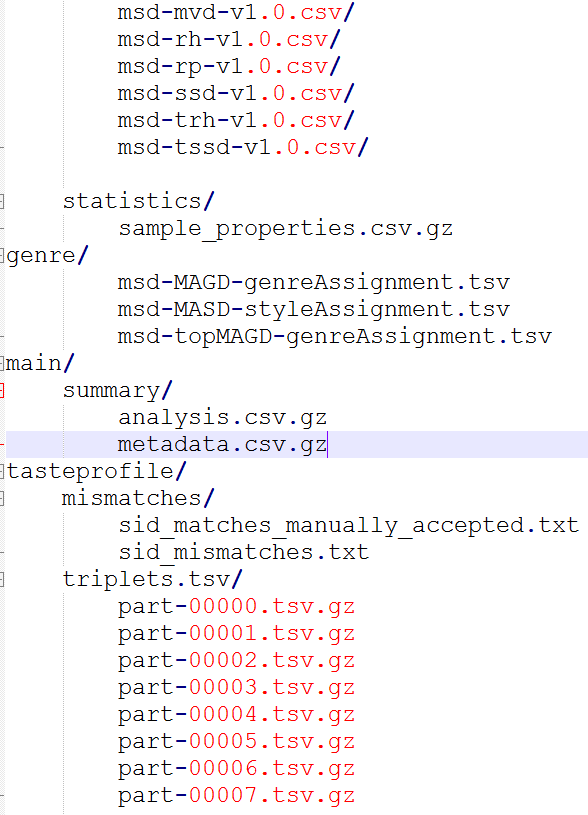
**Data Processing:**

Question1:

1. The directory structure of /data/msd/ in HDFS is shown in below screenshot. The .csv/ directories all contain 8 gzip part files named `part-0000X.csv.gz`. There are four directories in the dataset which are audio, genre, main, taste profile. The Audio directory consists of attributes, features and statistics. The audio attribute files correspond to the audio feature directories. Different attributes file consists of the schema of various different features hence to load such data we need to write a logic such that it extracts the schema and load each features dataset in a loop. Each features csv file is further divided into 8 part files which are gzipped. Thus in order to load any gzip files, it must be loaded by one executer thread to be decompressed and will be loaded into one partition in the data frame. The files which are gzipped cannot be divided further while loading as repartitioning does not work in the gzip files hence for .csv/ files will be loaded using atmost 8 partitions. The directory genre consists of three different files containing genres, styles and top genres respectively. The taste profile directory consists of mismatches and triplets. Mismatches directory consists of the songs which were mismatched and matches which were manually accepted. These files are in text format needs to be extracted from the file and put into data frame. Triplets directory consists of files which are tab separated and are gzip thus one executer thread for each part files is required. The main directory has summary directory which consists of analysis and metadata files. The genre directory consists of various files which are tab separated, hence while loading them we need to specify ‘\t’ as delimiter. Triplets directory consists of dataset of mismatches and mismatches manually accepted which are text format. Thus to load these file we need to extract each and every line using a for loop and put it in data frame. We have used parallelization of 64 for loading this data. The size of the audio directory is 12.3 GB which is highest compared to the other directories which has size in MB. Genre being 30.1 MB, main being 174 MB and taste profile being 490.4 MB.





b) It may be useful to repartition the larger datasets before doing any actual

data processing or analysis to increase the total number of partitions and improve the level of parallelism that can be achieved. This could also help in speeding up developing and testing code, as data can be limited, repartitioned, and then cached. Below command can be used for this purpose->

triplets = triplets.repartition(100)

c) The count of number of rows for each dataset can be obtained using hdfs command before loading the dataset which has been provided in the .py file. We can see the count of triplet’s dataset is too high being 48373586, whereas msd-MAGD-genreAssignment 422714 rows and style dataset has 273936 rows. All the features dataset is in the range of 994188 to 995001 number of rows. Matches manually accepted are 938 whereas mismatches count is 19094.

