SENG 550 - L02

Project Final Report

Group Members:

Name	**UCID**
Ivan Lou Tompong	
Alliana Dela Pena	
Thessalonika Magadia	

Table of Contents

Introduction and Motivation

Our goal is to build a model that will accurately predict the genre of a song based on its characteristics such as danceability, loudness, acousticness, topics mentioned in the song etc. We were inspired by Spotify and how it can create playlists based on genres.

Data Collection

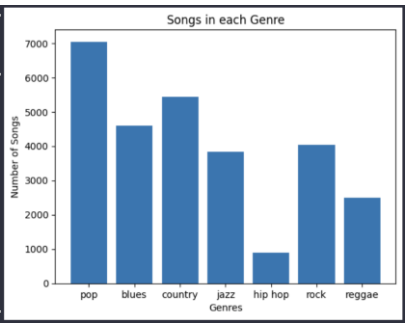
During the initial data collection, we decided to download a subset of the *Million Song Dataset* which contains roughly 10 000 songs, instead of the entire dataset. This is because downloading the entire dataset required 280 GB of storage space, which is around a quarter to a half of the available storage on each of our devices, while the subset is only around 1.8 GB compressed and 2.5 GB uncompressed. Initially, we planned to use this subset of the *Million Song Dataset* but after further inspection it was missing a lot of data we needed, a lot of the values were null such as the genre. Not only were there missing values, we also discovered that the files themselves in the subset we corrupt. Therefore, we decided to not use the *Million Song Dataset* and instead search for another dataset because in addition to the missing values, we were also unsure which of the values were valid or corrupt. Upon further searching we found the *Music Dataset: 1950 - 2019* [1] which had more than 28 thousand valid values for genre and categories to describe songs (energy, danceability, etc.).

Data Inspection and Validation

We performed our initial data inspection and validation on the subset of the *Million Song Dataset.* After writing and running some code to check the files in the dataset, we discovered that the files were corrupted and missing important values. Some values in the file were either completely blank or changed to 'NaN'. After discovering that the files were corrupt, there was no way for us to validate which values were valid and which ones were corrupted. At this point, we decided to stop using the subset of the *Million Song Dataset* and switched to the *Music Dataset: 1950 - 2019* [1].

To ensure our new dataset was valid we verified that there were no missing values or invalid values and that each row contained a unique song. We also checked to make sure that there was a good amount of song data for each genre (photo below). Each genre had around a thousand or more songs. Additionally, all the features we observed used the same scale (ranged from 0 - 1), so during training the weighing was not skewed.

genre	Total_Count
pop	7042
blues	4604
country	5445
jazz	3845
hip hop	904
rock	4034
reggae	2498



Data Filtering

We filtered the song dataset to contain the features we predicted will likely have the same values for the same genre. The list of features we extracted were obscene, music, valence, and energy of the song.

Each topic was its own feature and gave a measurement of how much the song included that topic. The closer the value is to 1, the more likely it is to include that topic. The list of topics were dating, violence, world/life, night/time, shake the audience, family/gospel, romantic, communication obscene, music, movement/place, light/visual perceptions, family/spiritual, like/girls, sadness and feelings. (Note: we evaluated all these features when performing EDA)

The features used to train both the Decision Tree and Logistic Regression for determining a song's genre are described below:

Obscene: measurement of how likely a song is to have a topic that is obscene

Music: measurement of how likely a song is to have a topic that is obscene

Valence: measurement of the positiveness expressed in the song. A song that has high valence sounds more happy and euphoric while low valence sounds sad or angry. 0 means low valence and 1 means high valence.

Energy: measurement of the energy of the song. High energy songs are fast and loud while low energy songs are slow and soft. 0 means low energy and 1 means high energy.

