almost identical again. I chose the last up sampling method because the precision and recall trade-off is the best.

Then try different threshold on up sampled training dataset, metrics are below

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Threshold | 0.2 | 0.3 | 0.4 | 0.5 |
| Precision | 0.113977 | 0.147626 | 0.20005 | 0.267771 |
| Recall | 0.901199 | 0.757307 | 0.549213 | 0.333125 |
| AUROC | 0.708628 | 0.708630 | 0.708632 | 0.707416 |

The AUROC metrics are almost identical again. The default threshold 0.5 has got the best precision-recall trade-off.

Then trained different models with up-sampling(1: 35%, 0:65%), the results are shown below

|  |  |  |  |
| --- | --- | --- | --- |
|  | Logistic Regression | Random Forest | Gradient-boosted Tree |
| Precision | 0.267771 | 0.274946 | 0.276128 |
| Recall | 0.333125 | 0.461528 | 0.522732 |
| AUROC | 0.707416 | 0.757095 | 0.784549 |

Gradient-boosted tree is the best model in my attempts.

f) As this is an imbalanced class problem, models tend to predict the majority class even if I used the up-sampling method to compensate this situation. And obviously, accuracy should not be used as a metric because it looks good but actually misleading. AUROC should be used as an overall performance measurement while also keep an eye on the precision recall trade-off. If I want to predict more actual electronic songs, based on the overall AUROC, I would prefer to see the recall higher while the precision is not too bad like my final model shows.

Q3:

1. Gradient Boosting uses an ensemble of decision trees to predict a target label. Chose maxDepth, maxBins and step size to build the parameter grid.

maxDepth - int, optional

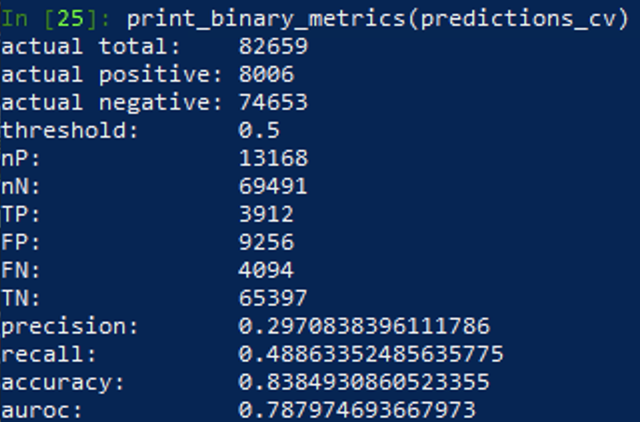
Maximum depth of tree (e.g. depth 0 means 1 leaf node, depth 1 means 1 internal node + 2 leaf nodes). (default: 3)

maxBins - int, optional

Maximum number of bins used for splitting features. DecisionTree requires maxBins >= max categories. (default: 32)

Step size – learning rate. (default: 1)