How do the counts compare to the total number of unique songs? – Number of unique songs are 384546. This shows that there are many song IDs which are repeated in the triplet’s dataset.

What is song id and what is track id?

Question 2)

1. The taste profile dataset consists of the song ids which are mismatched. Thus we need to remove mismatched song ids from our dataset. Thus the total triplets count was 48373586 and after removing the mismatched songs we have triplet count as 45795111.
2. As the dataset Audio has several features and several schemas, hence I have automated the process to load the features dataset. As all the feature dataset has 8 partitions in the csv file which are gzipped, we can achieve parallelism of 8 worker nodes.

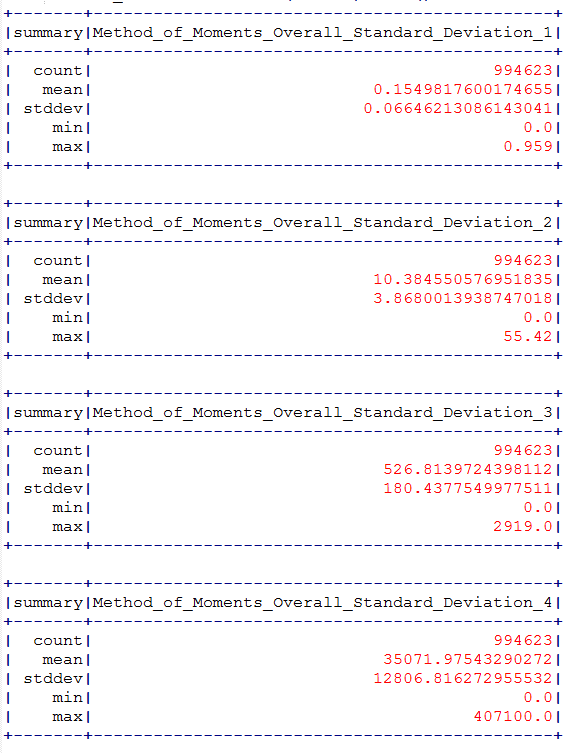
**Audio similarity:**

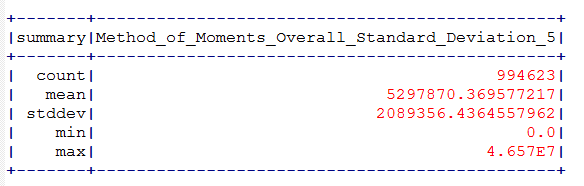
Question1)

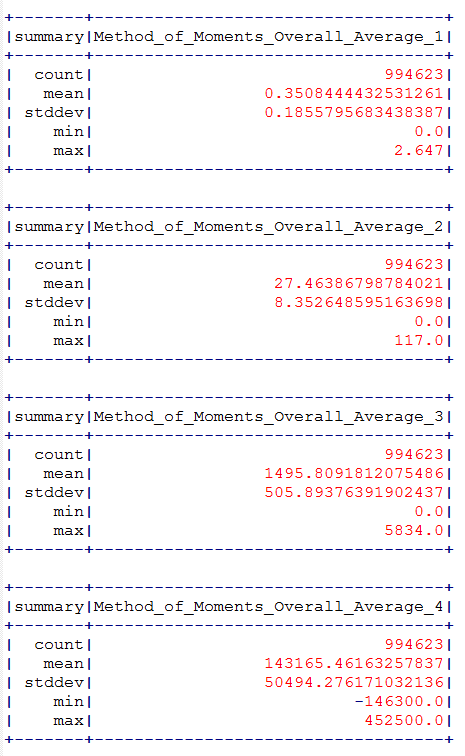
For the further classification models I have chosen the dataset msd-jmir-methods-of-moments-all-v1.0 which has 11 columns and 994623 number of rows.

