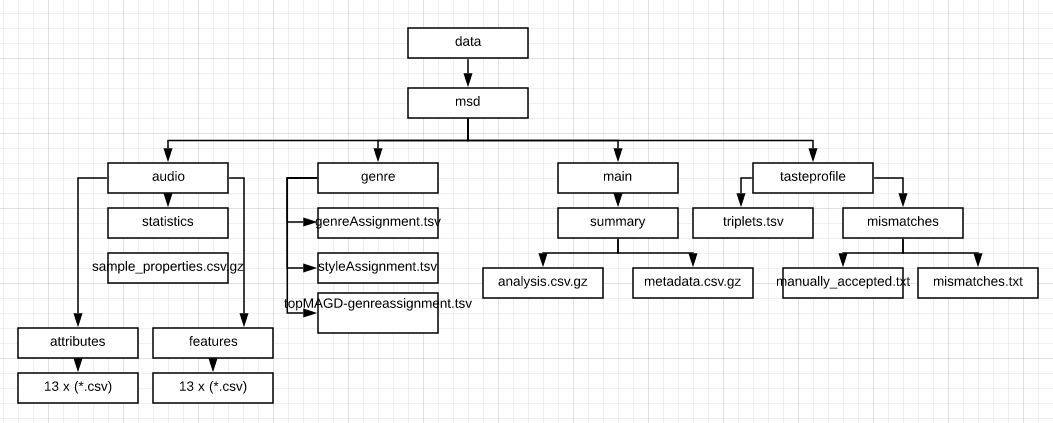
**STAT420\_Assignment 2**

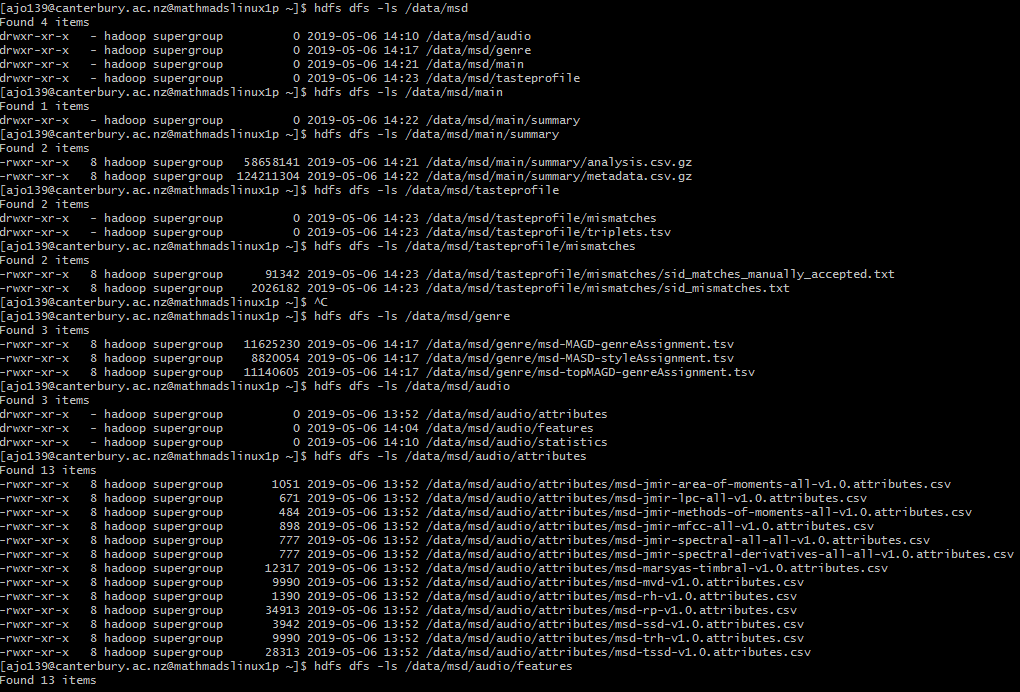
Data processing

Q1 Read through the documentation above and ﬁgure out how to read from each of the datasets. Make sure that you only read a subset of the larger datasets, so that you can develop your code efﬁciently without taking up too much of your time and the cluster resources.

1. Give an overview of the structure of the datasets, including ﬁle formats, data types, and the expected level of parallelism that you can expect to achieve from HDFS.



Million song data set (msd) directory consists of 4 subdirectories and subsequent files in .csv, .csv.gs, .tsv, .tsv.gz, .txt formats. All files were loaded using inferSchema: true option. Using printSchema function, all data types were checked. The data types used are string, integer and double.



Parallelism on spark defaults to the number of cores on all machines (worker nodes/ executors). To increase parallelism of spark processing, we may increase the number of executors on the cluster. Partitioning in Spark is a technique to handle parallelism more efficiently. One partition is created for each block of the file in HDFS of size 64MB. Custom partitioning can be used to optimize performance and achieve parallelism. Cluster configuration and requirements of the application are the deciding factors. Number of partitions in an RDD should be in parallel to the number of cores in the cluster. This will assure parallel processing of all the partitions and the resources will be utilized in an optimal way. (ProjectPro)

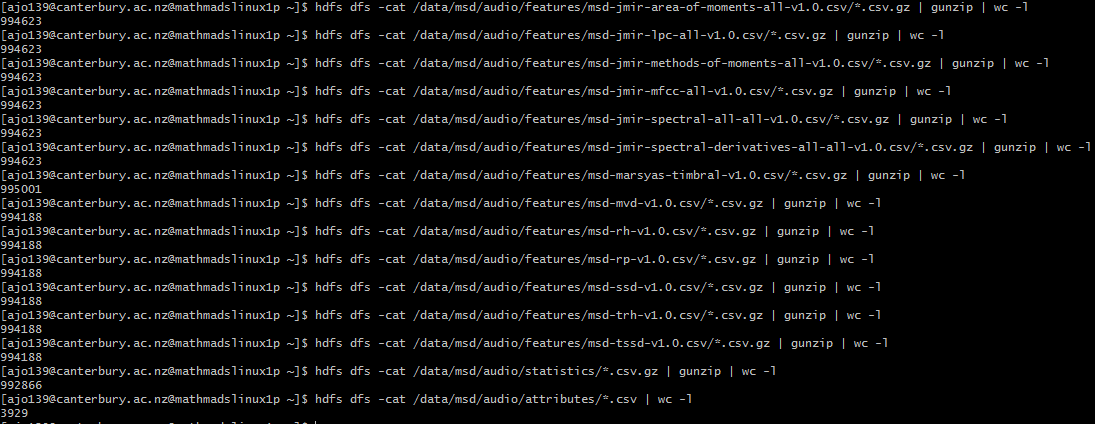
1. Look up the repartition method. Do you think this method will be useful?

Repartition method is used for custom partitioning of the data frame. It can either increase or decrease the number of partitions in a data frame. This method is useful to achieve an optimized performance of the system resources. Yet repartitioning is a fairly expensive full shuffle operation, in which whole data is taken out from existing partitions and equally distributed into newly formed partitions. That is data over all the partitions are equally populated.

If we are only decreasing the number of RDD partitions, spark has a less expensive and faster method, coalesce(), that avoids data movement. Using this method data is not equally distributed, that is partitions size varies.

1. Count the number of rows in each of the datasets. How do the counts compare to the total number of unique songs?

In audio folder, we have the subdirectory attributes, features and statistics. Features consists of 13 other subdirectories. wc -lc command is used to count the number of rows for each of these directories.



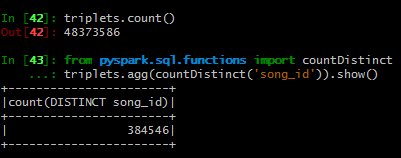
Same command is used to find out the number of rows genre and main directories



Tasteprofile directory consists of two subdirectories; triplets and mismatches. Number of rows are printed as below.



