Song recommendations based on MSD using Spark

DATA420Assignment 2

Xiaocui Zhang 16142030
UNIVERSITY OF CANTERBURY

Contents

1 About the data ..... - 1 -
2 Data processing ..... - 1 -
2.1 Understand the data ..... - 1 -
2.1.1 Data structure ..... - 1 -
2.1.2 Data size and repartition ..... - 2 -
2.1.3 Number of rows in each dataset ..... - 4 -
2.2 Data Pre-processing ..... - 6 -
2.2.1 Taste Profile data cleaning ..... - 6 -
2.2.2 Audio Features loading ..... - 6 -
3 Audio similarity ..... - 6 -
3.1 Exploratory data analysis ..... - 6 -
3.1.1 Method of Moments exploration ..... - 7 -
3.1.2 MAGD exploration ..... - 8 -
3.1.3 Label the songs ..... - 9 -
3.2 Binary classification model ..... - 9 -
3.2.1 About classification models ..... - 10 -
3.2.2 Split the dataset and train the models ..... - 11 -
3.2.3 Performance metrics ..... - 13 -
3.3 Multiclass classification ..... - 14 -
4 Song recommendations ..... - 16 -
4.1 Taste Profile exploration and filtering ..... - 16 -
4.2 Split the dataset and train the collaborative filtering model ..... - 19 -
4.3 Model evaluation and ranking metrics ..... - 20 -
4.4 Some discussion ..... - 22 -
Works Cited ..... - 24 -

1 About the data

The collection of datasets is referred to as the Million Song Dataset (MSD),a project initiated by The Echo Nest and LabROSA, and the core of the dataset is the feature analysis and metadata for one million songs, provided by The Echo Nest. It also contains other datasets contributed by organizations and the community.

2 Data processing

2.1 Understand the data

2.1.1 Data structure

The data is structured according to the directory tree in Figure 1.

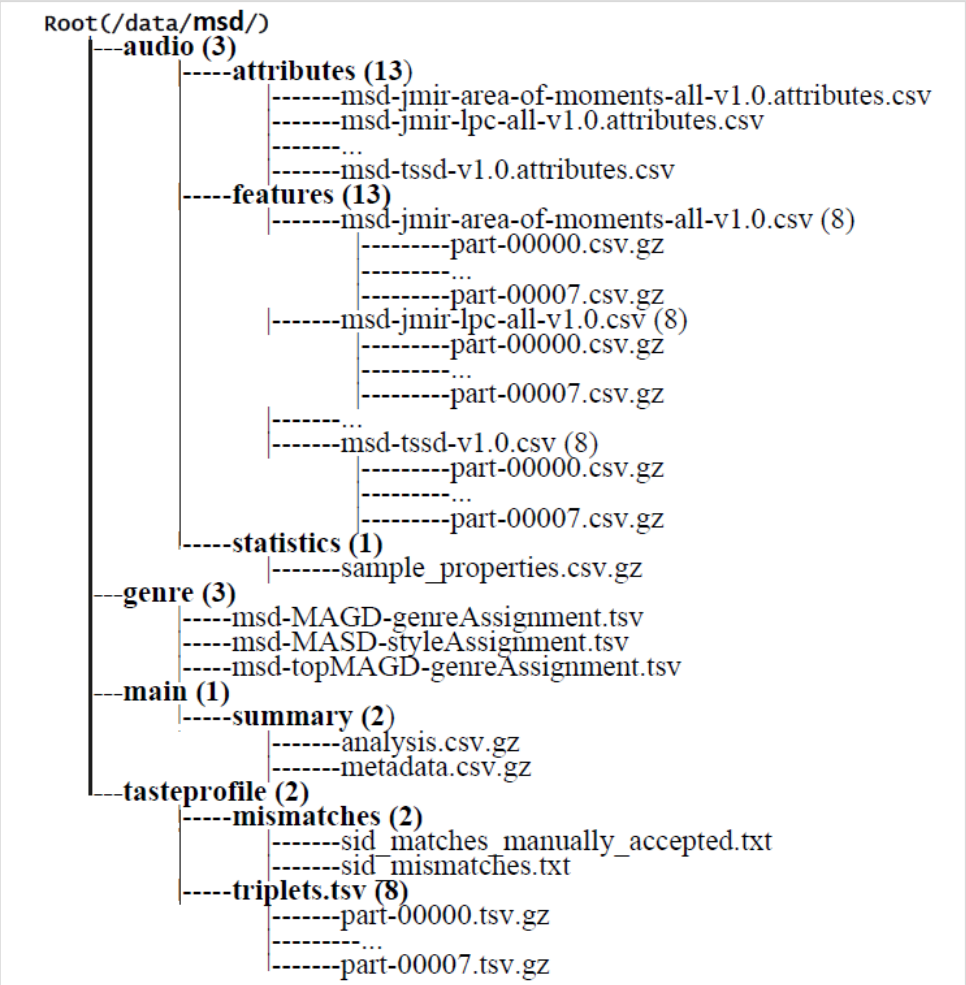

Figure 1 The overview of the structure of the datasets

2.1.2 Data size and repartition

The size of the data is **12.9 G** in total, but it takes up **103.5 G** disk space in the cluster because of the 8 replications.

Table 1 The size of dataset

Data set	File(s) size	Disk space in the cluster
audio	**12.3 G**	**98.1 G**
audio/features	**12.2 G**	**97.8 G**
msd-jmir-area-of-moments-all-v1.0	65.5 M	524.2 M
msd-jmir-lpc-all-v1.0	53.1 M	424.6 M
msd-jmir-methods-of-moments-all-v1.0	35.8 M	286.5 M
msd-jmir-mfcc-all-v1.0	70.8 M	566.1 M
msd-jmir-spectral-all-all-v1.0	51.1 M	408.9 M
msd-jmir-spectral-derivatives-all-all-v1.0	51.1 M	408.9 M
msd-marsyas-timbral-v1.0	412.2 M	3.2 G
msd-mvd-v1.0	1.3 G	10.3 G
msd-rh-v1.0	240.3 M	1.9 G
msd-rp-v1.0	4.0 G	32.3 G
msd-ssd-v1.0	640.6 M	5.0 G
msd-trh-v1.0	1.4 G	11.5 G
msd-tssd-v1.0	3.9 G	31.0 G
audio/attributes	**103.0 K**	**824.3 K**
msd-jmir-area-of-moments-all-v1.0	1.0 K	8.2 K
msd-jmir-lpc-all-v1.0	671	5.2 K
msd-jmir-methods-of-moments-all-v1.0	484	3.8 K
msd-jmir-mfcc-all-v1.0	898	7.0 K
msd-jmir-spectral-all-all-v1.0	777	6.1 K
msd-jmir-spectral-derivatives-all-all-v1.0	777	6.1 K
msd-marsyas-timbral-v1.0	12.0 K	96.2 K
msd-mvd-v1.0	9.8 K	78.0 K
msd-rh-v1.0	1.4 K	10.9 K
msd-rp-v1.0	34.1 K	272.8 K
msd-ssd-v1.0	3.8 K	30.8 K
msd-trh-v1.0	9.8 K	78.0 K