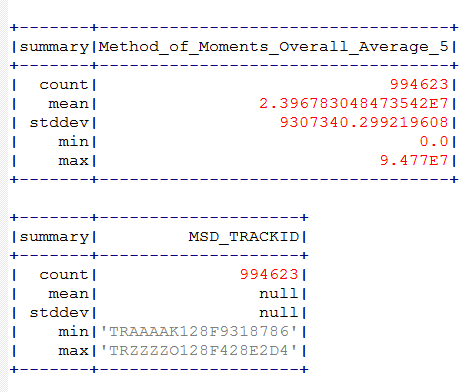
Before going for the Binary classification and the multiclass classification we will see the descriptive statistics for each of the columns as below-

Can change and make boxplot of the datasets for outlier detection. For scaling required or not

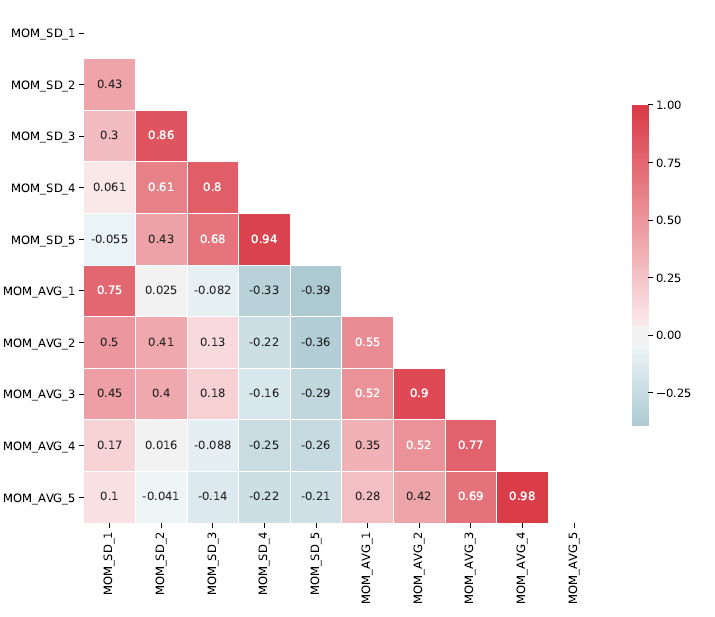






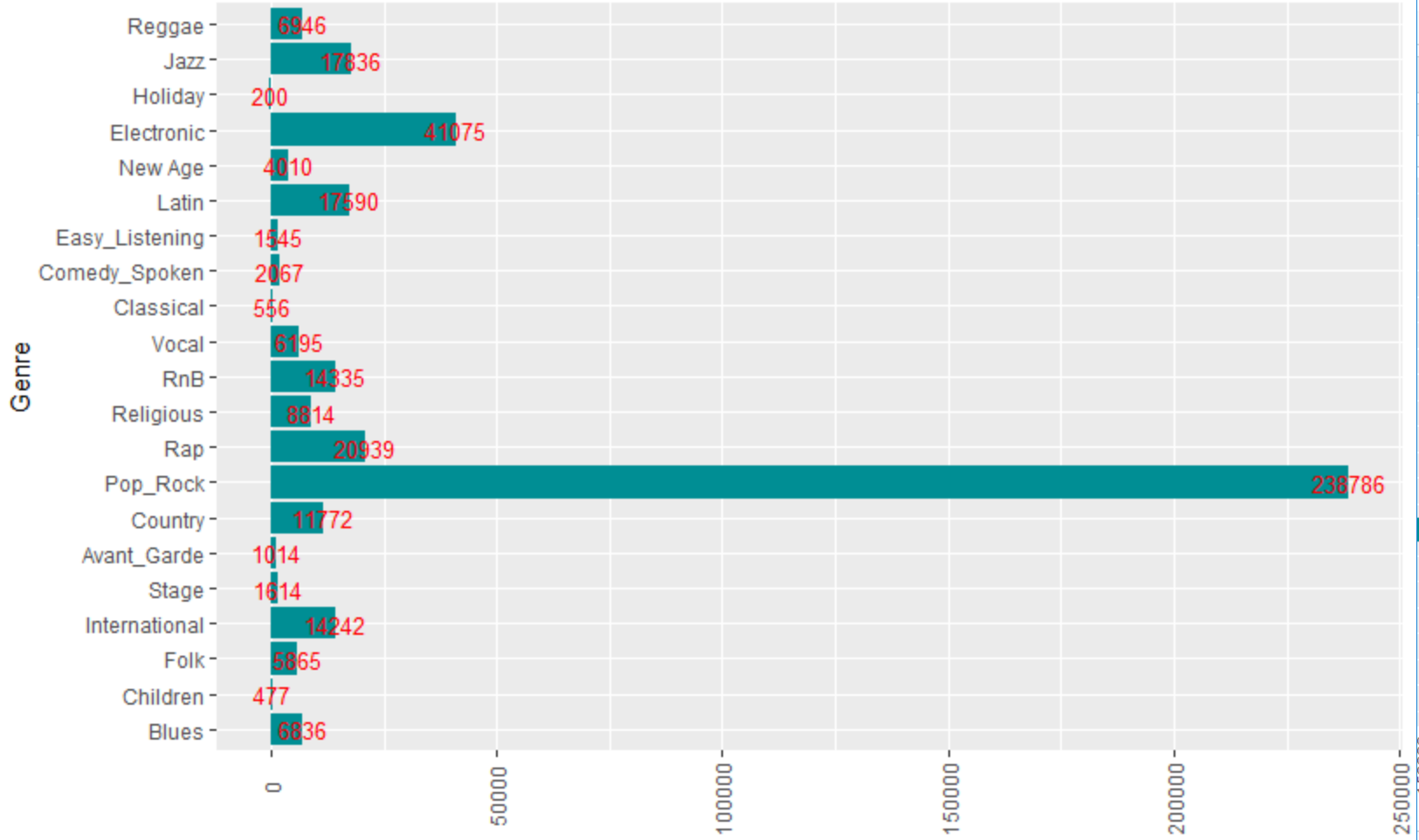


By the below correlation plot we can understand the correlation between different features-



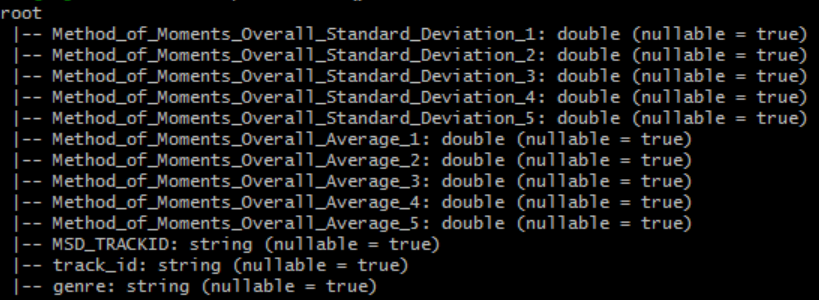
Column MOM\_Avg\_4 and MOM\_Avg\_5 are highly correlated. The highly correlated columns does not give much information hence we can delete either of the columns of the two.

Next we will take our genre dataset (MAGD) and visualize the distribution of genres. –



The above visualization tells us that few labels are high number of observations compared to other genres. This will lead to class imbalance problem if we are fitting a model. Hence we have to be careful when fitting the model.

In this part of the question I have merged the genre and features dataset so that every song has label. This is done so that features columns can help us classify the songs in the correct label or genre.



**Binary Classification:**

Question2a) In this section our aim is to create a model with binary target value hence we will do binary classification. We have created label column which contains information whether the genre is electronic or not. Electronic genre is given the value 1 and others are marked as 0. In this section our goal is to get the electronic genre, correctly at the same time we would not want other genre to be classified as electronic as it may lead to unsatisfied users. If a user likes only electronic song, constantly recommending wrong genres to him might cause user unsatisfied and he might stop using the website. However, if some electronic songs are not classified as electronic then it might not cause so much problem. Hence we will focus on the **recall** value for this problem statement.