Data Transformations

We encountered some challenges in finding other datasets so we did not merge data from different sources. Firstly, we wanted to add more songs to the dataset especially for the hip hop

genre since it had the least amount of songs. Unfortunately, it was difficult to find music datasets with valid data that contained the genre of the songs along with some of the features in *Music Dataset: 1950 - 2019*. The popular music dataset, *Million Song Dataset*, contained lots of null values for important fields such as genre. Secondly, we thought of adding more features to the existing songs. This could be done by finding datasets with the same songs but different features. Alas, the datasets we took a look at had a few thousand or less songs. The likelihood of being able to join the same songs from the new dataset to our current dataset was very low since millions of songs are released each year. There would not be any value in taking a look at the added features if most of them had null values.

The *Music Dataset: 1950 - 2019* was provided in a Comma Separated Value (.csv) format. One major issue we came across when parsing through the data was that the values for the artist_name, track_name, and lyrics features contained commas. Usually, to avoid problems with parsing the data from a CSV file, any comma that is not a delimiter is surrounded by a pair of double quotes. However, the CSV file for our dataset did not implement this solution, causing our program to parse the data improperly. To solve this problem, we took advantage of the Pandas Python library and Regular Expressions. Instead of immediately converting it into an RDD, after reading the file we converted it into a Pandas dataframe to take advantage of the data manipulation functionalities provided by the Pandas library. After creating the data frame, we then used regular expressions to remove the non-delimiter commas as shown in the image below. Although we did not use the artist_name, track_name, and lyrics features, we still had to modify the values of these features since they affected how the data was parsed.

```
#Create a Pandas Dataframe
df = pd.read_csv('/content/drive/MyDrive/THE Project Dataset/tcc_ceds_music.csv')

#Remove all the unwanted commas from the values in each cell specified; this is to allow for proper parsing of the CSV file
df["artist_name"] = df["artist_name"].replace(",", " ", regex = True)
df["track_name"] = df["track_name"].replace(",", " ", regex = True)
df["lyrics"] = df["lyrics"].replace(",", " ", regex = True)
```

Exploratory Data Analysis (EDA)

We performed EDA on the training data to determine which features to use for the model. We calculated the mean, min, max and standard deviation of each feature for each genre. The criteria for choosing the features was based on differences in values for each genre and that the standard deviation was small. After analyzing the results, the features that fit the criteria were valence, energy, obscene and music.

Average

Genre	avg(obscene)	avg(music)	avg(valence)	avg(energy)
pop	0.09809324238080422	0.055759647891562215	0.5321168434846117	0.6005143422166719
blues	0.07558528275348045	0.06737323654090048	0.5450540632352335	0.5812731100668176
jazz	0.09568490826683007	0.07807075086180251	0.47238867089041553	0.46469295572994157
hip hop	0.41242805274349587	0.02690622120102804	0.598321992318343 3	0.7040798311491189
country	0.0671034565909882	0.07346635509551783	0.5314313679678567	0.4679093038660788
rock	0.06345839684253082	0.04302056581252043	0.45637398092628817	0.7027609184154825
reggae	0.1382705776863633	0.04790059794034566	0.7023843115146462	0.5896337716602147

Minimum

Genre	min(obscene)	min(music)	min(valence)	min(energy)
pop	2.89185E-4	2.89185E-4	0.0103050288540807	0.0184878646860642
blues	3.33111E-4	3.33111E-4	0.0108202802967848	0.007056076
jazz	3.31016E-4	3.20924E-4	0.0	0.007556592
hip hop	3.48554E-4	3.17058E-4	0.015045342126957 9	0.2922701890189162
country	2.92398E-4	2.92398E-4	0.0154575432811211	0.0104796065702952
rock	3.11429E-4	3.18979E-4	0.011232481450948	0.0192886904976411
reggae	3.09598E-4	2.97354E-4	0.0136026380873866	0.064635452