msd-tssd-v1.0	27.6 K	221.2 K
audio/statistics	**40.3 M**	**322.1 M**
sample_properties	40.3 M	322.1 M
genre	**30.1M**	**241.0M**
msd-MAGD-genreAssignment	11.1 M	88.7 M
msd-MASD-styleAssignment	8.4 M	67.3 M
msd-topMAGD-genreAssignment	10.6 M	85.0 M
main/summary	**174.4 M**	**1.4 G**
analysis	55.9 M	447.5 M
metadata	118.5 M	947.7 M
tasteprofile	**490.4 M**	**3.8 G**
tasteprofile/mismatches	**2.0 M**	**16.2 M**
sid_matches_manually_accepted	89.2 K	713.6 K
sid_mismatches	1.9 M	15.5 M
tasteprofile/triplets.stv	**488.4 M**	**3.8 G**

All the data is stores in text files and some of them are compressed. The details are shown in Table 2.

Table 2 File formats

File formats	Filename extension	Subsets
text document	.txt	tasteprofile/mismatches
	.csv	audio/attributes
	.tsv	genre
archive and compressed	.csv.gz	audio/features； audio/statistics； main/summary
	.tsv.gz	Tasteprofile/triplets.stv

In general, “when Spark reads a file from HDFS, it creates a single partition for a single input split. Input split is set by the Hadoop *Input Format* used to read this file. If it is *TextInputFormat* in Hadoop, which would return us a single partition for a single block of HDFS while the split between partitions would be done on line split, not the exact block split”(Degget). The default block size is 128M, but all uncompressed text files are quite small. Hence, we can get one

partition for each file respectively. Meanwhile, spark does not split a compressed file even if the file size is larger than 128M(default size), and we can get only one partition for one compressed file.

We could repartition a big dataset into small ones so there would be more parallel tasks in Spark. However, the repartition transformation will trigger shuffle, which is an expensive operation(I/O, duration). Reasonable partitions enable us to utilize the cores in the cluster sufficiently and avoid excessive overheads in managing small tasks. The optimal number of tasks is 2 to 3 times the number of CPU cores.

In this case, we could try to repartition the big compressed file after loading it into Spark in consideration of the number of allocated CPU cores. For instance, the triplets.stv includes eight compressed files and we could get eight partitions after loading. And we could repartition it to 16 or 32 according to the number of the allocated CPU cores(8 CPU cores).

2.1.3 Number of rows in each dataset

Table 3 Row counts of each dataset

Data set	# of rows
audio	
audio/features	
msd-jmir-area-of-moments-all-v1.0	994623
msd-jmir-lpc-all-v1.0	994623
msd-jmir-methods-of-moments-all-v1.0	994623
msd-jmir-mfcc-all-v1.0	994623
msd-jmir-spectral-all-all-v1.0	994623
msd-jmir-spectral-derivatives-all-all-v1.0	994623
msd-marsyas-timbral-v1.0	995001
msd-mvd-v1.0	994188
msd-rh-v1.0	994188
msd-rp-v1.0	994188
msd-ssd-v1.0	994188
msd-trh-v1.0	994188
msd-tssd-v1.0	994188
audio/attributes	

msd-jmir-area-of-moments-all-v1.0	21
msd-jmir-lpc-all-v1.0	21
msd-jmir-methods-of-moments-all-v1.0	11
msd-jmir-mfcc-all-v1.0	27
msd-jmir-spectral-all-all-v1.0	17
msd-jmir-spectral-derivatives-all-all-v1.0	17
msd-marsyas-timbral-v1.0	125
msd-mvd-v1.0	421
msd-rh-v1.0	61
msd-rp-v1.0	1441
msd-ssd-v1.0	169
msd-trh-v1.0	421
msd-tssd-v1.0	1177
audio/statistics	
sample_properties	992866
genre	
msd-MAGD-genreAssignment	422714
msd-MASD-styleAssignment	273936
msd-topMAGD-genreAssignment	406427
main/summary	
analysis	1000000(the first row is column names)
metadata	1000000(the first row is column names)
tasteprofile	
tasteprofile/mismatches	
sid_matches_manually_accepted	938
sid_mismatches	19094
tasteprofile/triplets.stv	48373586

According to the metadata, there are 1,000,000 unique songs in total. For all the datasets under the audio features branch, the numbers of rows are all larger than 994,000, which means that over 99.4% songs have their audio features correspondingly.

2.2 Data Pre-processing

2.2.1 Taste Profile data cleaning

There are 19094 records in the mismatched tasteprofile associated with 18972 unique songs. By adding the manually accepted matches, the number of truly mismatched songs is 18912.

48,373,586 user- song -play count triplets are saved in the triplets.stv file. We used left_anti joining to remove the truly mismatched songs , then got a clean triplet set. 45,795,111 triplets are preserved.

2.2.2 Audio Features loading

There are 13 feature sets, and the feature names in attribute files are coincident. I tried to add different prefixes for features from different attribute files respectively so that we could distinguish them after joining (if necessary).

Additionally, the count of rows in each feature attribute set is always one larger than the dimensions given by the documents. By checking the schema of each feature dataset, we could find that theTRACK_ID was attached at the end. But the last column names varied in these feature sets, such as MSD_TRACKID, track_id or instanceName. Besides, the length of values of the last column are not of the same. We renamed the last column and made them have consistent format, which would be convenient for further analysis.

3 Audio similarity

In the beginning, we increased the resources up to 4 executors, 2 cores per executor, 4 GB of executor memory, and 4 GB of master memory.

3.1 Exploratory data analysis

There are multiple audio feature datasets, and the number of columns and rows for each feature sets are show below:

Table 4 Feature sizes

feature name	DF name	#of cols	# of rows	feature dimensions
Rhythm Patterns	RP	1441	994188	1440
Statistical Spectrum Descriptors	SSD	169	994188	168
Rhythm Histograms	RH	61	994188	60
Temporal Statistical Spectrum Descriptors	TSSD	1177	994188	1176
Temporal Rhythm Histograms	TRH	421	994188	420
MVD	MVD	421	994188	420
MARSYAS timbral features	MT	125	995001	124
Low-level features	JS	17	994623	16
Low-level features derivatives	JSD	17	994623	16
Method of Moments	**JMM**	**11**	**994623**	**10**
Area of Moments	JAM	21	994623	20
Linear Predictive Coding	PLC	21	994623	20
MFCC features	MFCC	27	994623	26

Method of Moments (msd-jmir-methods-of-moments-all-v1.0) has the least dimensions(10) with smaller size(35.8 M, Table1), so we picked it to get started.