Before diving into any model we also need to see if the features provided to us needs scaling or not. The describe function gives details of all the columns of the dataset. We can see that the mean of all the columns varies by huge number. Hence, it will be preferable to standardise the features. For this I will be choosing three models for binary classification problem from spark.ml library. I will be choosing the models based on its explain ability, interpretability, training speed, predictive accuracy, hyper parameter tuning, dimensionality, and issues with scaling. Keeping this in mind I have chosen my three models as logistic regression, Random Forest and Gradient boost.

The **Logistic regression** model is based on log transform of the linear regression model. It has good interpretability and can work well for binary classification problem. The binary logistic regression requires the dependant variable to be binary and the representation 1 is generally considered as our desired outcome. Logistic regression works well with large sample sizes and in this case we have ample amount of data, hence we can take logistic regression model is one of our choice. This model has parameter class weights which can take care of the class imbalance issue which we might face with this data. There are lot of features in our dataset, hence we need a way to vectorise it using vector assembler. This creates vectors of the all the features depending on which the probability is calculated for the given test set to assign it to the respective class. The logistic regression does not do implicit feature selection hence considering all the features to calculate the outcome. This model is comparatively faster as does not have to do feature selection. In order to do feature selection in this method we will be needing hyper parameter tuning. As this is dataset has class imbalance we can tune the threshold value to get a better result. More number of parameter tuning can lead to longer time for training the model.

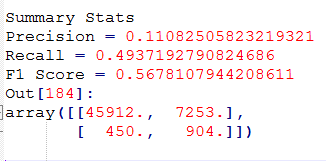
**Random Forest**: It is interpretable and transparent thus can be explained to the client easily. Tuning the hyper parameters is less compared to logistic regression. This is doing better job compared to logistic regression. Random forest performs well on large datasets and can handle thousands of variables without variable deletion. As it can estimate which variables are important for the classification. It can effectively handle missing values though in our dataset we are not facing this problem. It can balance the error rate in class population unbalanced dataset which will be very useful for our dataset as it is unbalanced. This model is fast and useful for large dataset like we have. If we go for Bayesian or Gaussian process for huge datasets, the training speed would be too low.

The third model chosen is **Gradient Boosting**: This is a tree based model hence it is interpretable. It trains model in sequential manner and takes the outcome of each tree to assign weights to them. The tree which results in good classification are given more weights and the tree with poor classification are given more weights. Thus it is an ensemble model. As it is an ensemble model training time would be little more to simple logistic regression. This model does a better job than random forest as it trains itself from previous tree to improve. Hyper parameters to tune are less compared to logistic regression. Gives a better result than logistic regression at the cost of more time in training the model.

Question 2b) I have taken Electronic as label 1 and other genres as label 0 and stored it in the label column. This is required because for binary classification we need only two classes. The class is highly imbalanced for the binary level as we can see below-

|  |  |
| --- | --- |
| Label | count |
| Electronic (1) | 40027 |
| Other genres (0) | 373266 |

This tells that the class imbalance problem needs to be taken care when we are training our model as without taking care of class imbalance situation the model will do a bad prediction for minority class which in this case is electronic genre.



Question2c) For solving the class imbalance problem, I have chosen different methods for different models. For the logistic regression model, I have used stratified sampling for test train split and then used class weight approach for re-sampling as it has a parameter of class weight which can take care of the class imbalance. Whereas for gradient boost model and random forest classifiers, I have chosen stratified sampling for splitting the data into test and train followed by up sampling and down sampling the train data to fit the model which reduces class imbalance problem.

The **stratified sampling** has been chosen for the test train split so that while training and testing our model we have data for both the labels. I had chosen sampleBy function for stratified sampling. Stratified sampling mainly helps in preparing our test data in such a way that it has all the class labels present in it. This is completely different from resampling method of up sampling and down sampling. However, after using this approach I learnt that it does not do exact stratified sampling as there is a possibility that we might not get rows for all the class labels in our test or train. Thus the window approach is better for the stratified sampling which performs calculation over a set of rows. In this approach we are creating random column which generates random numbers for each rows in the dataset. This ensures that we have desired percentage of classes in test and train set. But in case of sampleBy there is a probability of getting no rows for a particular class.