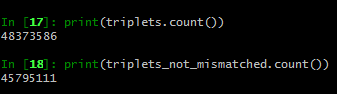
To count distinct song ids, schema is defined for the triplets file and counted total number of unique songs.



Q2 Complete the following data preprocessing.

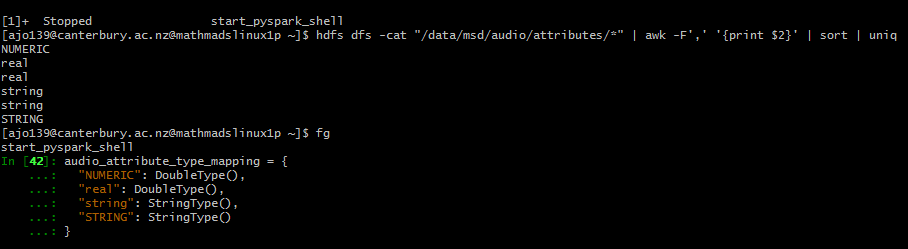
1. Filter the Taste Proﬁle dataset to remove the songs which were mismatched.

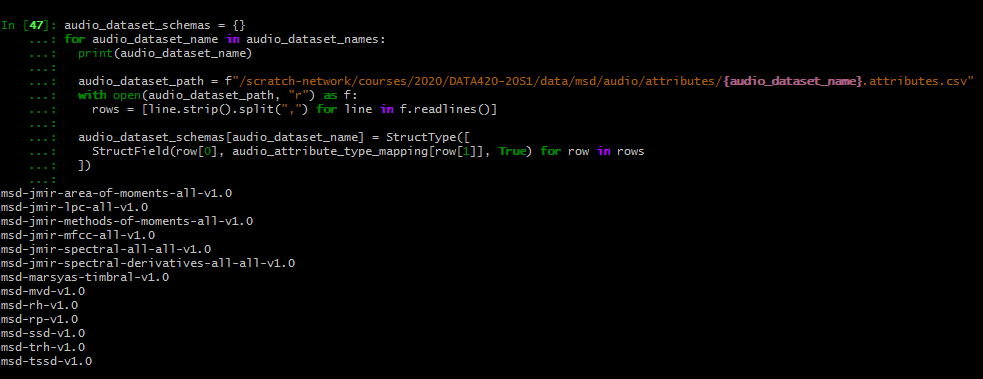
Schemas where defined for the matches manually accepted file and mismatched text file and are removed from the triplet’s data frame that we had defined previously. 2578475 songs were filtered out.

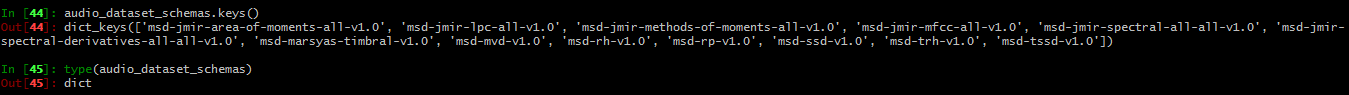


1. Load the audio feature attribute names and types from the audio/attributes directory and use them to deﬁne schemas for the audio features themselves. Note that the attribute ﬁles and feature datasets share the same preﬁx and that the attribute types are named consistently. Think about how you can automate the creation of StructType by mapping attribute types to pyspark.sql.types objects.

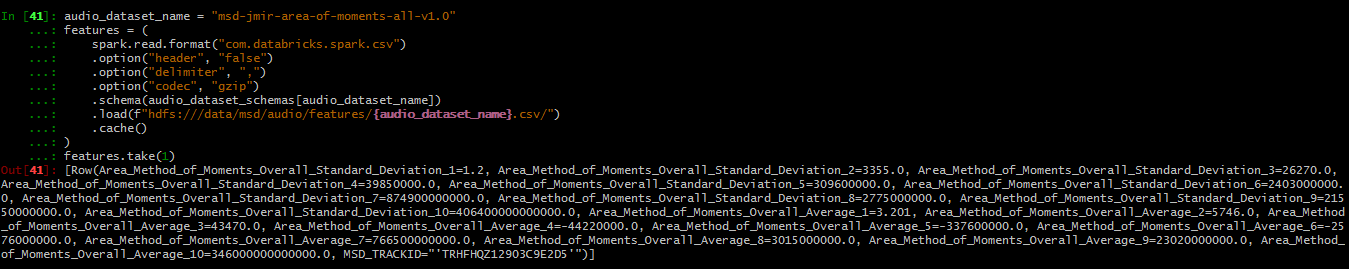
The awk command is used to load the attribute types along with sort and uniq commands. Later we mapped these attribute types to spark data types. We created a list object to hold or those dataset names and later we created a dictionary object to automate the creation of StructType and to define schemas for all sub datasets in the features directory.







Now, each of the data sets can be loaded using this automated schema definition.