Max

Genre	max(obscene)	max(music)	max(valence)	max(energy)
pop	0.9262233306627072	0.8748923115545906	0.9917559769167352	0.9979979354710576
blues	0.8815789473684204	0.9016104088116506	0.9969084913437758	1.0
jazz	0.9058169516996866	0.826659165	0.988664468	0.9989989677355288
hip hop	0.919747331588844	0.747253808	0.9711459192085736	0.9919917418842308
country	0.891104802015270 1	0.8646616513489878	0.9866034624896948	0.9919917418842308
rock	0.9536469159830224	0.8102126284088121	0.9804204451772464	0.9989989677355288
reggae	0.8722641064030218	0.956937797832092	1.0	0.9959958709421154

Standard Deviation

Genre	stddev_samp(obscene)	stddev_samp(music)	stddev_samp(valence)	stddev_samp(energy)
pop	0.18512381756594443	0.11995301175173757	0.24857585347008493	0.2294654493314584
blues	0.15308830695164155	0.13200302943841546	0.23982362159462361	0.24602686310743038
jazz	0.18523004418151445	0.14075334972898482	0.2671895901832142	0.2616898542690432
hip hop	0.23100288170534983	0.0727821827478837	0.20448445342944158	0.1475720132883968
country	0.13982235881260477	0.1343406063945987	0.23862588897446743	0.22398193252111478
rock	0.1415749383362921	0.100512839173911	0.24383903776915328	0.23183810061904073
reggae	0.2037808365995177	0.11118584769217883	0.19675791605119303	0.17156156437208056

Model Building

Predicting the genre of a song would be categorized as a classification problem. Since we would be classifying each song between seven of the genres (pop, country, blues, rock, jazz, reggae and hip hop), we would need to build a multi-class classification model. We decided to go with the logistic regression model and the decision tree model which are widely used.

Initially, we were using Spark's `LogisticRegression` library to build the model but when we were analyzing the results, we realized that it is only able to be used for binary classification (2 classes). So we opted to use `LogisticRegressionWithLBFGS` which allowed for multiclass classification.

For both models, we needed to transform the data to be used by `LogisticRegressionWithLBFGS` and `DecisionTreeClassifier`. We first encoded the genre using StringIndexer to be represented by a numeric value. After the encoding, the feature column was merged into a single vector column called features. After transforming the training data into an acceptable input format, the model was able to be built.

Results

In this section we will be going over the evaluation metrics results of the `LogisticRegressionWithLBFGS` and `DecisionTreeClassifier` model. We took a look at the accuracy, precision, true positive rate, false positive rate, f-measure and recall values for both models. Additionally, in each model we observed the evaluation metrics for each label as well as overall.

Logistic Regression Model

Below is the logistic regression formula (not regularized) used to find the weight (w) that will result in the minimized logloss. It is an iterative technique, for each training data point it will calculate the log loss of the corresponding weight. Weight will keep being updated until minimized log loss is found.

$$\min_{\mathbf{w}} \sum_{i=1}^{n} \ell_{log}\left(y^{(i)} \cdot \mathbf{w}^{\top}\mathbf{x}^{(i)}\right)$$

Input - example Labeled Point:

```
[0.00228833,0.439339511345486,0.1560181368507831,0.176150446340266] 4.0
```

Example of a prediction:

```
prediction = lr_model.predict([0.001144165, 0.001144165,
0.6465375103050287, 0.2542309629690136])
```

Output - expected output of prediction above (country = 1):

```
1
```

In the example above the model correctly predicts the genre, but this is not always the case. This is why we need to also use evaluation metrics to determine how good a model is. We can do this by applying the model to the test data then compare the actual results vs predicted results. Below are the evaluation metrics for the Logistic Regression Model.