3.1.1 Method of Moments exploration

994,623 observations are in the Method of Moments dataset with 11 columns. Except the TRACK_ID, there are 110 features left. The descriptive statistics for each feature are shown in Figure 2.

	count	mean	stddev	min	max
JMM_Method_of_Moments_Overall_Standard_Deviation_1	994623	0.15498176001746347	0.06646213086143025	0.0	0.959
JMM_Method_of_Moments_Overall_Standard_Deviation_2	994623	10.384550576952284	3.868001393874676	0.0	55.42
JMM_Method_of_Moments_Overall_Standard_Deviation_3	994623	526.8139724398092	180.43775499775265	0.0	2919.0
JMM_Method_of_Moments_Overall_Standard_Deviation_4	994623	35071.97543290272	12806.816272955577	0.0	407100.0
JMM_Method_of_Moments_Overall_Standard_Deviation_5	994623	5297870.369577217	2089356.4364558004	0.0	4.657E7
JMM_Method_of_Moments_Overall_Average_1	994623	0.3508444432531317	0.18557956834383818	0.0	2.647
JMM_Method_of_Moments_Overall_Average_2	994623	27.46386798784065	8.352648595163757	0.0	117.0
JMM_Method_of_Moments_Overall_Average_3	994623	1495.8091812075531	505.8937639190221	0.0	5834.0
JMM_Method_of_Moments_Overall_Average_4	994623	143165.46163257837	50494.27617103219	-146300.0	452500.0
JMM_Method_of_Moments_Overall_Average_5	994623	2.396783048473542E7	9307340.29921968	0.0	9.477E7

Figure 2 The overview of the structure of the datasets

There are no missing values, but the ranges of features varied widely. “Distance algorithms like KNN, K-means, and SVM are most affected by the range of features because they use distances between data points to determine their similarity”(Bhandari, Aniruddha.). Therefore,

we should scale our data before employing a distance-based algorithm so that all the features can contribute equally to the result.

The Pearson correlation matrix of numeric variables in Figure 3 shows some features are highly correlated(an absolute correlation coefficient of >0.7). For example, the last two features are strongly correlated with a coefficient of 0.985(3dp)

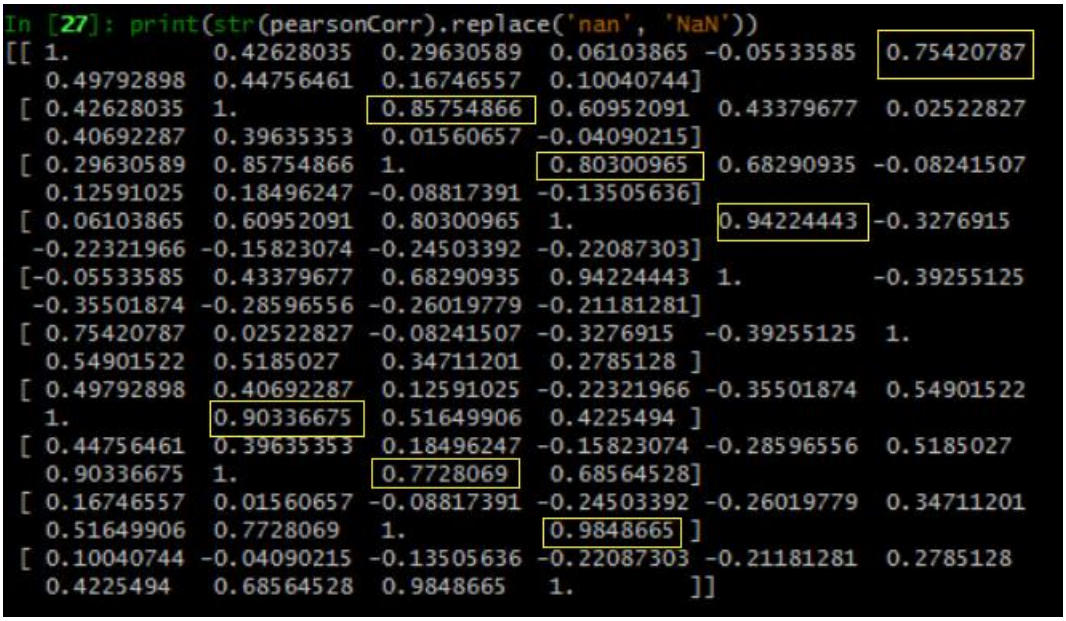

Figure 3 Pearson correlation matrix of features

Generally, “an absolute correlation coefficient of >0.7 between two predictors indicates the presence of collinearity”(RekhaMolala).Potential solutions would be removing some of the collinear independent variables or choosing methods such as PCA or PLS which are analysis methods designed for highly correlated variables and are competent to control the model complexity as well.

3.1.2 MAGD exploration

There are 422,714 rows in the MSD All Music Genre Dataset (MAGD).While after removing the 18912 mismatched songs, 415,350 records are remained. Afterwards, we calculated the counts of songs for each genre and plotted the distribution(Figure 4). There are 21 different genres in MAGD. The Pop_Rock style songs are the most in the data set, and it accounts for more than 56.36%, followed by the electronic and rap respectively. Obviously, classes of genres are imbalanced.