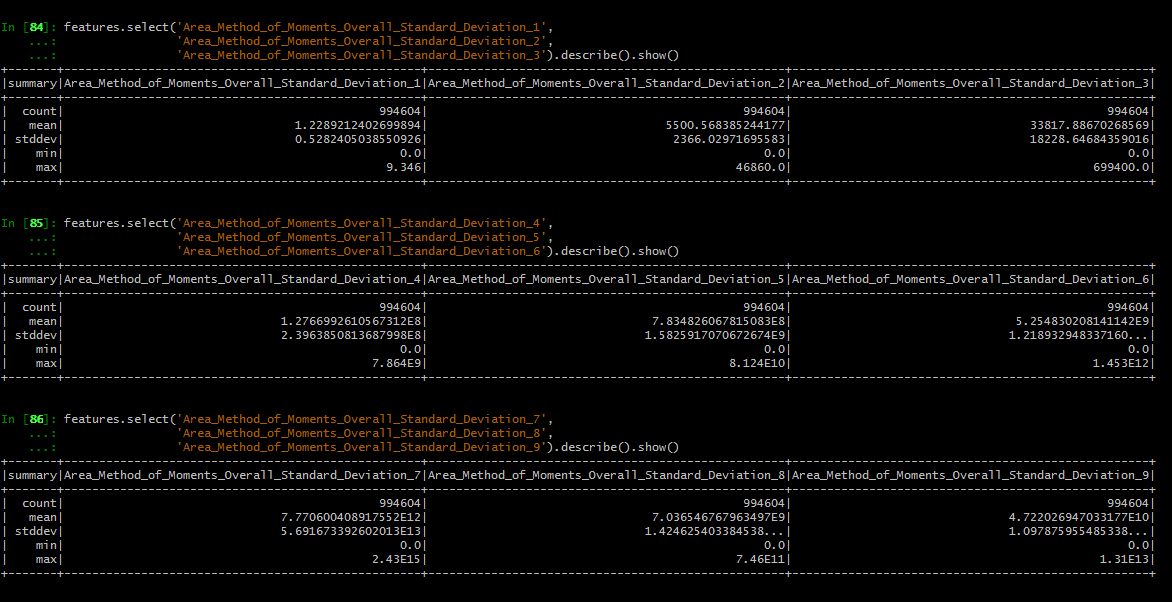


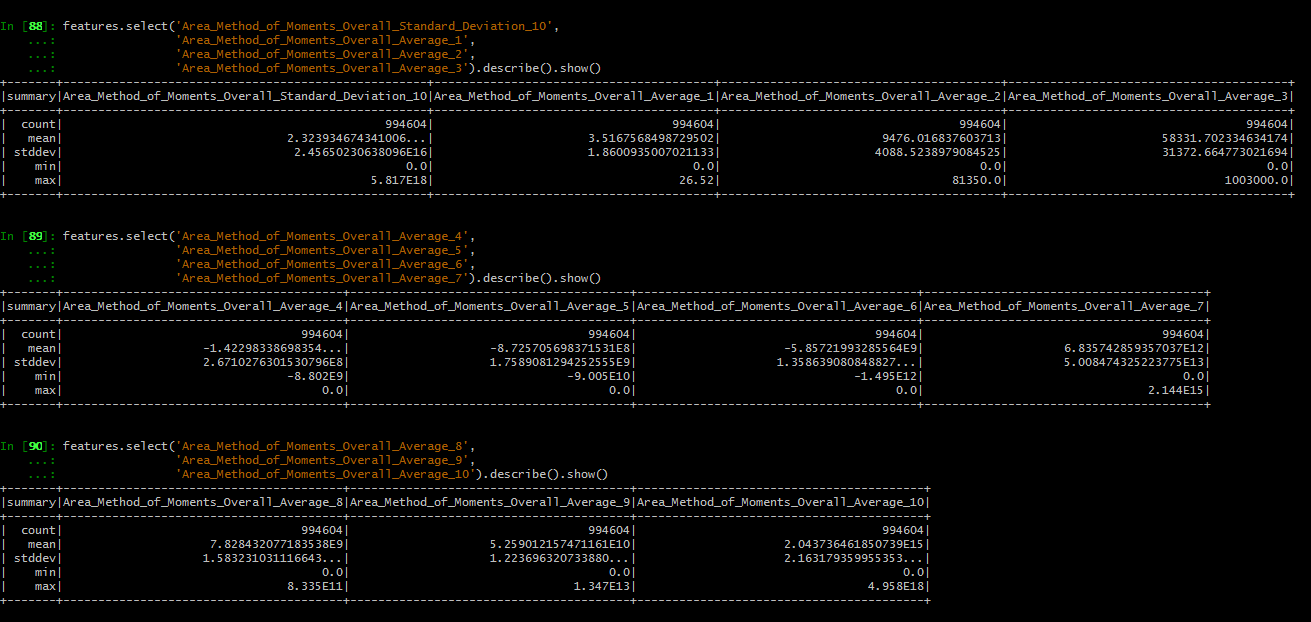
Audio Similarity

Q1 There are multiple audio feature datasets, with different levels of detail. Pick one of the small datasets to get started.

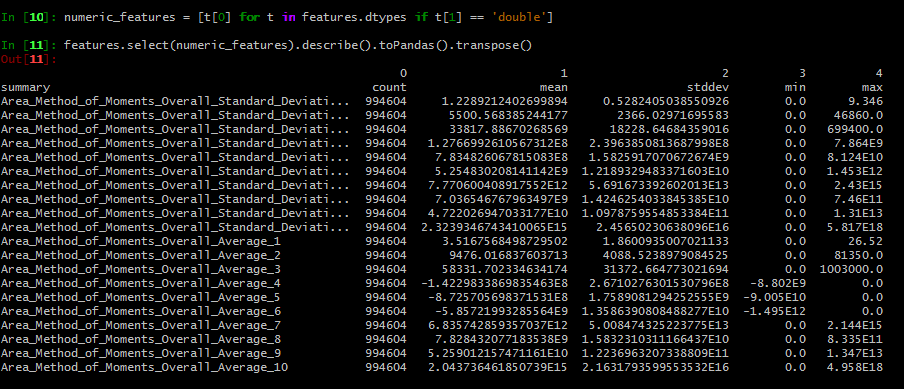
(a) The audio features are continuous values, obtained using methods such as digital signal processing and psycho-acoustic modeling.

Produce descriptive statistics for each feature column in the dataset you picked. Are any features strongly correlated?

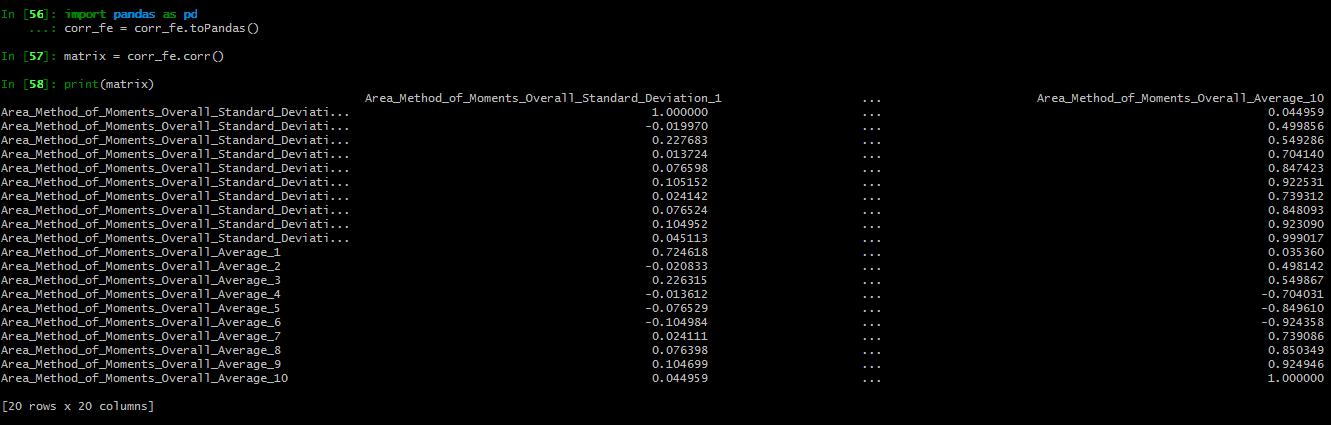




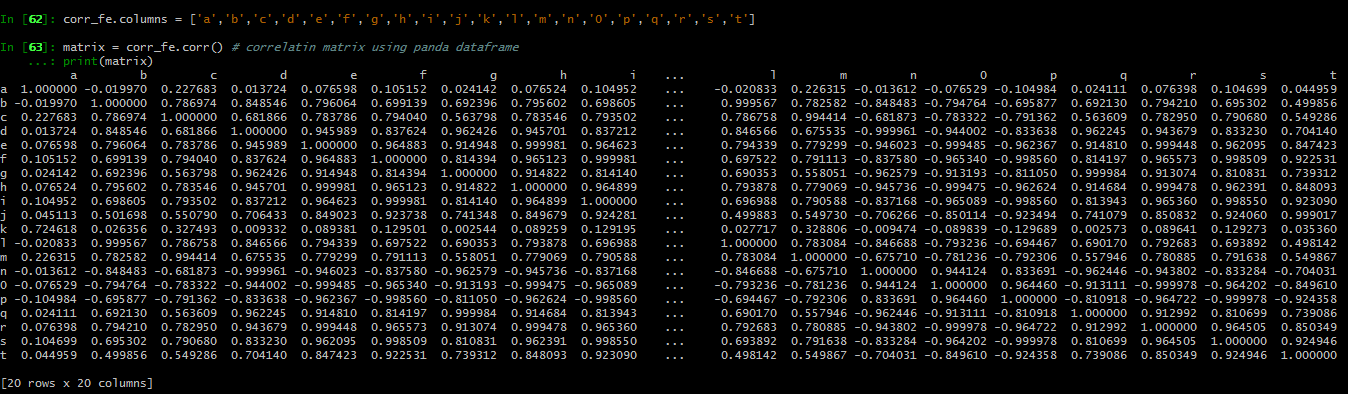
Using panda data frame features:



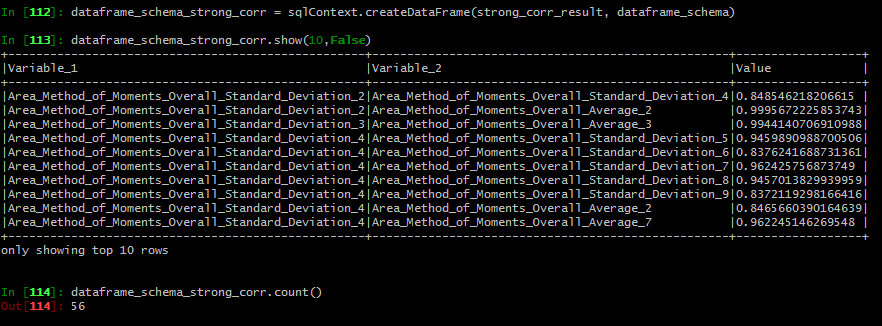
I have tried correlation method in spark, that returned an error. Later I converted spark data frame into a panda data frame to use the corr() in panda.