Training data: overall evaluation metrics:

```
Accuracy:  0.28977187640393354
Precision:  0.2498500296896114
True Positive Rate:  0.28977187640393354
False Positive Rate:  0.19339745330727665
F-Measure:  0.2302213933772739
Recall:  0.28977187640393354
```

Test data: overall evaluation metrics:

```
Accuracy:  0.28348722868449455
Precision:  0.24749787416651806
True Positive Rate:  0.28348722868449455
False Positive Rate:  0.18731820750428999
```

```
F-Measure:  0.223609301506696
Recall:  0.28348722868449455
```

Test data: Evaluation metrics by label:

```
Class 0.0 true positive rate: 0.6217980914113511
Class 0.0 false positive rate: 0.5356017643352237
Class 0.0 precision: 0.2669253988788271
Class 0.0 recall: 0.6217980914113511
Class 0.0 F1 Measure: 0.37351033338361744

Class 1.0 true positive rate: 0.3712720632988436
Class 1.0 false positive rate: 0.20235961768219832
Class 1.0 precision: 0.3104325699745547
Class 1.0 recall: 0.3712720632988436
Class 1.0 F1 Measure: 0.33813747228381374

Class 2.0 true positive rate: 0.0
Class 2.0 false positive rate: 0.0
Class 2.0 precision: 0.0
Class 2.0 recall: 0.0
Class 2.0 F1 Measure: 0.0

Class 3.0 true positive rate: 0.34831460674157305
Class 3.0 false positive rate: 0.11347813979392926
Class 3.0 precision: 0.3308702791461412
Class 3.0 recall: 0.34831460674157305
Class 3.0 F1 Measure: 0.33936842105263154

Class 4.0 true positive rate: 0.036440677966101696
Class 4.0 false positive rate: 0.009358849001257158
Class 4.0 precision: 0.39090909090909093
Class 4.0 recall: 0.036440677966101696
Class 4.0 F1 Measure: 0.06666666666666668

Class 5.0 true positive rate: 0.046831955922865015
Class 5.0 false positive rate: 0.019834493629318273
Class 5.0 precision: 0.1837837837837838
Class 5.0 recall: 0.046831955922865015
Class 5.0 F1 Measure: 0.07464324917672888

Class 6.0 true positive rate: 0.12949640287769784
Class 6.0 false positive rate: 0.023198114377868752
Class 6.0 precision: 0.16143497757847533
Class 6.0 recall: 0.12949640287769784
```

```
Class 6.0 F1 Measure: 0.1437125748502994
```

Decision Tree Model

Overall, the Decision Tree model has very similar evaluation metrics to the Logistic Regression.
Training data: overall evaluation metrics:

```
Accuracy:  0.31418160035940695
Precision:  0.3268263734996215
True Positive Rate:  0.314181600359407
False Positive Rate:  0.17674426604287155
F-Measure:  0.27600917878074344
Recall:  0.314181600359407
```

Test data: overall evaluation metrics:

```
Accuracy:  0.25686533157452934
Precision:  0.24375994587938324
True Positive Rate:  0.2568653315745293
False Positive Rate:  0.18736548953291532
F-Measure:  0.20247133934151137
Recall:  0.2568653315745293
```

