Machine Learning Nanodegree Capstone Project

Classifying Song Genres from Audio Data

Ahmad Alshehri – February 7th, 2019

I. Definition

Project Overview

In 2005, music streaming services were flung to the forefront of public attention with Pandora. By fusing the streamlined interface of iTunes with related musical characteristics, Pandora created an online service which recommended new music based on a user's listening history, allowing users to bookmark artists and discover new acts. While it certainly took a while to gain traction, Pandora influenced several modern streaming services including Spotify, and by 2013, the website had over 200 million users, demonstrating its influence on the modern world of streaming.

Well in recent years streaming services with huge catalogs have become the primary means through which most people listen to their favorite music. But at the same time, the sheer amount of music on offer can mean users might be a bit overwhelmed when trying to look for newer music that suits their tastes. Streaming services have been looking into means of categorizing music genres in order to allow for personalized recommendations systems for the users. One method involves direct analysis of the raw audio information in a given song, scoring the raw data on a variety of metrics.

“These songs recommendations are so on point! How does this playlist know me so well?” I have always asked myself this. Trying to answer this question, I started by preparing the meta data about the tracks that contains about 17,733 songs. After implementing some data cleaning and wrangling to prepare the set for the analysis, I used as a benchmark here. Benchmarking is the process of comparing your result to existing method or running a very simple machine learning model, just to confirm that your problem is actually 'solvable'. Therefore, a benchmark model needs to be run on the exact same dataset that is used in order to see whether the results are comparable to the final optimized model's results.

Problem Statement

The goal of this project is to distinguish between rock and Hip-hop music based only on the track information. A song is about more than its title, artist, and number of listens. We have another dataset that has musical features of each track such as energy, danceability, and acousticness on a scale from -1 to 1. In this project, I will be using a dataset comprised of songs of two music genres Hip-Hop and Rock. I will train a classifier to distinguish between the two genres based only on track information derived from The Echo Nest (now part of Spotify). These exist in two different files, which are in different formats - CSV and JSON. While CSV is a popular file format for denoting tabular data, JSON is another common file format in which databases often return the results of a given query.

My hypothesis is to first merge the two datasets together by the track ID and remove any irrelevant information my model before starting my analysis. The solution to this problem is to apply a supervised learning model with labels as the target variable. Though, having too many features could lead to overfitting, I will apply the following techniques before choosing my supervised learning model of choice: Data Preprocessing, Dimensionality Reduction, and Principle Components Analysis

This seems to me as a binary classification problem since the songs are either one of the two different categories (Rock and Hip-Hop). My solution approach will be mainly supervised learning, my models of choice are Logistic Regression classifier. A simple decision tree classifier model will be used as a bench mark model that the logistic regression model performance can be measured with.

Metrics

Looking at the row data, it seems Rock genre have the high percentage of the data. Therefore, some data balancing might be required to avoid creating a biased model. Since the dataset is unbalanced, the appropriate metric to evaluate the performance of this classifier is by calculating the Mean F1 Score. The F1 score can be interpreted as a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0. To relate to the contribution of precision and recall to the F1 score. The formula for the F1 score is given as follows:

F1 = 2*(precision * recall) / (precision + recall)

Precision and recall can be calculated as:

Precision = number of TP / (number of TP + number of FP)

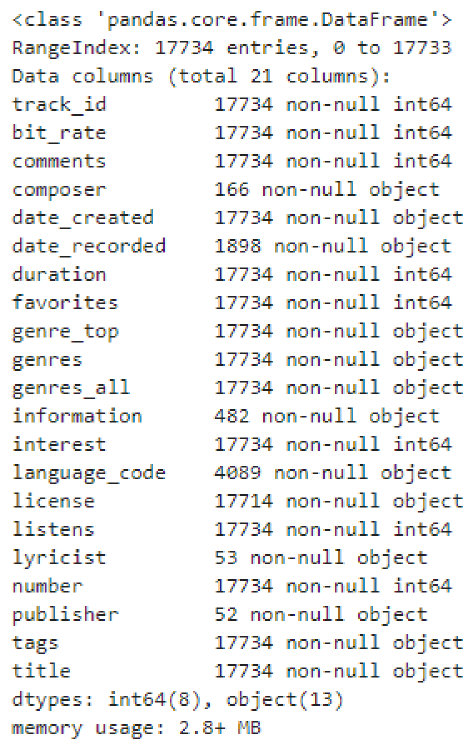
Recall = number of TP / (number of TP + number of FN)

Where, TP is True Positives, FP is False Positives and FN is False Negatives.