For a better view, I have changed the column names



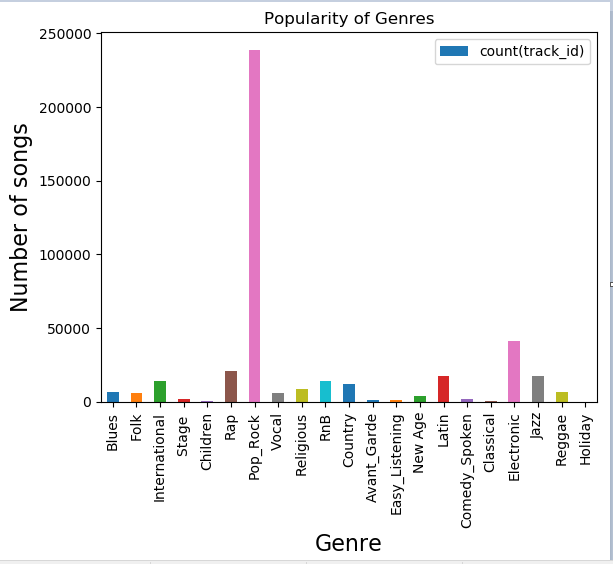
Apparently, many of the features are strongly correlated. A new data frame is created to list strongly correlated variables, using statistics, numpy and panda libraries



(b) Load the MSD All Music Genre Dataset (MAGD).

Visualize the distribution of genres for the songs that were matched.





Genres for the songs that were matched is plotted using a user defined function.

1. Merge the genres dataset and the audio features dataset so that every song has a label.

The features data set is left joined to the genre data set. The quotes on the track\_ id variable is removed.



Q2 First you will develop a binary classiﬁcation model.

(a) Research and choose three classiﬁcation algorithms from the spark.ml library.

Justify your choice of algorithms, taking into account considerations such as explainability, interpretability, predictive accuracy, training speed, hyperparameter tuning, dimensionality, and issues with scaling.

Based on the descriptive statistics from Q1 part (a), decide what processing you should apply to the audio features before using them to train each model.

Classification is for dividing items into categories. I have chosen the Linear SVMs, Gradient-boosted tree classifier, and Logistic regression classification models for msd classification problem. Evaluation of the problem using versatile class of algorithms is a suggested step. Nature of each data set is different and suits differently for each base learner.

Logistic regression well appropriates for binary classification model. It is an extension of the linear regression model for classification problems. This is the most popular machine learning algorithm used for classification problems. Logistic is a probabilistic binary linear classifier

Linear support vector machines (Linear SVMs) is a standard method for large scale classification tasks. It is a linear method that supports only binary classification on spark. Linear SVMs are trained with L2 regularization by default, while spark.mlib supports L1 regularization alternatively. Regularizer encourage simple model and avoids overfitting. It has a meaningful stopping criterion that efficiently handles training error. (Kecman). Linear SVMs are non-probabilistic binary linear classifier.

* Linear SVM is linearly scalable (scales linearly with the size of the training data set)
* Efficient in dealing with extra-large data set
* It works for a high dimensional data in both sparse and dense format
* Extremely fast machine learning algorithm
* Superior performance when high accuracy is required
* Simple and easy to implement

Gradient Boosted Trees (GBTs) are ensembles of decision trees in order to minimize a loss function. Spark.mllib supports GBTs for binary classification and regression only. (Spark)

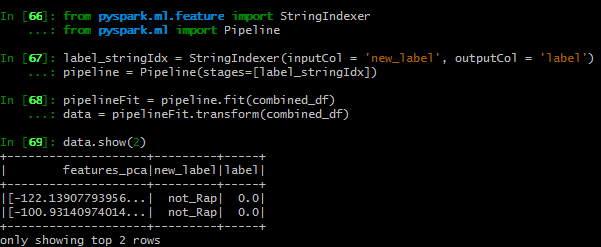
* GBTs do not require feature scaling
* Able to capture non-linearities and feature interactions
* Tuning parameters include loss, numIterations, learningRate, algo etc.
* runWithValidation method to prevent overfitting
* GBTs can be used in the field of learning to rank
* One of the most powerful technique for building predictive models

**Data preprocessing**

In machine learning, it is critical to determine the optimal features that are used to train the model. Feature engineering involves turning the features (inputs of interest) into n-dimensional feature vectors. (Kabii)

All features in our data are continuous, except the track\_id. Since we do not have any categorical predictors, no need to deal with feature extraction. Sparse vectors (large number of zeroes) and the curse of dimensionality (more training data for more features to avoid overfitting) can be dealt with feature reduction.

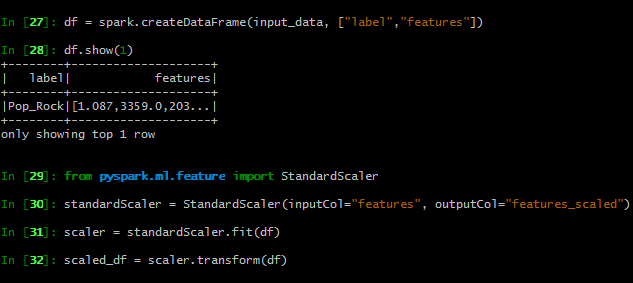
We need to do category indexing for the response variable anyways and fitted using pipeline.

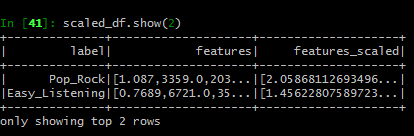


First, we leave out variables that we are not interested in for our model. Thus, we will remove the track\_id column from our data. We also remove genre columns with null values. I used na.drop() function to check for any missing values. It returned the same count that matched the previous count (with all values). This means no missing values in the data. Thus we do not have to deal with the imputation of the missing data either.

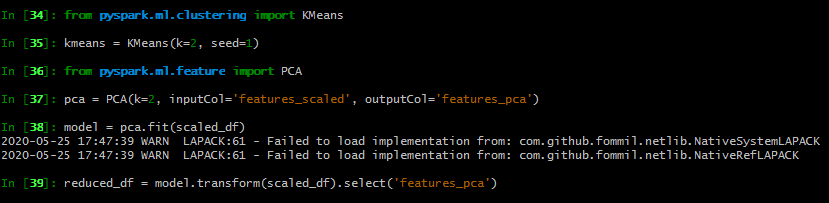
From the descriptive statistics in of the audio features data, we see attributes have a wide range of values, that needs to be normalized. Spark ML library makes it easy to do machine learning on big data scalable. We need to separate features from the target variable. We use map () and DenseVector() function to do this. A dense vector is used to store arrays of values for use in pyspark. We create a new data frame out of the input data and re-label the columns as ‘label’ and ‘features’. Here we will pass a list as the second argument (Willems). Now we may use StandardScaler from ml library to standardize our features.

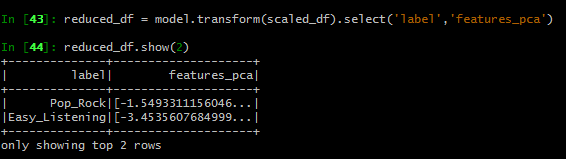






Since we found out strong correlations between the predictor variables, we will apply feature reduction using PCA (k=2).

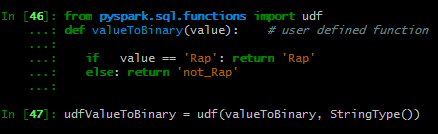


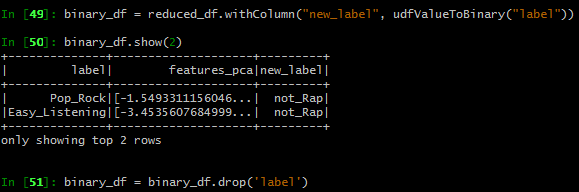


(b) Convert the genre column into a column representing if the song is ”Rap” or some other genre as a binary label.