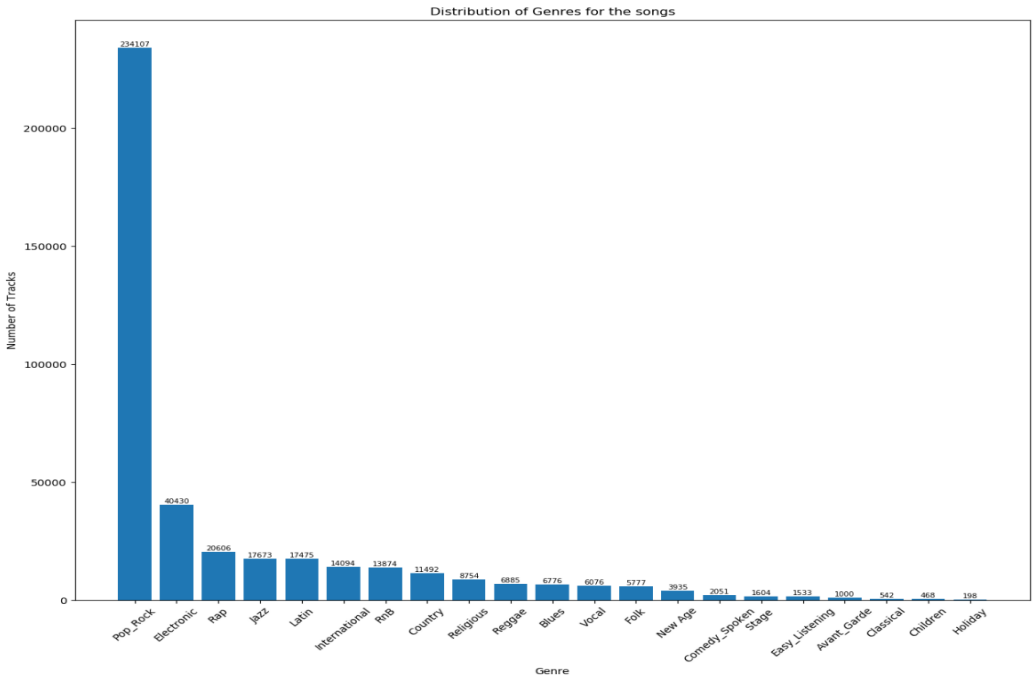

Figure 4 The distribution of genres for the songs

3.1.3 Label the songs

After merging the audio features dataset(JMM) and the genres dataset(MAGD) , only 413,293 labeled songs are left.

3.2 Binary classification model

By converting the genre column into a column named “Class” representing if the song is ”Rap”(1) or not(0).

All the features are numerical, so it is not necessary to encode any categorical features. There are 413,293 instances left after merging. The radio of observation counts to the number of predictors is 41,329.3, which is quite large, so we do not need to worry about overfitting.

However, there are three problems in the merged dataset.

- some features are highly correlated.
- the ranges of values of each feature varied
- the outcome variable is of class imbalance, and the details are shown in Table 5.

Table 5 Unbalanced classification

Class	Count	Ratio(5dp)
1(Rap)	20566	0.04976
0(not Rap)	392727	0.95024

3.2.1 About classification models

As we do not know if the data is linearly separable, so we started with logistic regression as it is more interpretable with high training speed. we could treat it as our baseline. We could also try the Linear SVM Classifier, then turn to non-linear classifier such as random forest or Gradient-boosted trees.

Logistic regression is a predictive analysis algorithm and based on the concept of probability. It is easy to be influenced by extreme values and cannot deal with high intercorrelations issues among the predictors.

Linear SVM Classifier in Spark ML supports binary classification with linear SVM only. A good separation will be achieved by the hyperplane that has the largest distance to the functional margin, so it handles outliers better than logistic regression. But high dimensions mean lower training speed and poor interpretability. Scaling is highly recommended at preprocessing step for SVM.

Random Forest supports both binary and multiclass labels, as well as both continuous and categorical features. When features are on the various scales, Random Forest also works well.

SVM gives us the distance to the boundary while Random Forest tell us the probability of belonging to class .Hence, Random Forest is intrinsically suited for multiclass problems, while SVM is intrinsically two-class.

As Random Forest is an example of a general-purpose method and the performance is always not bad. I would like to use a similar classifier. Gradient-Boosted Trees are ensembles of decision trees and it iteratively train decision trees in order to minimize a loss function.

3.2.2 Split the dataset and train the models

As this is an unbalanced classification problem, there are several methods, such as sampling-based approaches, choosing appropriate metrics, using cost-function based models, or finding the optimal threshold, we could do to mitigate this issue.

Firstly, besides the random splitting the training and testing data, we tried four sampling methods (stratification sampling, down sampling, up sampling, Observation weighting) and compare the performance based on the Logistic Regression model.

Table 6 rates of observations in each class in training data

Sampling	Positive ratio	Negative ratio	Total number
Random split(no sampling)	0.049893	0.950107	331110
Stratification	0.049759	0.950241	330633
Up sampling	0.344308	0.656579	479135
Down sampling	0.33484	0.66516	**49337**
Observation weighting	0.049893	0.950107	331110

Again, all feature values are continuous numeric in JMM, the only thing we need to do is to remove the intercorrelations when using Logistic regression. PAC is a good choice to this problem since it yields a feature subspace that maximizes the variance along the axes. But it makes sense to standardize the data, especially, if feature values are on different scales. Therefore, we should scale the features before applying PCA although Logistic Regression are not affected by scaling of input data . We fit PCA models on the train data after standardization to avoid leaking the information of test data, then “to standardize test data on training set mean and as well use the eigenvectors generated using the training set”(Aegis, Divyesh).

Additionally, as for how to choose principal components, we used proportions of explained variance. According to the plot(Figure 5) ,We are going to take the first **5** components that capture about 97% of the total variance and the line flattens out gradually.

Component	Explained Variance	Accumulative Variance
1	0.422	0.422
2	0.306	0.728
3	0.149	0.877
4	0.071	0.948
5	0.024	0.972
6	0.015	0.987
7	0.007	0.994
8	0.004	0.998
9	0.002	1
10	0	1

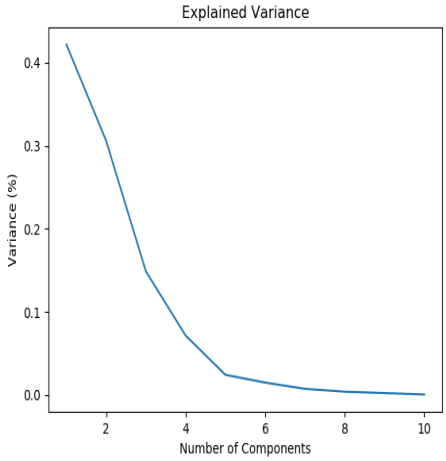

Figure 5 The explained variance of PCA

Focusing on the Logistic regression model, after the preprocessing using standardization and PCA , the evaluation metrics are shown in Table 6.

Table 7 Evaluation metrics comparison(threshold=0.5)

Sampling	precision	recall	accuracy	auroc	F1
Random split(no sampling)	0.1916	0.0180	0.9479	0.8416	0.0329
Stratification	0.1763	0.0141	0.9477	0.8427	0.0261
Up sampling	0.2064	0.6293	0.8627	0.8448	0.3108
Down sampling	0.2103	0.6139	0.8675	0.8449	0.3133
Observation weighting	0.2078	0.6300	0.8635	0.8448	0.3125

Tuning the prediction threshold will change the precision and recall of the model and it is an important part of model optimization. We tried some thresholds around 0.5, and part of the results are shown in Table 8.