After splitting the data, we again need to consider class imbalance as stratified sampling will not completely reduce the class imbalance, it helps in ensuring if we have data for both the labels in our test and train in good amount.

For further reducing imbalance problem, I have gone with down sampling approach first but as the difference in the two labels is very high, completely down sampling the data will cause huge data loss. Whereas completely up sampling may bring too much of biasness in the data as the data points are generated from the old ones which can’t introduce much variance in the dataset. Thus I have chosen to up sample the electronic label whereas down sample the other labels. I have not kept the count of both the labels same as the difference between the labels are too high and bringing to exact amount of data for both labels may not be required. Without taking care of the class imbalance all the models are performing bad.

The requirement of the balancing the class also depend on what we want to focus our classification on. For example, if we are more concerned with correct classification of other genres, unbalanced class will cause impact but not as much as if our main focus is classification of electronic genre.

Before stratified sampling I have the below count of the class labels:

|  |  |
| --- | --- |
| **Label** | **Count** |
| 0 | 373266 |
| 1 | 40027 |

After the stratified sampling and splitting in test and train dataset:

Train Dataset:

|  |  |
| --- | --- |
| **Label** | **Count** |
| 0 |  |
| 1 |  |

Test Dataset:

|  |  |
| --- | --- |
| **Label** | **Count** |
| 0 |  |
| 1 |  |

After re-sampling using up-sampling and down sampling:

**Pre- processing** **Steps** ->

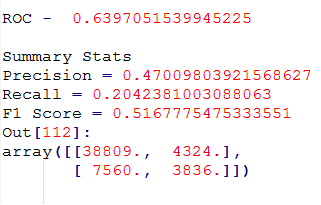
Though trees do not mind missing values, taking care of missing values, helps in the performance of logistic regression model. The dataset **does not contain any missing values**; hence we do not require dropping the missing values in this case. I have not used any imputation technique as well as our dataset is not having missing values.

Row count with NA – 413293 | Row count without NA – 413293

Though the below distribution shows that the features have huge difference in the scales but as logistic regression and tree methods are not affected by the scale difference, we need not scale the features.

Next step I will do for pre-processing would be to **remove the columns which are highly correlated** to each other. This step is necessary as few methods works well with correlated variables whereas few does not, hence it will be preferable to apply this processing. Logistic regression deals with OLS as it is log transform of linear regression, hence correlated values needs to be removed for this case. The columns such as Method\_of\_Moments\_Overall\_Standard\_Deviation\_4 and Method\_of\_Moments\_Overall\_Standard\_Deviation\_5, Method\_of\_Moments\_Overall\_Average\_4 and Method\_of\_Moments\_Overall\_Average\_5, Method\_of\_Moments\_Overall\_Average\_2 and Method\_of\_Moments\_Overall\_Average\_3 are highly correlated to each other, however I will consider correlation above 0.98 as high correlation and as Method\_of\_Moments\_Overall\_Average\_5 and Method\_of\_Moments\_Overall\_Average\_4 have high correlation and I am removing Method\_of\_Moments\_Overall\_Average\_5 in my pre-processing step.

Next step would be to vectorise different columns using **vector assembler** so that it converts the columns into single vector column. It is useful for combining raw features and features generated by different feature transformers into a single feature vector, in order to train ML models like logistic regression and decision trees. **Vector Slicer** is not used in this case as our dataset has only 10 features and we might get better information if we keep all the features. However, I have used **PCA** to get better components which might contribute in the genre classification. And found that the recall value is improving slightly for my genre electronic from 0.536 to 0.543. PCA changes our existing features into useful components which can best describe our genres for our dataset. However, model does not perform extraordinarily better with PCA, this might be because the features in the dataset provided to us are not much different from each other. The columns are all method of moment average and standard deviation which are not very different from each other. PCA are helpful when the features are non-correlated and different from each other. I have checked the PCA features only on one model that is random forest-