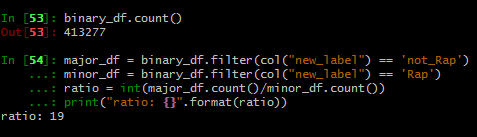
What is the class balance of the binary label?

I used a user defined function to generate a new binary label column.





If the classes are not distributed equally in the data set, there is class imbalance. Only 4% of the songs belongs to genre Rap and 96% belongs to the other genres. The ratio of the class imbalance is 19. That means for every Rap entry there are 19 not\_Rap entries.

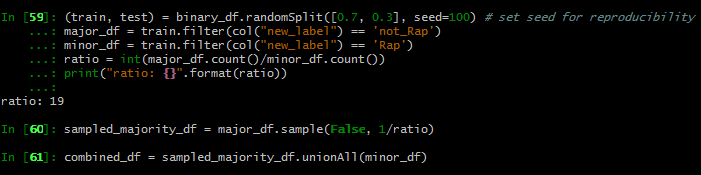


1. Split the dataset into training and test sets. Note that you may need to take class balance into account using a sampling method such as stratiﬁcation, subsampling, or oversampling. Justify your choice of sampling method.

Class imbalance can be addressed in a number of ways such as adding class weights, changing algorithms, random resampling etc.

Random resampling strategies are effective solutions to obtain a more balanced data distribution. I will consider under sampling with such large amount of data. Random deletion of examples in the majority class is called random under sampling. Resampling is not applied to the test dataset.

Dataset is split into train and test datasets. Now we apply under sampling on the train data. We used sample () method in spark to do this. First split the train data according to the class label. Delete rows of the majority class by the value of ratio. And then combined it with the minority class to form the new data frame.

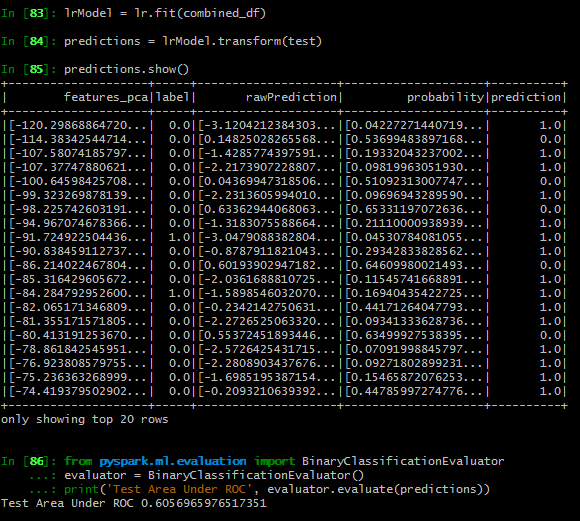


Under sampling reduces overall size of the training data (Wan). Now ratio between the two class labels is 1.



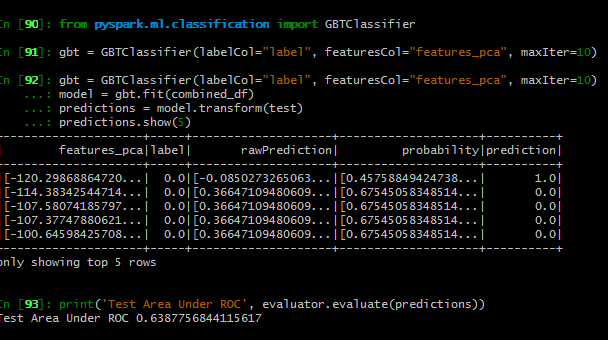
1. Train each of the three classiﬁcation algorithms that you chose in part (a).

Logistic regression model



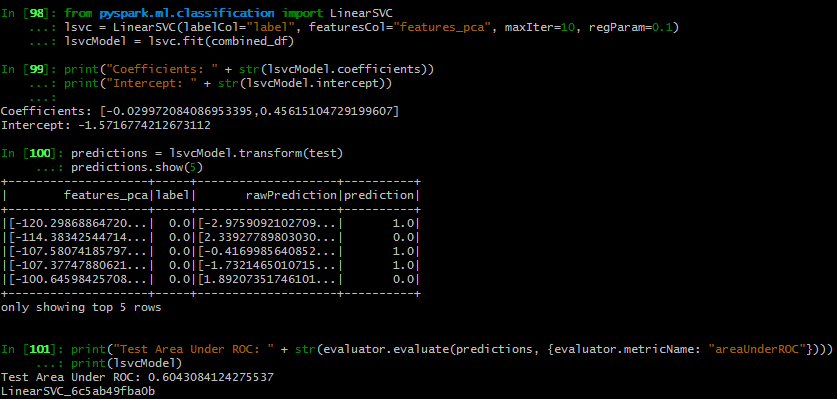
Area under ROC (AUC) for the logistic regression model is 60.56%

Gradient boosted tree classifier



Area under ROC (AUC) for the Gradient Boosted Tree classifier model is 63.87%

Linear support vector machine

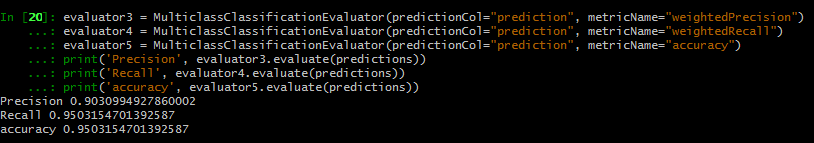


Area under ROC (AUC) for the Linear Support Vector Machine model is 60.43%

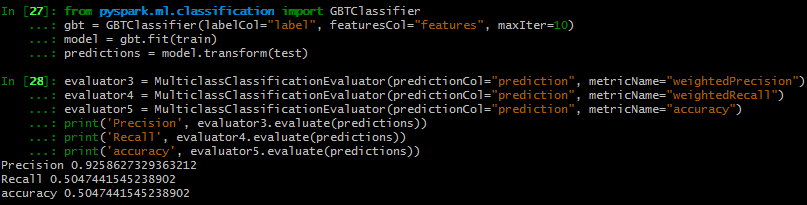
1. Use the test set to compute a range of performance metrics for each model, such as precision, accuracy, and recall.

Multiclass Classification Evaluator is imported to get the performance metrics for the models. Binary classification evaluator gives on the metrics, Area under ROC (AUC) and Area under PR.

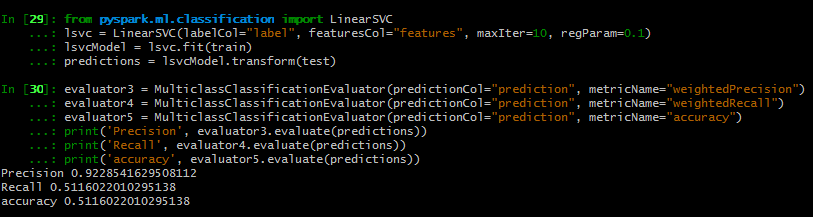
Logistic regression



Gradient boosted tree classifier



Linear support vector machines



(f) Discuss the relative performance of each model and of the classiﬁcation algorithms overall, taking into account the performance metrics that you computed in part (e).

How does the class balance of the binary label affect the performance of the algorithms?

In terms of accuracy, logistic regression performs the best, with an accuracy of 95%. If we consider area under ROC, all three models performed average. Class imbalance of the model will lead to a biased result. A well-balanced class label can generate an unbiased result and better performance of the model.

Q3 Now you will tune the hyperparameters of your binary classiﬁcation model.

1. Look up the hyperparameters for each of your classiﬁcation algorithms. Try to understand how these hyperparameters affect the performance of each model and if the values you used in Q2 part (d) were sensible.