Table 8 Evaluation metrics comparison using different threshold

Sampling	precision	recall	accuracy	auroc	threshold	F1
Random split(no sampling)	0.2778	0.1100	0.9421	0.8416	0.3	0.1576
Stratification	0.2788	0.1089	0.9416	0.8427	0.3	0.1566
Up sampling	0.2382	0.4847	0.8983	0.8450	0.6	0.3194
Down sampling	0.2410	0.4684	0.9012	0.8449	0.6	0.3183
Observation weighting	0.2392	0.4822	0.8990	0.8448	0.6	0.3198

Up sampling increases the possibility of overfitting because it replicates the minority class observations. By contrast, down sampling can help increase the training speed and solve

storage issues (49337 instances in training data, versus 331110 instances using random split) by reducing the number of training data samples when the training data set is huge. Of course, it will discard potentially useful information which could be important for building rule classifiers.

"Recall expresses the ability to find all relevant instances in a dataset, precision expresses the proportion of the data points our model says was relevant were truly relevant"(Koehrsen, Will). F1 score is a tradeoff between recall and precision as it combines them using the harmonic mean. By comparing these sampling methods based on Logistic regression, the last three approaches have similar performance. Hence, I would choose down sampling as it picks less observations to save time and space. All the following model training are based on **down sampling** and the threshold is fixed at **0.5** for model comparison.

3.2.3 Performance metrics

After down sampling the audio features dataset(JMM), there are 49337 rows in training data and 82183 observations are used for testing.4046 songs are actual positive (Rap) and 78137 ones are actual negative(not Rap).

In section 3.2.2.1, we used PCA to remove the intercorrelations while LogisticRegression() in pyspark.ml package has the parameter elasticNetParam ,which regularizes or shrinks the coefficient estimates towards zero to avoid the risk of overfitting and solve the collinearity issue. By using the default parameters , the performance metrics of the four models based on testing data from audio feature of methods-of-moments are showed below:

Table 9 Evaluation metrics comparison : default parameters

model	precision	recall	accuracy	auroc	time	F1
LogisticRegression	0.2234	0.6315	0.8738	0.8584	19s	**0.3300**
LinearSVC	0.2104	0.6557	0.8619	0.8500	36s	0.3186
RandomForest	0.2071	0.6614	0.8585	0.8537	5s	0.3154
Gradient-boosted trees	0.2121	0.6851	0.8592	0.8689	25s	0.3239

Time: including model fitting and testing.

LogisticRegression has the best performance but it is a bit time-consuming. Random Forest is the fastest models but its F1 score is the lowest. It is because Random forests are built on decision trees, and decision trees are sensitive to class imbalance.

We got the Table 9 by using the default parameters. However, many models have important parameters which cannot be directly estimated from the data. For example, numTrees and maxDepth are main for Random Forest. Increasing numTrees will decrease the variance in predictions but training time increases roughly linearly ;Increasing maxDepth makes the model more expressive and powerful. However, deep trees take longer to train.

It is essential to tune the hyperparameters as they "directly control the behavior of the training algorithm and have a significant impact on the performance of the model"(Wu, Jia).

Besides, cross validation is a useful technique to test the effectiveness of a machine learning models, and it is also a re-sampling procedure used to evaluate a model if we have a limited data(Sanjay.M).

Accordingly, The next step is to tune the critical hyperparameters by grid method and to use 10-folds cross-validation to make sure that the performance is stable.

Table 10 cross-validation performance evaluator=areaUnderROC

model	precision	recall	accuracy	auroc	F1
LogisticRegression	0.2234	0.6315	0.8738	0.8584	0.3300
LinearSVC	0.2151	0.6426	0.8670	0.8520	0.3223
RandomForest	0.2165	0.6846	0.8625	0.8690	0.3290
Gradient-boosted trees	0.2155	0.6903	0.8611	0.8717	0.3285

Hyperparameters optimization improve the performance metrics of the last three models but it hasn't changed the results of Logistic regression model.

3.3 Multiclass classification

Some algorithms like logistic regression and SVM are designed for binary classification at the beginning, So they cannot be used for multi-class classification problems directly. But heuristic methods could convert the multi-class(K) classification problem into multiple binary classification tasks. One-vs-all takes one class as positive and rest all as negative, and it trains

K classifiers. The class with highest probability among the K classifiers is selected. Unlike one-vs- all that splits it into one binary dataset for each class, the one-vs-one approach splits the problem into K(K-1)/2 binary classifications (Sanjay.M). Each binary model may predict one class and the observation is then classified according to a majority vote amongst the models. One vs all trains less classifiers and it is faster, and it is commonly used. One vs one is less prone to imbalance. In this case, the classes are terribly imbalanced, so maybe the one-vs-all strategy is better for finding the Pop_Rock songs while one-vs-one is optimal for other genres.

Linear SVC in pyspark.**ml** package supports binary classification only. Gradient-boosted trees has similar performance as Random forest, but it is time-consuming. So, I am planning to select multinomial Logistic regression and Random forest for multiclass classification.

StringIndexer() could maps a string column of labels to an ML column of label indices, so each genre has a corresponding integer. We split the dataset into training data(80%) and testing data(20%) randomly. The testing results is shown in the Table 11.

Table 11 performance metrics using ml package

	model	F1	accuracy	weighted precision	weighted recall
Default parameters	LogisticRegression	0.4524	0.5741	0.4098	0.5741
	RandomForest	0.4274	0.5716	0.3736	0.5716
Cross validation	LogisticRegression	0.4524	0.5741	0.4098	0.5741
	RandomForest	0.4536	0.5804	0.4018	0.5804

Accuracy and weighted recall are the same in multiclass classification

As the classes of genres are imbalanced, I also tried to down sample the Pop_Rock genres, but I have not yet reached any good performance. The accuracy for multiclass classification is lower than that for Binary classification, but the F1 score increases.

In order to get the performance metrics for each genre, I turned to the pyspark**.mllib** package and metrics a bit different(Table 12). Besides, the metrics for each class are shown in Table 13. The model is useless for the last 19 classes because they have less observations. In general, the performances of our models go worse by the inclusion of multiple genres.