**Training using different models without hyper parameter tuning:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Metrics** | **Logistic Regression** | **Random Forest** | **Gradient Boost** |
| Precision (1) | 0.9143142643017736 | 0.4543653250773994 |  |
| Recall (1) | 0.12219347299891495 | 0.31066892464013546 |  |
| F1 Score (1) | 0.2155762689402306 | 0.8492200831590838 |  |
| ROC | 0.746827541926941 | 0.7740343809452515 | 0.7935033818142319 |

**Metrics** Used for evaluating the model- The model can be evaluated by any of these metrics depending upon our requirement. In this problem my first concern is how well the Electronic genre is getting predicted and we are not much concerned about the other genre prediction. Hence for taking recall and precision value I have considered only for label 1 rather than taking overall recall and precision.

**Training using different models with hyper parameter tuning:**

Number of folds used by me for the hyper parameter tuning for each model is 5 as increasing the number of folds will take more time.

|  |  |  |  |
| --- | --- | --- | --- |
| **Metrics** | **Logistic Regression** | **Random Forest** | **Gradient Boost** |
| Precision (1) | 0.6908568573569822 | 0.3744616164175323 | 0.5192307692307693 |
| Recall (1) | 0.18253522986040066 | 0.37389324563622567 | 0.340603890290551 |
| F1 Score (1) | 0.28877228704936436 | 0.8798965127719273 | 0.8550491358861403 |
| ROC | 0.7383256535483098 | 0.8001028812800077 | 0.8122251039436328 |

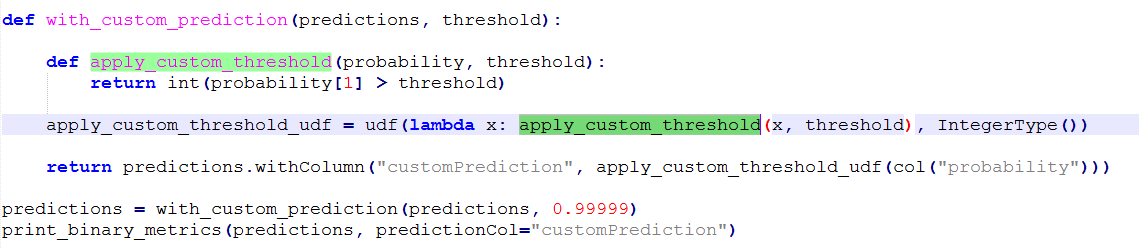
**Explain each hyper parameter**

**Hyper parameter tuning with Logistic regression:**

For the logistic regression I have used below hyper parameter tuning parameters-

1. Regularization parameter – which helps in regularization which prevents over fitting. Thus can perform well with test data.
2. Elastic Net Parameter – The elastic net parameter used by me in this case are 0.0, 0.5 and 1.0. Parameter of 0.0 is for ridge and 1.0 is for lasso whereas 0.5 is combination of both.
3. Number of iterations- I have chosen iterations as 5, 10 and 20. This parameter is used to define the number of iterations that we might be using in order to optimize our model. I could have given more range of values to see if the results changes with changing the parameters.

Threshold – This parameter tells the model at what probability, it needs to classify certain song to label 1. For instance, if threshold is 0.7 and if the probability of the features to lie in class 1 is 0.5 then it will get assigned to class 0. I tried with different threshold, without using hyper parameter tuning and found the result was good for 0.3 threshold. When using default 0.5 value for threshold, we see that the precision for electronic genre has decreased by 20% with very slight increase in recall by 7%. Hence, I have kept my threshold value as 0.3. However, instead of increasing and decreasing threshold, we could also implement our custom prediction with changed threshold as in the below code-



As for different datasets different values of hyper parameter is good, hence to get the right value we must change the values and check our performance.

With hyper parameter tuning for logistic regression, the F1 score for electronics genre improves very slightly when compared to without hyper parameter tuning.