Hyperparameters in machine learning is the term used to distinguish from the standard model parameters. A mathematical formula with a number of parameters that is used to learn from the data set is a machine learning model by definition. This is done through the process model training. In this process we fit the model parameters with the training data.

On the other hand, hyperparameters express high-level properties of the model that cannot be directly learned from the regular training process. These parameters are used to generate optimize results from the data. High-level properties imply the complexity, speed of performance, capacity to learn etc. These parameters need to be predefined and are decided by training different models and setting different values to generate the best result.

Logistic regression model:

1. maxIter: maximum number of iterations to run the optimization algorithm, >=0(default =100)
2. regParam: Regularization parameter, >=0 (default = 0)
3. elasticNetParam: A combination of L1 and L2 regularization in range [0,1] (default = 0)
4. aggregationDepth: suggested depth for tree aggregate, >=2 (default = 2)
5. fitIntercept: whether to fit an intercept term (default = True)
6. family: Description of the label distribution to be used in the model (default = auto) (Rai)

In our trained model we used, maxIter 10, regParam 0.3 and elasticNetParam 0.8 which are sensible.

Gradient boosted tree classifier:

1. maxDepth: maximum depth of a tree used to control overfitting
2. maxBins: maximum number of bins created by ordered splits
3. maxIter: maximum number of iterations

The trained model used the maxIter 10.

Linear support vector machine:

1. maxIter: maximum number of iterations to run the optimization algorithm, >=0(default =100)
2. regParam: Regularization parameter, >=0 (default = 0)

The trained model used the maxIter 10 and regParam 0.1.

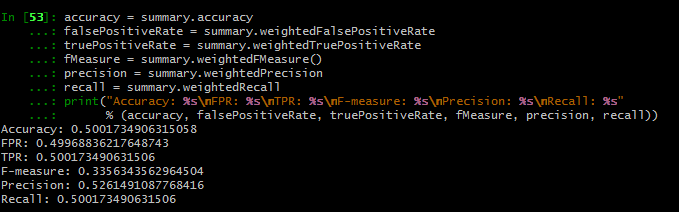
(b) Use cross-validation to tune some of the hyperparameters of your best performing binary classiﬁcation model.

How has this changed your performance metrics?

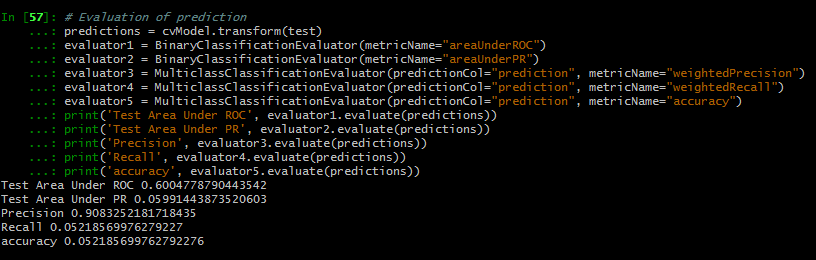
To my surprise my cross-validation model does not show any improvements in the performance. I have tried changing the paramGrid values. But it does not get any better.

To speed up the process, I have enabled parallelism. This argument is set to 8 (number of cores used). The total process took 50ms. 8 jobs in parallell and 10 tasks per each job.

Metrics from the model summary:

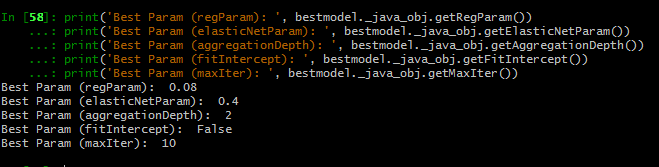


Model predictions:



Area under ROC (AUC) for the cross validation model is 60%

Hypertune parameters for the best model:



Q4 Next you will extend your work above to predict across all genres .

1. Look up and explain the difference between the one vs. one and the one vs. all multiclass classiﬁcation strategies. Do you expect one of these strategies to perform better than the other in general?

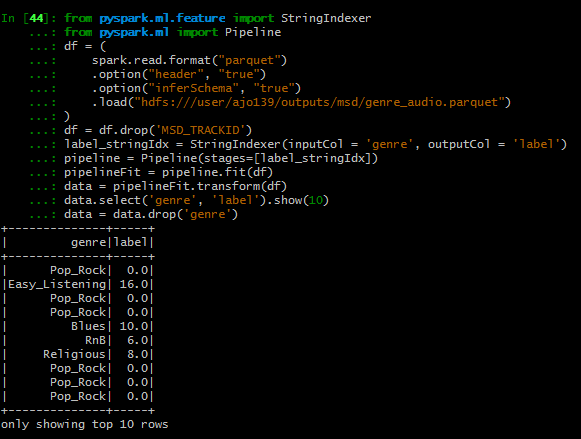
Some classification algorithms support multi-class classification. one-vs-one and one-vs-rest are two different strategies that shares a common approach for the multi-classification problems. A multi-class classification dataset is split into multiple binary classification dataset and fits a binary classification model on each.

One-vs-one strategy splits a multi-class classification into one binary classification problem per class. This is a suggested method for support vector machines. Where as one-vs-rest strategy splits a multi-class classification into one binary classification problem per each pair of class. Eg. Logistic regression, Perceptron

1. Choose one of your algorithms from Q2 that is capable of multiclass classiﬁcation and performed well for binary classiﬁcation as well.

Spark.ml does not support multi- class classification for the Gradient Boosted Tree (GBT) classifier and Linear SVMs. Logistic regression, that uses the one-vs-rest strategy can be used for multiclass classification.

1. Convert the genre column into an integer index that encodes each genre consistently. Find a way to do this that requires the least amount of work by hand.



StringIndexer from the pyspark.ml library is used for index encoding for the genre column

(d) Split your dataset into training and test sets, train the classiﬁcation algorithm that you chose in part (b), and compute a range of performance metrics that are relevant to multiclass classiﬁcation. Make sure you take into account the class balance in your comments.

How has the performance of your model been affected by the inclusion of multiple genres?

First we split the data into train and test using pyspark.sql.window. Class imbalance of a multi-class classification problem can be handled in multiple ways.

1. Balance the training set

Oversample the minority class

Under sample the majority class

Synthesize new minority classes

1. Throw away minority classes and switch to an anomaly detection framework
2. Different strategies at the algorithm levels

Adjust the class weights

Adjust the decision threshold

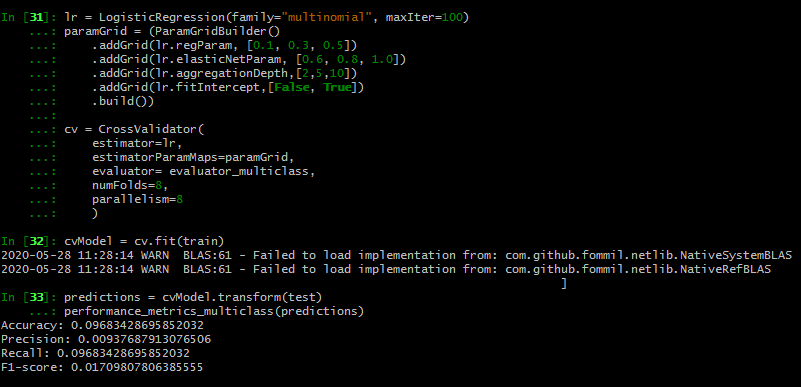
Modifying an existing algorithm

Construct an entirely new algorithm (Faucett)

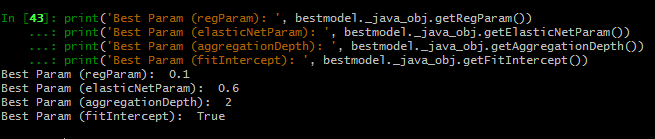
In our problem, I would consider under sampling the pop\_Rock genre.