1. Use cross-validation to tune the three hyperparameters of the gradient boosted trees. The result is shown below

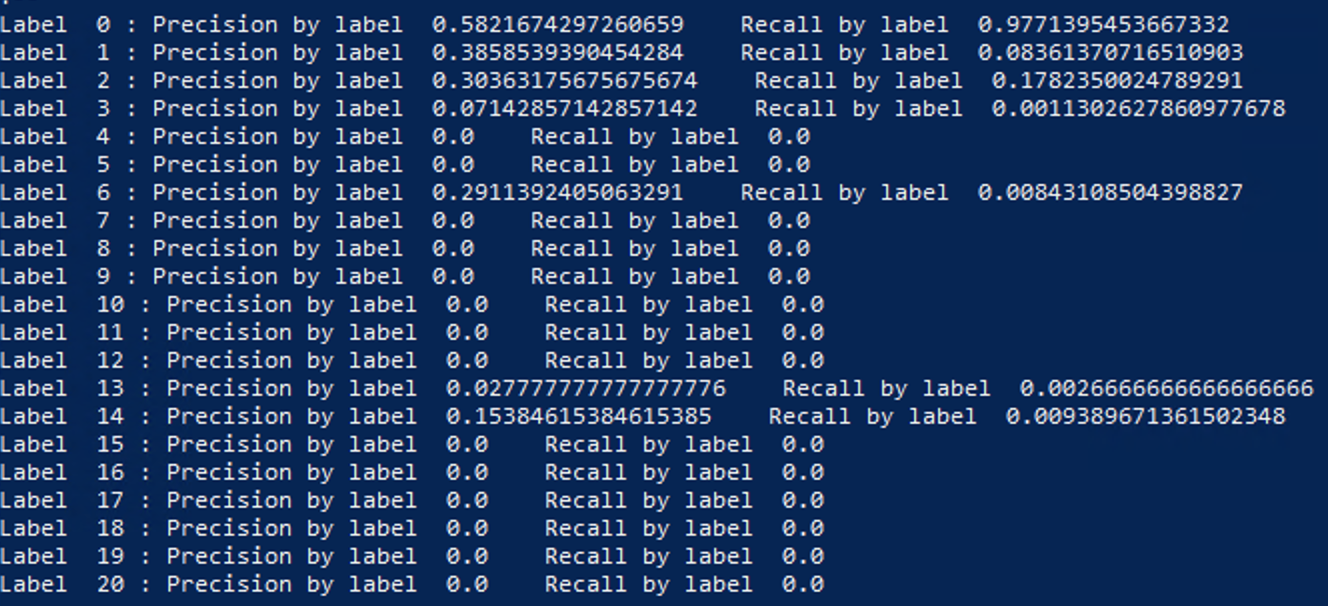


It is just a little better than before (0.787974 compared to 0.784549) in terms of AUROC.

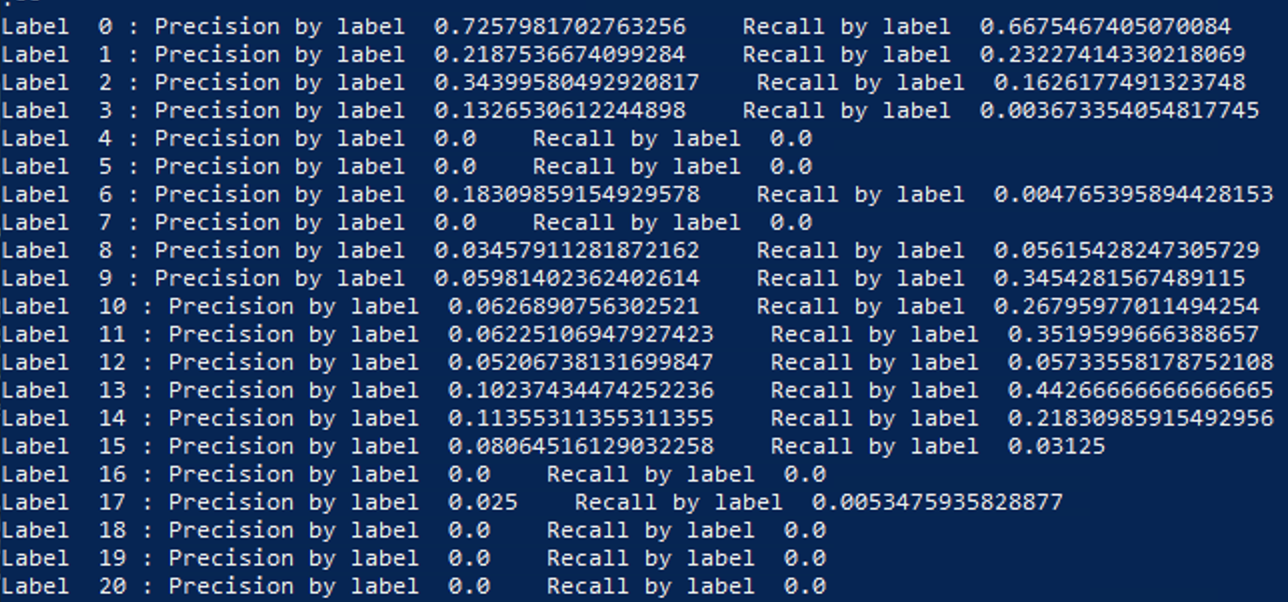
Q4:

1. Tried gradient boosted trees to do the multiclass classification, but failed. Then used the logistic regression. It can predict one class out of multiple classes by treating one class and the rest of the classes as two classes.
2. Use the StringIndexer feature to encode each genre consistently. There are 21 genres in the dataset which are labelled from 0 to 20.
3. I compared the results with and without observation reweighting to the training dataset, see metrics below

Without weighting



With weighting



It is clear that the majority class’s performance is much better than other classes. Without the reweighting, the pop\_rock class has a recall of 0.977 because it is absolutely dominant in the dataset. That means almost all the pop\_rock songs are correctly labelled. However the precision is much lower than recall which means many songs of other genres are also labelled as pop\_rock.

With the observation reweighting, the overall performance is improved. The majority class performance is more balanced, and the minor classes are significantly improved in both precision and recall although there are still genres with zero in their measurements.

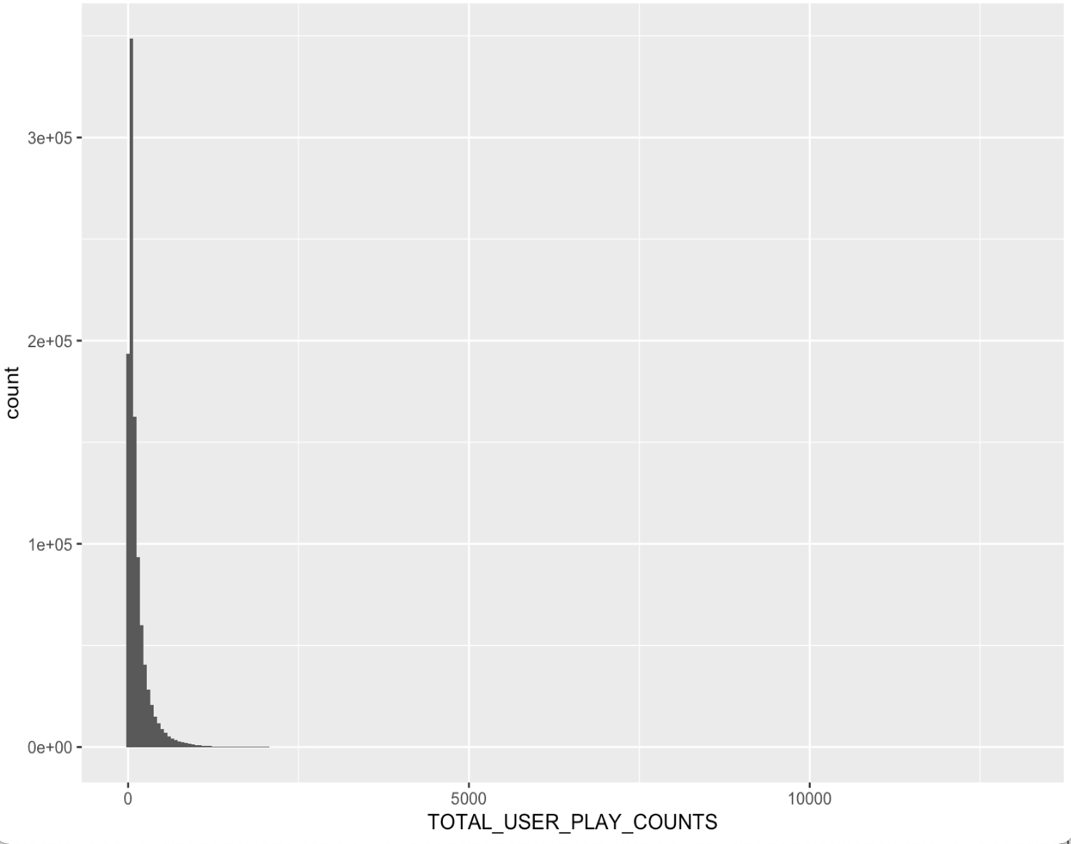
In terms of the electronic genre, with multiclass classification, the overall performance is worse than binary classification.