Test data: Evaluation metrics by label:

```
Class 0.0 True Positive Rate: 0.5479658463083877
Class 0.0 False Positive Rate: 0.44029615626969126
Class 0.0 Precision: 0.28075141533710757
Class 0.0 Recall: 0.5479658463083877
Class 0.0 F-Measure: 0.3712778628551982

Class 1.0 True Positive Rate: 0.54169202678028
Class 1.0 False Positive Rate: 0.2956989247311828
Class 1.0 Precision: 0.31010452961672474
Class 1.0 Recall: 0.54169202678028
Class 1.0 F-Measure: 0.3944161311766009

Class 2.0 True Positive Rate: 0.0036656891495601175
Class 2.0 False Positive Rate: 0.004874551971326165
Class 2.0 Precision: 0.1282051282051282
Class 2.0 Recall: 0.0036656891495601175
Class 2.0 F-Measure: 0.007127583749109052

Class 3.0 True Positive Rate: 0.08813559322033898
Class 3.0 False Positive Rate: 0.1156586115379243
Class 3.0 Precision: 0.11158798283261803
```

```
Class 3.0 Recall: 0.08813559322033898
Class 3.0 F-Measure: 0.0984848484848485

Class 4.0 True Positive Rate: 0.033707865168539325
Class 4.0 False Positive Rate: 0.05834029518240044
Class 4.0 Precision: 0.0851528384279476
Class 4.0 Recall: 0.033707865168539325
Class 4.0 F-Measure: 0.04829721362229102

Class 5.0 True Positive Rate: 0.027548209366391185
Class 5.0 False Positive Rate: 0.005254170497832654
Class 5.0 Precision: 0.3333333333333333
Class 5.0 Recall: 0.027548209366391185
Class 5.0 F-Measure: 0.050890585241730284

Class 6.0 True Positive Rate: 0.12949640287769784
Class 6.0 False Positive Rate: 0.007195137079766778
Class 6.0 Precision: 0.3829787234042553
Class 6.0 Recall: 0.12949640287769784
Class 6.0 F-Measure: 0.19354838709677422
```

Critical Analysis and Discussion

Ideally, these evaluation metrics should all be closer to one excluding the false positive rate. Since the overall evaluation metrics for both models scored below 0.3, it means that the models performed very poorly in predicting the genre of the songs. To find the cause we can take a look at the evaluation metrics by label.

Looking at the evaluation metrics by label:

- Low precision shows that out of all the songs the model predicted, the label assigned was usually incorrect.
- Low recall means that out of all the songs of that genre, only a few of them were predicted correctly as the label's associated genre.
- Low f-measure shows that the data is imbalanced. Since it also combines precision and recall which also scored low, the model does a poor job of classifying the genre of the songs.

Out of all the labels, Class 2.0 (blues) had the lowest metrics out of them all for both models. For the logistic regression model, it scored all zeroes. This means that the model did not predict any of the blues songs as the right genre since true positives would be zero. This may have

contributed to the low precision of the other labels since it would predict it as the other genres and increase the false positives. This would also lower the accuracy of the other genres since it would decrease the true negative.

On the other hand, Class 0.0 (pop) had the highest true positive rate and false positive rate for both models. The models are more biased to the pop genre since it has more data samples. Increasing the threshold for the pop label would aid in reducing the amount of false positives.

One other reason for the low scores may be due to the four features we selected. While they were selected because of their values having greater differences between the genres, they were still too close. With the differences between the mean being either approximately 0.2, 0.1, or even 0.005. If the differences of the values between the different features are too small, the model will have a more difficult time distinguishing which genre it belongs to. For instance suppose the mean of the feature “music” for country and pop are both ~0.1. This will make it difficult for the model to predict whether a song with a “music” feature of 0.1 has a genre of country or pop.

After comparing the evaluation metrics for training data and test data, it is observed that there is a high training error and validation error (when the model is run against the training data it has low accuracy and precision). This means we have a high bias, which may be solved by adding more features to the model or by decreasing the regularization parameter.

Comparison between Decision Tree Model and Logistic Regression

The Decision Tree Model is a non-parametric supervised algorithm that can be used for both classification and regression tasks. Being a non-parametric model means that it makes no assumptions about the data, making it a great fit for our project since it allows us to not worry as

much as to whether we chose the “right” or “wrong” features.