Data preprocessed through vectorization, PCA feature reduction (k=5) and string Indexing. All the stages are pipelined. Then, we split the data into train and test using pyspark.sql. window library. We did stratified random sampling that assumes observation independence. This is to ensure the splits partition the outcome with minimal sample variance.

Train data is under-sampled by the ratio. Logistic regression model is fitted with hyperparameter tuning and cross validation. K-fold validation and parallelism is set to 8; the number of cores we used.



The whole process is done in 55ms. 37 tasks in each of the 8 job ids. Prediction done with the test data. Accuracy of our model is 96.8%. Hyper tuning parameters for the best model are:

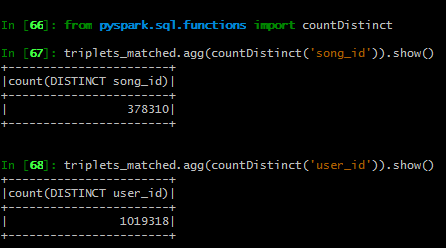


Inclusion of multiclass (multiple genres) has improved the performance of the model.

Song recommendations

Q1 First it will be helpful to know more about the properties of the dataset before you being training the collaborative ﬁltering model.

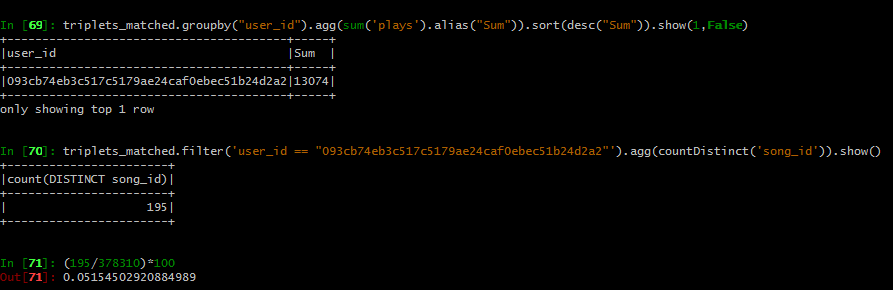
1. How many unique songs are there in the dataset? How many unique users?



After removing all mismatches, there are 378310 unique songs and 1019318 unique users are there.

(b) How many different songs has the most active user played?

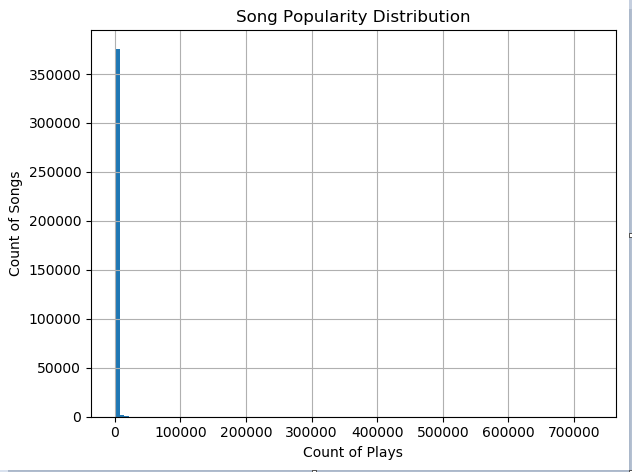
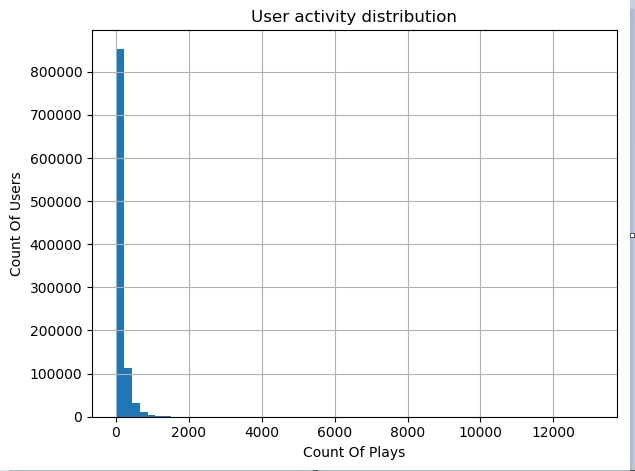
What is this as a percentage of the total number of unique songs in the dataset?



The most active user had played 195 different songs which is 5.15% of the total number of unique songs in the dataset.

(c) Visualize the distribution of song popularity and the distribution of user activity.

What is the shape of these distributions?

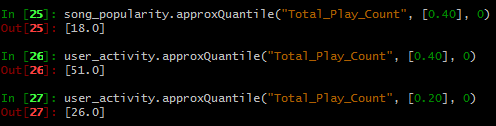
 

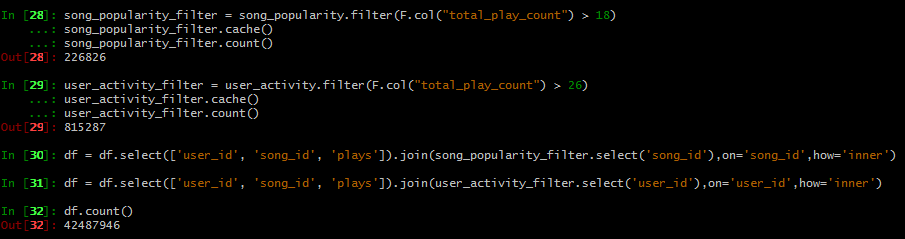
Both distributions are right skewed. Only popular songs are played in a significant amount. Majority of the songs have a smaller number of plays. Similarly, majority of the users exhibits a play count less than 100.

(d) Collaborative ﬁltering determines similar users and songs based on their combined play history. 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.

Create a clean dataset of user-song plays by removing songs which have been played less than N times and users who have listened to fewer than M songs in total. Choose sensible values for N and M and justify your choices, taking into account (a) and (b).

To avoid data sparsity issue, songs that have been played less than 18 (40th percentile) and users who listen songs less than 26 (20th percentile) are removed and inner joined with triplet dataset.

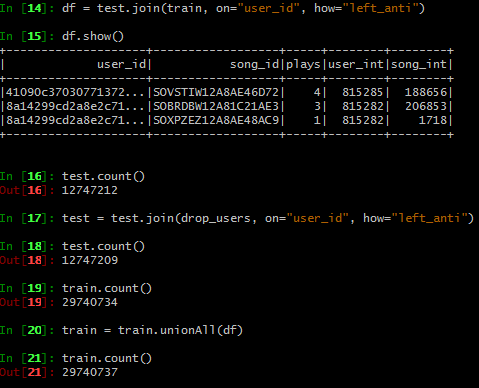




(e) Split the user-song plays into training and test sets. Make sure that the test set contains at least 20% of the plays in total.

Note that due to the nature of the collaborative ﬁltering model, you must ensure that every user in the test set has some user-song plays in the training set as well. Explain why this is required and how you have done this while keeping the selection as random as possible.

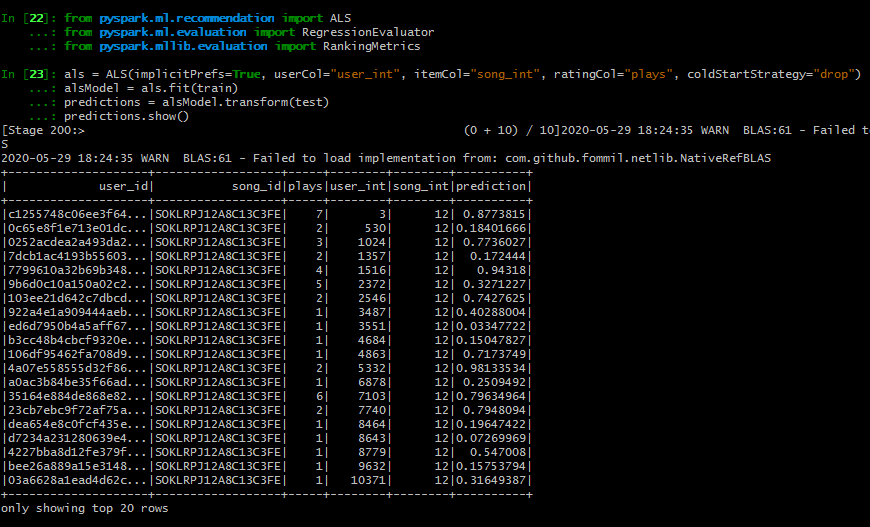
Preprocessed data is split into test and train. Subtract method is used to find the users in the test data but not in the train. 2 users in the test data are not there in the train data. These users where extracted and added to the train data using unionAll method. This is to avoid nominal behaviors while testing.