Table 12 performance metrics using ml package

model	weighted F1	accuracy	weighted precision	weighted recall
LogisticRegressionWithSGD	0.0178	0.0964	0.3291	0.0964
Random Forest	0.4233	0.5698	0.3707	0.5698

Table 13 performance metrics for each class using mllib package

Class	LogisticRegressionWithSGD			Random Forest		
	F1	precision	recall	F1	precision	recall
Class0	0.0017	0.5652	0.0008	0.7252	0.5710	0.9937
Class1	0.1752	0.0960	0.9985	0.1340	0.4955	0.0775
Class2	0	0	0	0	0	0
…	0	0	0	0	0	0
Class20	0	0	0	0	0	0

Except down sampling , over sampling like SMOTE or ADASYN would be another opinion for the imbalanced problems of multiclass classification(due to time limitations of time, I did not finish the trial). Meanwhile, 21 classes are quite a lot and it is practical to coalesce classes to reduce the cardinality, and the novel levels, or should we say, the groups with less observations could be allocated to “other”, which could simplify the issue.

4 Song recommendations

The song recommendation system based on collaborative filtering, which describes algorithms that generate songs recommendations for specific users based on the combined user-song play information(triplets) from all users.

4.1 Taste Profile exploration and filtering

After data cleaning in Section 2.2.1, 45,795,111 user - song - play count triplets are preserved in taste profile set. There are 378,310 unique songs and 1,019,318 unique users. By grouping USER_ID, wo discovered that a user listened 4316 out of 378,310 the unique songs with a percentage of 1.14% , which is the highest among all users. I treated this user as the most active user, and he/she played 5146 times in total.

Focusing on the group statistical result(user activity), we found some stranger users. For example, the user(id=52a6c7b6221f57c89dacbbd06854ca0dc415e9e6) only listened 10 songs but played 2219 times, which is abnormal. We tried two methods to find these unusual users. The first one is to calculate the rate of repeated plays based on user activity, and the second way is to watch the PLAY_COUNT in the taste profile set. If we choose appropriate thresholds for each measure, the counts of unusual users are similar. So, I decided to use the second one which it is easier. We set the threshold at 50(which is adjustable), which means if the user played a single song 50 times, we will treat it as a robot user. Finally, 55507 robot users are filtered out and 963811users remained. Accordingly, only 376343 unique songs are left.

We also grouped SONG_ID and got the popularity of songs. The distribution of song popularity and the distribution of user activity are shown in the Figure 6 and 7.

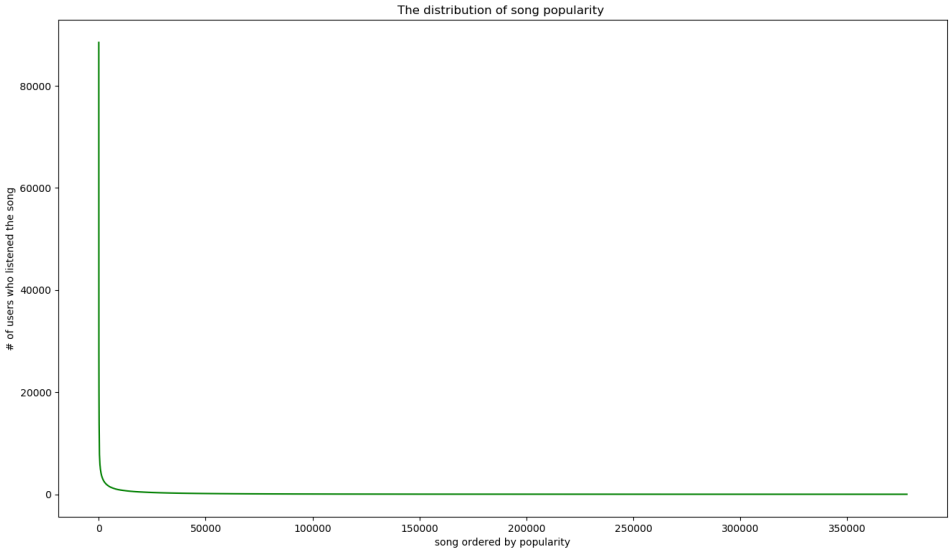

Figure 6 The distribution of song popularity

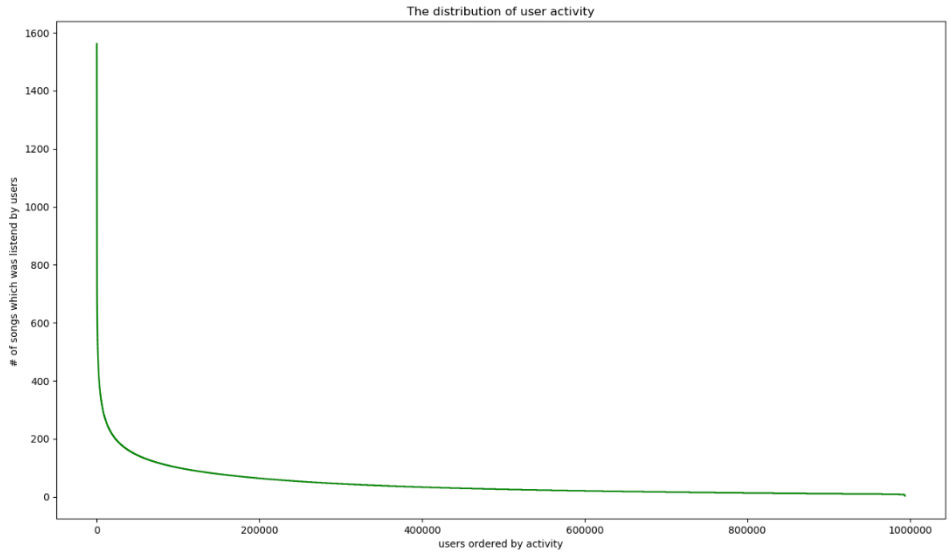

Figure 7 The distribution of user activity

Each figure shows a typical power law distribution with a heavy right tail. Hence, only a small number of unique songs are listened commonly by a few users. Most songs are rarely listened. Accordingly, 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.

For a new user, there is no user - song - play count triplet in the history records about him/her The recommendation engine meets him/her for the first time and doesn't know the personal preferences of this user. To mitigate the user cold start problem in recommender systems, besides profile completion, we would elicit preference by providing general recommendations based on popularity.

According to Pareto Principle, we could assume 80% of meaningful information in the song recommendation system are contributed by the 20% songs and users. Therefore, I filtered out the songs which have been played less than 60 times and the users who have listened to fewer than 70 songs(N=60, M=70). Finally, only 21,721,180 observations are remained with 85913 unique songs and 179891 unique users.