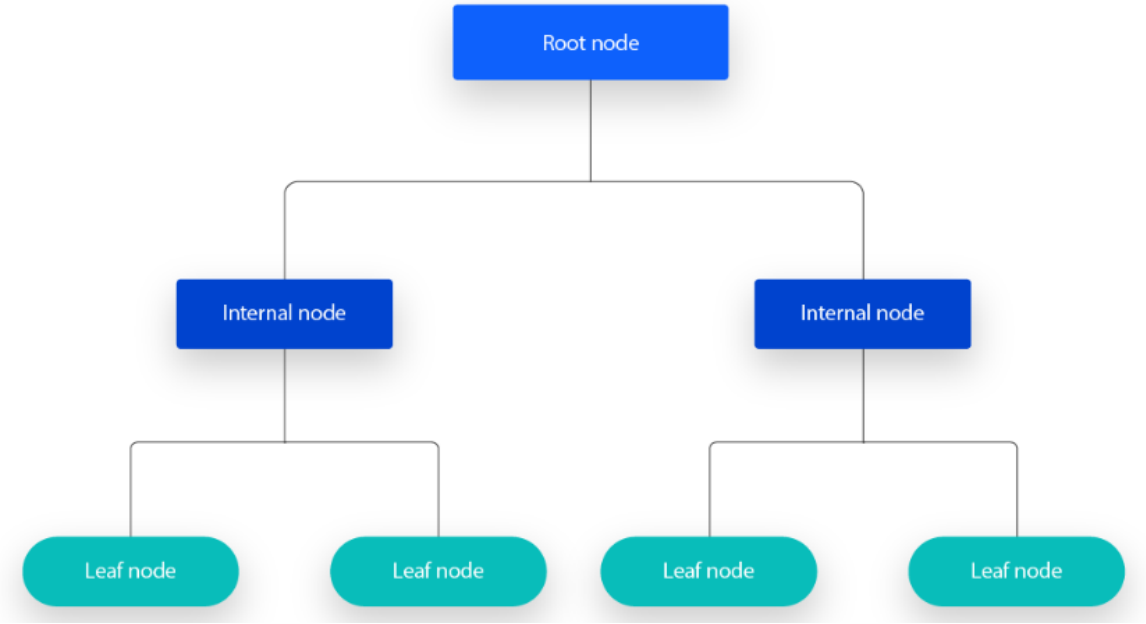


Figure 1: General Structure of a Decision Tree Model

Decision Tree models implement the divide and conquer strategy by conducting a greedy search, a problem solving approach for selecting the best possible option available at the moment, to identify the optimal split points in the tree. The splitting of each node in the tree is an interactive process, repeated from the top-down until most or all of the records have been classified.

Similar to the Decision Tree model, the Logistic Regression model is a model often used for classification. Logistic Regression is a parametric model that estimates the probability of an event occurring given a set of independent variables.

Both the Decision Tree and Logistic Regression models have their advantages and disadvantages over one another. The Logistic Regression model is better for binary classification and modeling data sets where there is a linear relationship between the classes and the features [2]. On the other hand Decision Tree models are better for handling multiclass classification problems and nonlinear relationships between the classes and features as a non-parametric model. Being a non-parametric model, Decision Trees are much more applicable to multiclass classification and various types of datasets in general. However, non-parametric models are prone to overfitting and require much more data to train properly. Decision Trees can also be biased towards a specific classifier if that classifier dominates the dataset. On the other hand, being a parametric model, Logistic Regression models make assumptions regarding the distribution and parameters (means, standard deviation, etc.) of the data. These assumptions allow Logistic Regression models to be more powerful and efficient at making accurate predictions. At the same time, the assumptions made by parametric models simplify the problem, making them less suitable for complex problems.

Within the context of our dataset, both the Decision Tree and Logistic Regression model performed relatively similar. The difference between the evaluation metrics of both models are too small for us to say that one was outright better than the other. On paper, the Decision Tree seems more suited to our dataset than a Logistic Regression model, since our data set has multiple classifiers and nonlinear relationships between the classifiers and the features of the data. But the evaluation metrics say otherwise. Overall we can conclude that the models are underfitted and should be further reiterated.

References

[1] https://www.kaggle.com/datasets/saurabhshahane/music-dataset-1950-to-2019/data
[2] https://gustavwillig.medium.com/decision-tree-vs-logistic-regression-1a40c58307d0