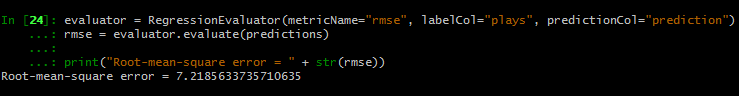
Q2 Next you will train the collaborative ﬁltering model.

1. Use the spark.ml library to train an implicit matrix factorization model using Alternating Least

ALS model is trained and fitted on the test data.

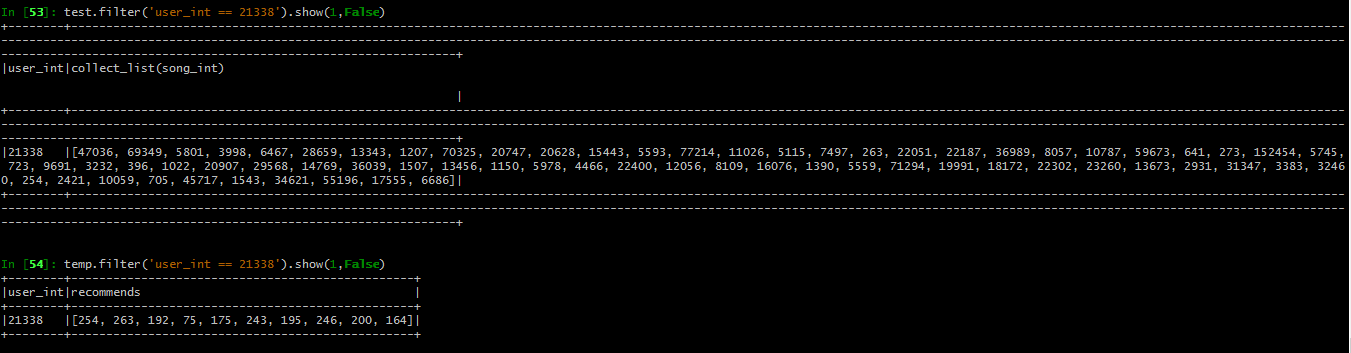


RMSE value is 7.2



1. Select a few of the users from the test set by hand and use the model to generate some recommendations. Compare these recommendations to the songs the user has actually played. Comment on the effectiveness of the collaborative ﬁltering model.

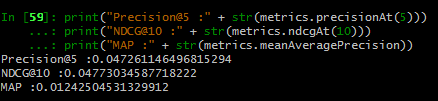
Subset of users is selected and compared with the ALS model prediction. The user that I have compared had listened 2 songs which are actually recommended for him. Yet most of the predictions differs from what the user had actually played. This shows that the performance of the model is average.



(c) Use the test set of user-song plays and recommendations from the collaborative ﬁltering model to compute the following metrics

• Precision @ 5 • NDCG @ 10 • Mean Average Precision (MAP) Look up these metrics and explain why they are useful in evaluating the collaborate ﬁltering model. Explore the limitations of these metrics in evaluating a recommendation system in general. Suggest an alternative method for comparing two recommendation systems in the real world.

Assuming that you could measure future user-song plays based on your recommendations, what other metrics could be useful?



Q3 The method used to train the collaborative ﬁltering model above is one of many.

1. Explain the limitations of using this model to generate recommendations for a streaming music service such as Spotify. In what situations would this model be unable to produce recommendations for a speciﬁc user and why?

In what situations would this model produce low quality recommendations and why?

Spotify is a media service provider and music streaming service. The system works on both explicit and implicit feedbacks. ALS model is associated with implicit feedbacks. A case study of matrix factorization with Spotify revealed certain problems.

* Baseline prediction for an unknown rating (cold start problem)
* Users preferences may change
* Movies are more popular at certain times (Time sensitive baseline predictor)
* Users baseline rating may change
* Low quality recommendation on highly sparse data
* Popularity bias

Cold start problem

When the system cannot draw any inferences for users or items about which it has not yet gathered sufficient information, it is a cold start problem. In this situation, the model is unable to produce recommendations for a specific user (wikipedia).

Sparse data and popularity bias

The model produces low quality recommendations, when music recommendation (streaming data) involves, modeling highly sparse data. Popularity bias, when vast majority of ratings are associated with the popular tracks is another situation where the model performance degrades.

1. Suggest two other recommendation systems that could be used in each of these scenarios respectively.

Collaborative filtering algorithm Singular Value Decomposition (SVD) can handle massive dataset, sparseness of rating matrix, scalability and cold start problem. Content based recommendation system, based on user profiles is an alternative solution to cold start problem.

Item-item collaborative filtering for recommender system is not susceptible to popularity bias and so is a solution to handle this problem. This method is invented and used by Amazone.

1. Based on the song metadata provided in the million song dataset summary, is there any other business logic that could be applied to the recommendations to improve the real world performance of the system?

The whole recommendation system is based on the taste profile of the users. The list of similar artists associated with each track can be utilized. The users would get an improved experience by listening to unknown talents. Also, it could be helpful to provide a platform for the new artists.

# References

Faucett, Tom. *Learning from Imbalanced Classes*. 25 August 2016. online. 28 May 2020. <http://www.svds.com/learning-imbalanced-classes/>.

Kabii, David. *PySpark Feature Engineering and High Dimensional Data Visualization with Spark SQL in an Hour*. 30 July 2019. online. 25 May 2020. <https://medium.com/@david.kabii/pyspark-feature-engineering-and-high-dimensional-data-visualization-with-spark-sql-in-an-hour-b2fea0de2472>.

Kecman, Te-Ming Huang & Vojislav. *LinearSVM*. 2009. online. 22 May 2020. <http://www.linearsvm.com/>.

ProjectPro. *How Data Partitioning in Spark helps achieve more parallelism?* 07 May 2017. online. 20 May 2020. <https://www.dezyre.com/article/how-data-partitioning-in-spark-helps-achieve-more-parallelism/297>.

Rai, Dhiraj. *Logistic Regression in Spark ML*. 02 November 2018. online. 28 May 2020. <https://medium.com/@dhiraj.p.rai/logistic-regression-in-spark-ml-8a95b5f5434c>.

RAY, SUNIL. *6 Easy Steps to Learn Naive Bayes Algorithm with codes in Python and R*. 11 September 2017. online. 22 May 2020. <https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/>.

Spark. *Classification and regression*. n.d. online. 22 May 2020. <https://spark.apache.org/docs/latest/ml-classification-regression.html#gradient-boosted-tree-classifier>.

Wan, Jun. *Oversampling and Undersampling with PySpark*. n.d. online. 22 May 2020. <https://medium.com/@junwan01/oversampling-and-undersampling-with-pyspark-5dbc25cdf253>.

wikipedia. *Cold start (recommender systems)*. n.d. online. 29 May 2020. <https://en.wikipedia.org/wiki/Cold\_start\_(recommender\_systems)>.

Willems, Karlijn. *Apache Spark Tutorial: ML with PySpark*. 29 July 2017. online. 25 May 2020. <https://www.datacamp.com/community/tutorials/apache-spark-tutorial-machine-learning#preprocess>.