**Song recommendation**

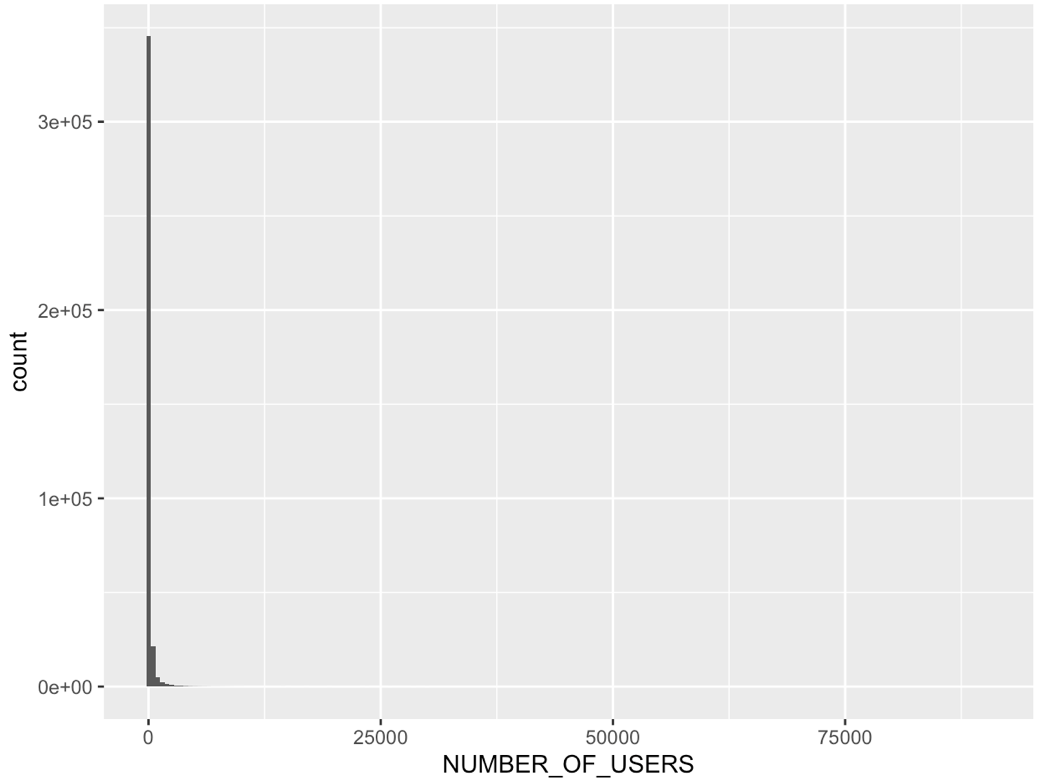
Q1:

1. There are 378310 unique songs and 1019318 unique users in the triplets dataset.
2. The most active user played 4316 songs which accounts for 1.14% of the total number of unique songs in the dataset.
3. Both the user activity and song popularity distributions are significantly left skewed. See the charts below

User activity



Song popularity



1. Choose the 50% quantile as a threshold and reduce the triplets to 36602346 from originally 45795111 rows.
2. Used 70-30 random split to split the training and test sets. Because of the nature of collaborative filtering model, it makes predictions based on the users’ history. So it can only predict the users when they appeared in the training set.