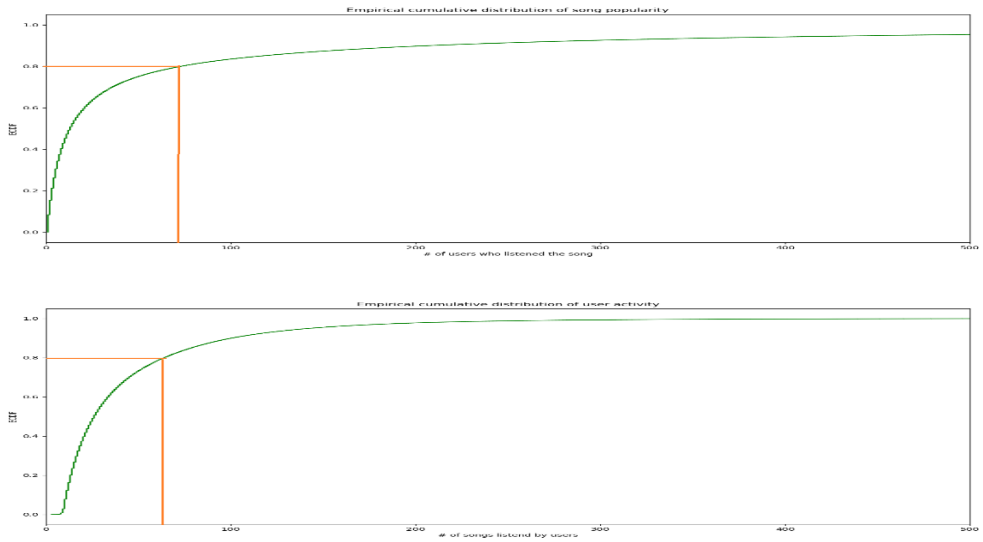

Figure 8 The ECDF of song popularity and user activity

4.2 Split the dataset and train the collaborative filtering model

We spited the user-song plays into training and test sets with the ratio 80:20. 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. If some users in testing set do not have related history in training set, the matrix of user-latent generated by the ALS will have no corresponding vector for these users and the system cannot make recommendations for these users.

To ensure that all users in testing data exist in the training data, we create a column called row_num to making the number of rows in the window which is partitioned by USER_ID and ordered by SONG_ID. So, by using the modulo operator(%) with row_num and 5, the observation with remainder=0 will be the 20% users belonging to the testing data. By doing this, 17,448,754 triplets with 179,891 unique users in the training data and 4,272,426 records with 179,889 unique users in the testing data. All users in testing data exist in the training data. While two users in the training data are not in testing data, because when using M and N to filter the popular songs and active users, the grouped statistical results are changed accordingly, and the two users only listened four songs after filtering.

Alternating Least Squares (ALS) are widely used for matrix factorization. We considered the play counts as the column for *ratings*. It is implicit preference rather than ratings given by users

directly, so we could train an implicit matrix factorization model and set the parameter implicitPrefs=True. StringIndexer() is used to encode USER_ID and SONG_ID with the name *user* and *item* correspondingly.

4.3 Model evaluation and ranking metrics

We selected a few of the users from the test set by hand and used recommendForAllUsers() to generate ten recommendations. Comparing these recommendations to the songs the user has actually played, very few intersected songs occurred and sometimes there is no intersection at all. If I ranked the predicted score for recommendations and compared them with the songs the user has really played in testing data, more intersections appeared.

The RMSE between the actual ratings and the ones predicted by model is 5.62. Some ranking metrics like Precision, NDCG and MAP could be calculated through RankingMetrics(). Precision is the percentage of selected elements that are relevant to the user. It is primarily concerned with being good at finding items without thinking of the orders. It treats all the errors in the recommended list equally. In fact, it is reasonable to weight the errors heavily at the top of the list. MAP gives more weight to errors that happen high up in the recommended lists, but it is not fit for fine-grained numerical ratings. NDCG considers the graded relevance values. But if the ratings are incomplete, setting the missing values to 0 would mark them as irrelevant items. Besides, we need to manually handle the case where the IDCG is equal to zero(Taifi, Moussa).

The crucial part using RankingMetrics() is to generate the predictionAndLabels which combines the recommended items and the ground truth relevant items. We ranked the prediction values(scores) for each user in the predication data and picked the top 10 recommended items(songs) for each user in the testing data. Then we did the similar operation for the raw testing data and got the top 10 ground truth relevant items(Top 10 songs which played the most) for each user. By joining the two results together on user, we could get the predictionAndLabels RDD to compute the ranking metrics. The result is show below:

- PRECISION @ 5: 0.82
- NDCG @10: 0.816
- MAP: 0.487

If we used the recommendForAllUsers() functions to generate the top 10 recommendations for each user, the ranking metrics went worse.

- PRECISION @ 5: 0.040
- NDCG @10: 0.038
- MAP: 0.009

The reasons are explained below:

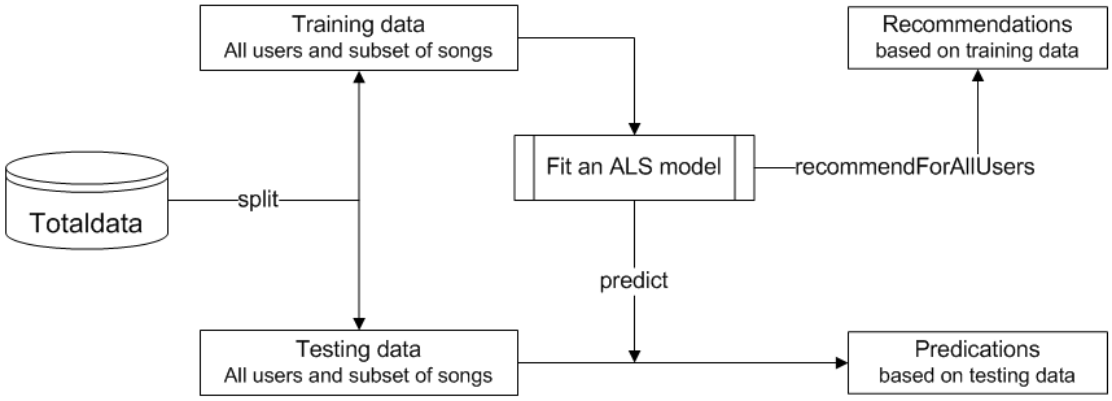

Figure 9 The Flow chart of recommendation

All thought we ensure all the user in the test set has some user-song plays in the training set as well, but we cannot guarantee all the songs in the test set exist in the training set. For the predictions, we are more likely to get top K songs that are truly relevant only focusing on the testing data. For the recommendations, the top K songs are based on the training data. The key point is that songs in the test set are not in train, and there are more songs in the training set than that in testing set in general.