**Hyper-parameter tuning in Gradient Boost:**

Max\_depth – This parameter is used to specify how deep the tree would be. The deeper the tree more information will be gathered. If we make this value too large, this will lead to overfitting of the training data. Thus optimum value needs to be chosen.

Max\_Bin- yet to write

Max\_Iter- This specifies the number of iterations it will go through.

**Hyper Parameter tuning for Random Forest:**

**numTrees** - Number of trees in the forest. This parameter is usually the most important setting

**maxDepth** - Max number of levels in each decision tree

As random forest behaving really slow when putting the number of trees as more than 20 and throwing java heap size memory issue, thus I have chosen only 10 trees when tuning the hyper parameter. I have fixed the max tree depth as 5 and 10. This number generally works as referred article and could not check further on these number due to time constraint.

**Multiclass classification** –

For multiclass classification there are two approach that can be used. These are one vs one and one vs. rest approach. **One vs rest** approach is generally used with logistic regression to convert the model from operating only for binary classification to multi-class classification problems. It is an example of machine learning reduction for performing multi-class classification problem giving a base classifier which can perform binary classification efficiently. In this case our purpose is to predict a particular class. The classifier for class m is trained to predict whether the label is m or not, distinguishing m from rest of the class.

Another Approach of converting a binary classifier to multiclass classifier is **one vs one**. In this approach Two pairs of classes are selected at a time and a binary classifier trained for them. This is done for every possible pair of classes thus there are n(n-1)/2 of them where n is the total number of classes. During the classification phases, all the binary classifiers are tested. For each of them, a “win” for one class is a vote for that class. The class with the most votes wins. SVM can be implemented using this approach as it is primarily a binary classifier.

In general, the one vs one approach can take more number of time and give better accuracy whereas one vs. rest can be cost effective. However, it strongly depends on the dataset as to which one will perform better. For highly imbalanced dataset, one vs one might perform better at the cost of high computational cost. Hence, another approach might be using one vs. all with class weight. And as we have huge dataset, I would be going with one vs rest approach.

**Logistic regression** is generally used for binary classification hence to use it for multi-class problem, we can either use cross entropy or one vs. all approach. Random forest can by itself solve the multiclass problem but it might behave better with **one vs. rest** approach. For multiclass classification, I have checked logistic regression and random forest with one vs rest approach which give the below output. Though the ROC is almost similar to the random forest without one vs. rest approach, it is found that one vs. rest approach is not able to classify all the classes. Few classes are not classified at all.

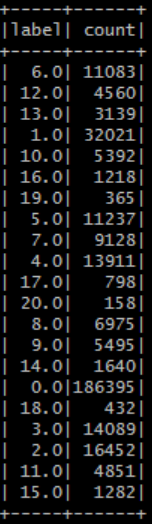
Logistic Regression:

# ROC = 0.25347

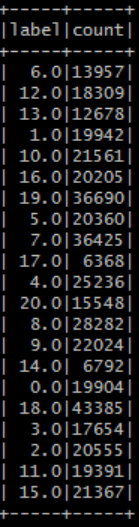
Random Forest:

# ROC = 0.252532

For multiclass classification class imbalance has to be taken care for all the classes. The count for different labels just after stratified sampling using sampleBy function is as below. We see that class 15 is not there in the list. Thus sampleBy is missing few class. Hence we need to use the window function approach which does the exact stratification.



Even after doing the stratified sampling, we have 185986 records for class 0 and only 149 observations for class 20 which might not be useful in training or prediction. The model might classify all the observation to class 0. Hence up sampling and down sampling is required for balancing the class. Thus performing up sampling for the classes with high count and down sampling for classes with low count value. If we the upsampling and downsampling is not proper, we are not getting any classification for the classes 4,5,15,16,18,19 and 20. This might be because further up sampling is required for minority classes. Thus, I needed to change the ratios for few of the classes for getting classification for all the classes. The count of the classes after up sampling and down sampling is as below-



Results for Random forest classifier for multiclass classification is as below-