To do this, create the dataset containing the users in test dataset but not in training dataset. Sort by user id and filter 70% of the items in each user id as a temporary dataset. Union the temporary dataset to the training dataset so that all the users in test dataset are contained in training dataset.

Q2:

1. Set the parameter “implicitPrefs=True” to train an implicit matrix factorization model using the play counts as the weights.
2. Randomly choose the user encoded id 333

Recommended songs:

[12, 0, 87, 49, 93]

Relevant songs:

[3956, 1447, 68178, 12267, 7984, 4363, 44421, 3230, 1881, 263, 5245, 1078, 20208, 3234, 4042, 2500, 9059, 617, 26016, 9306, 399, 20807, 2080, 866, 2309, 35431, 89739, 1505, 2207, 4697, 465, 52429, 65859, 1658, 2973, 21419, 1434, 7484, 20684, 47565, 65018, 36675, 57319, 12099, 53200, 1324, 46528, 37495, 10338, 30036, 6614, 2987, 23599, 12329, 43810, 79156, 36722, 115779, 19310, 48145, 42988, 2028, 130949, 80913, 50299, 2335, 59725, 16222, 27623, 33765, 28526, 68973, 21683, 568, 10058, 6440, 16190, 1383, 41603, 4431, 17337, 32650, 7291, 4203, 169038, 27729, 79346, 72141, 37557, 18548, 38232, 5131, 3074, 9384, 745, 20956, 23765, 77028, 71556, 554, 43050, 1168, 128956, 36947, 1706, 77072, 3854, 1106, 12015, 950, 1144, 47237, 8919, 100, 112676, 3053, 23773, 48771, 69786, 12243, 238, 1112, 2826, 13356, 75782, 28582, 152753, 1492, 34228, 41483, 2918, 8394, 11211, 26801, 51909, 2891, 9282, 12467, 11825, 2008, 35944, 546, 14842, 99462, 51630, 26813, 15420, 2193, 24885, 139457, 39369, 35814, 2219, 3753, 124942, 6734, 2244, 8400, 39222, 11337]

None of the recommended songs is in relevant songs

Now choose an active user encoded id 3

Recommended songs:

[15, 20, 23, 4, 21]|

Relevant songs:

[200, 2536, 4358, 1272, 1523, 159, 1877, 755, 15, 857, 5, 2057, 183, 36, 13366, 147, 2714, 48, 1338, 96831, 680, 19692, 8980, 2110, 6747, 2143, 51, 2983, 116, 104459, 842, 2242, 56018, 425, 53725, 175, 1620, 35192, 12338, 2281, 68424, 6915, 638, 36469, 5012, 9717, 11842, 8929, 9463, 2954, 44836, 99374, 35966, 539, 4181, 1595, 1139, 51410, 104456, 413, 125504, 23463, 469, 120, 2696, 83932, 1675, 20394, 1761, 703, 13924, 3051, 1225, 5006, 37126, 32017, 6061, 1968, 439, 5287, 795, 2163, 568, 1076, 17389, 7917, 6436, 9611, 2853, 13427, 9597, 8178, 1573, 13440, 15447, 2139, 1216, 571, 6, 3607, 48120, 4785, 6658, 709, 3135, 26910, 2445, 26165, 1989, 583, 4431, 8366, 3672, 798, 19595, 16570, 129041, 4, 3156, 4729, 10190, 6730, 6444, 1774, 3131, 33579, 1874, 10, 1527, 13108, 2561, 16321, 87278, 25992, 1700, 1270, 15454, 8425, 334, 3642, 3119, 1723, 500, 215, 3088, 94304, 2463, 18994, 16555, 1832, 354, 61847, 94, 1663, 8240, 7040, 16883, 2855, 530, 8278, 868, 11205, 6320, 67861, 2648, 3107, 1473, 332, 43855, 2472, 32321, 859, 751, 3186, 1161, 1669, 87914, 54347, 10607, 2292, 28872, 5138, 514, 15662, 2914, 1260, 15119, 4372, 1218, 1763, 985, 3312, 9002, 68708, 6227, 8394, 1654, 4240, 61, 1001, 9481, 13336, 60050, 3359, 8885, 1515, 47623, 10670, 458, 34261, 15178, 467, 2025, 16919, 50275, 1010, 89577, 15041, 27334, 11766, 10733, 9155, 58, 22099, 68940, 5769, 8402, 7095, 28428, 3568, 9188, 8033, 7928, 3130, 3919, 10442, 35988, 136, 5787, 13372, 24090, 24837, 32741, 36906, 3618, 23327, 2278, 2691, 161671, 513, 13593, 158198, 7729, 1020, 1229, 182026, 7890, 9536, 20764, 2340, 89731, 31952, 1734, 22385, 153, 175073, 2055, 10477, 278, 161900, 134258, 61456, 38210, 88669, 69034, 1906, 2886, 1156, 131374, 9589, 7323, 5606, 20476, 7999, 339, 5581, 405, 86614, 8943, 8000, 5349, 41806, 19160, 88743, 3745, 5044, 51767, 12628, 419, 16797, 18641, 51219, 43676, 12981, 1679, 4101, 252, 979, 450, 18604, 16386, 43864, 21406, 17123, 42434, 29666, 40330, 28452, 20232, 8680, 4653, 338, 229, 1066, 286, 4836, 12216, 6973, 24340, 40882, 14757, 23566, 11039, 56818, 259, 2866, 40171, 9130, 58696, 55714, 1463, 50, 882, 6588, 16746, 173071, 4270, 35910, 1543, 4390, 595, 1659, 159622, 392, 1341, 7975, 182, 301, 606, 8203, 89093, 1027, 1147, 16, 52166, 75, 1276, 34924, 3493, 894, 1026, 7373, 34073, 33581, 12222, 38986, 973, 44606, 759, 51034, 19922, 23142, 9634, 50752, 20309, 23301, 23789, 3602, 523, 13506, 2627, 26960, 3663, 66010, 179, 26884, 1117, 1593, 30318, 3791, 5198, 7835, 2493, 15303, 1025, 3094, 847, 1204, 4243, 50793, 1072, 1630, 130200, 19848, 64, 16426, 670, 12946, 826, 43588, 149888, 27, 2650, 30230, 23162, 40409, 31798, 114, 14907, 19142, 2817, 101099, 1367, 94205, 28328, 347, 79893, 109626, 1127, 2694, 3735, 29615, 650, 5501, 181686]

There are two recommended songs in relevant songs.

The more the user interacted with the songs, the better collaborative filtering model performs.

1. Set N = 31, M = 26, the metrics are shown below

|  |  |
| --- | --- |
| Precision @ 10 | 0.025652809734823964 |
| NDCG @ 10 | 0.035131396903573614 |
| MAP | 0.007966674384688845 |

Change N = 1000, M = 100, the metrics improved below

|  |  |
| --- | --- |
| Precision @ 10 | 0.04143192136905654 |
| NDCG @ 10 | 0.055527255204457046 |
| MAP | 0.0067677319185041225 |

Precision is intuitive, but same reward is applied to different ranks. It is better that lower ranks to have less value.

NDCG extends the concept of position(rank) matters, but the rank penalty is linear, not log scaled, which is strong.

Mean average precision is an average of precisions of predictions of all users.

The ranking metrics are all offline with a static set of relevant items. However, it is unlikely that they will perform well.

A/B testing(splitting users into A and B groups) could be used to compare two recommendation systems in the real word.

If I could measure future user-song plays, there are three metrics could be useful.

1. Users’ click/interactive rate of the recommended songs
2. Users’ playing time of the recommended songs
3. Users’ rating of the recommended songs