Ranking metrics are used for offline candidate systems comparison based on historical data. However, users' behavior is the final judgement on the systems .The best way to compare two recommendation systems is to test them in the real world(Pandey, Parul). A/B test is a good choice because we can get actual feedback from real users, but that will cost money.

If we could measure future user-song plays based on your recommendations, conversion rate or click-through rate(Wikipedia) would be useful metrics as they could tell us the how actually the users love the songs recommended by our system. "No metric matters more than how real customers react to recommendation you produce in real world"(Rakesh4real). By the way,

financial indicator matters more in the commercial world. For example, the revenue would be a direct metric to evaluate the effectiveness of the system.

4.4 Some discussion

A collaborative filtering model might be possible to get very impressive results but deploying it into real life would be a complicated thing. Some common and important challenges like cold start problem, filter bubbles, outliers, scalability, sparsity, economic profits, etc.

Cold start problem is the most common one and it happens when new users or items joining in. The system cannot predict for new users as there is no history about her preference. The matrix of user-latent generated by the ALS will have no corresponding vector for this user and the system cannot make recommendations for her accordingly. Similarly, as for a new item(song), we cannot predict whom to recommend it to. Content-based models focus on data available about items or users and ignore the interactions between users and items. Hence, its recommended results are very different from those produced by collaborative models. However, they can make predictions for unseen items and usually have a better coverage compared to collaborative models(Höng, Alan).

In addition, the sparsity problem also has negative effect on the effectiveness of recommendations. It is hard to find enough dependable similar users since the active users only rated a small portion of items overall(Guo, Guibing). Learn higher level attributes from items by looking at metadata would mitigate this issue. Hybrid recommender system combine content-based models, collaborative filtering model, and other approaches so that they overcome a lot of the challenges of each individual approach(Höng, Alan). They can deal with new items, new users, and missing features. In this case, artist related features should be used to the system .We could also collect other features about the user and songs. For instance, we could ask the users what they are interested in when they sign up for the website. Given enough features, Hybrid recommenders can perform for returning users as well as new users/items.

Besides the accuracy, diversity is also meaningful for the recommendation system. We could try shuffling the orders in recommendations slightly by taking account of the song types or artists to increase the variety. But before we do that, it is better to understand users' preferences. Some users would prefer to stay in their comfort zone while others will get tired of their favorite

songs when looped playing. It is practicable to collect users' preference when they sign up, or to apperceive their personalities which will influence users' behavior according to their history or other relevant information, like extracting information from social medias to draw the user portrait (be careful about the data ethical issues). If the user values diversity, there would be some space for the improvement of the effectiveness. It is a safe way to swap the items to make the songs played by the same artist be next to each other. By doing this, a song at the bottom of the list would have a change to appear ahead, and this song would be a new genre for the user. In addition, we could also add some new songs to the ranking list and review the user's feedback.

Finally, if the play counts are not actual, for example, a 24 hours convenient store chooses a song as its background music, the implicit rating information is not reliable at all. In this case, the system would produce low quality recommendations. Some statistical analysis or mining algorithms could be applied to find out the fake users and to improve the effectiveness of the system.

Works Cited

Degget. “How Does Spark Partition(Ing) Work on Files in HDFS?” *Stack Overflow*, 1 Dec. 1964, stackoverflow.com/questions/29011574/how-does-spark-partitioning-work-on-files-

Bhandari, Aniruddha. “Feature Scaling: Standardization Vs Normalization.” *Analytics Vidhya*, 14 Apr. 2020, www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization/.

RekhaMolala. “Correlation and Collinearity - How They Can Make or Break a Model.” Medium, Clairvoyant Blog, 23 Mar. 2020, blog.clairvoyantsoft.com/correlation-and-collinearity-how-they-can-make-or-break-a-model-9135fbe6936a.

Aegis, Divyesh. “When to Use PCA before or after a Train-Test Split?” Codementor, www.codementor.io/@divyeshaegis/when-to-use-pca-before-or-after-a-train-test-split-vxdrlu6ci.

Koehrsen, Will. “Beyond Accuracy: Precision and Recall.” *Medium*, Towards Data Science, 10 Mar. 2018, towardsdatascience.com/beyond-accuracy-precision-and-recall-3da06bea9f6c.

Brownlee, Jason. “How to Use One-vs-Rest and One-vs-One for Multi-Class Classification.” *Machine Learning Mastery*, 3 Feb. 2020, machinelearningmastery.com/one-vs-rest-and-one-vs-one-for-multi-class-classification/.

Sanjay.M. “Why and How to Cross Validate a Model?” *Medium*, Towards Data Science, 13 Nov. 2018, towardsdatascience.com/why-and-how-to-cross-validate-a-model-d6424b45261f.

Wu, Jia, et al. "Hyperparameter optimization for machine learning models based on bayesian optimization." Journal of Electronic Science and Technology 17.1 (2019): 26-40.

“Empirical Cumulative Distribution Function (ECDF) in Python.” Python and R Tips, 18 May 2019, cmdlinetips.com/2019/05/empirical-cumulative-distribution-function-ecdf-in-python/.

Taifi, Moussa. “MRR vs MAP vs NDCG: Rank-Aware Evaluation Metrics And When To Use Them.” Medium, The Startup, 29 Nov. 2019, medium.com/swlh/rank-aware-recsys-evaluation-metrics-5191bba16832.

Pandey, Parul. “Recommendation Systems in the Real World.” Medium, Towards Data Science, 25 May 2019, towardsdatascience.com/recommendation-systems-in-the-real-world-51e3948772f3.

Rakesh4real. “Evaluating Recommendation Systems - Part 2.” Medium, Fnplus Club, 30 July 2019, medium.com/fnplus/evaluating-recommender-systems-with-python-code-ae0c370c90be.

Narapareddy, Akhilesh. “How Twitter Does It? Challenges in Implementing

Recommender Systems at Scale.” Medium, Towards Data Science, 22 Sept. 2019, towardsdatascience.com/how-twitter-does-it-challenges-in-implementing-recommender-systems-at-scale-353f7a1ae4ab.

Höng, Alan. “Challenges & Solutions for Production Recommendation Systems.” *Codementor*, www.codementor.io/@alanfhoeng/challenges-solutions-for-production-recommendation-systems-o1bdgsocd.

“Recommender System.” Wikipedia, Wikimedia Foundation, 12 May 2020, en.wikipedia.org/wiki/Recommender_system#Collaborative_filtering.

Guo, Guibing. "Improving the performance of recommender systems by alleviating the data sparsity and cold start problems." Twenty-Third International Joint Conference on Artificial Intelligence. 2013.