II. Analysis

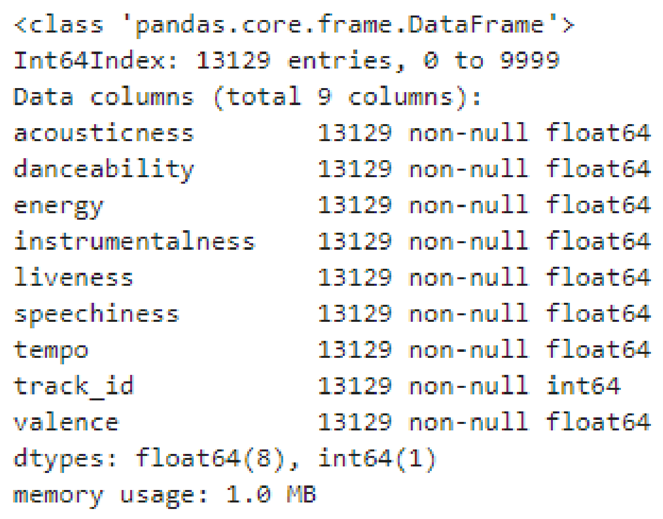
Data Exploration

Datasets obtained from kaggle. Both datasets contain important information regarding each song. Starting by creating pandas DataFrames out of these files so that its features and labels be can merged for classification later on. The (fma_Rock_HipHop) data were loaded to the (tracks) label and below is a description of the data.



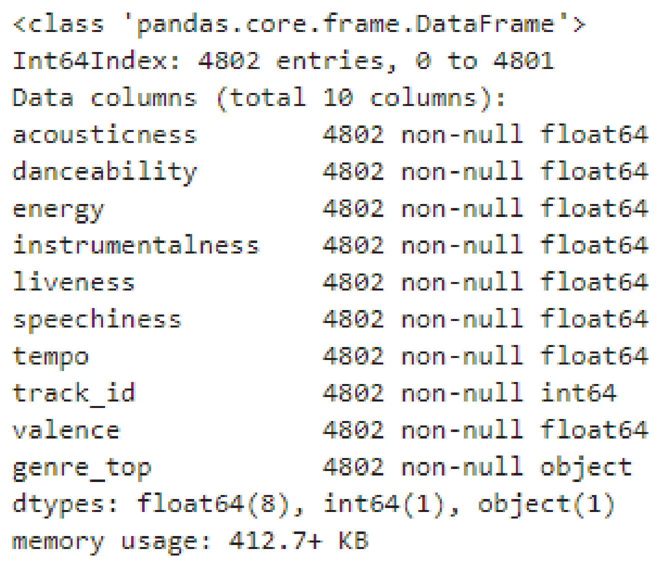
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17734 entries, 0 to 17733
Data columns (total 21 columns):
track_id         17734 non-null int64
bit_rate         17734 non-null int64
comments         17734 non-null int64
composer         166 non-null object
date_created     17734 non-null object
date_recorded    1898 non-null object
duration         17734 non-null int64
favorites        17734 non-null int64
genre_top        17734 non-null object
genres           17734 non-null object
genres_all       17734 non-null object
information      482 non-null object
interest         17734 non-null int64
language_code    4089 non-null object
license          17714 non-null object
listens          17734 non-null int64
lyricist         53 non-null object
number           17734 non-null int64
publisher        52 non-null object
tags             17734 non-null object
title            17734 non-null object
dtypes: int64(8), object(13)
memory usage: 2.8+ MB
```

Likewise, the (econest-metrics) data were loaded to the (econest-metrics) label. And,below is a description of the data.



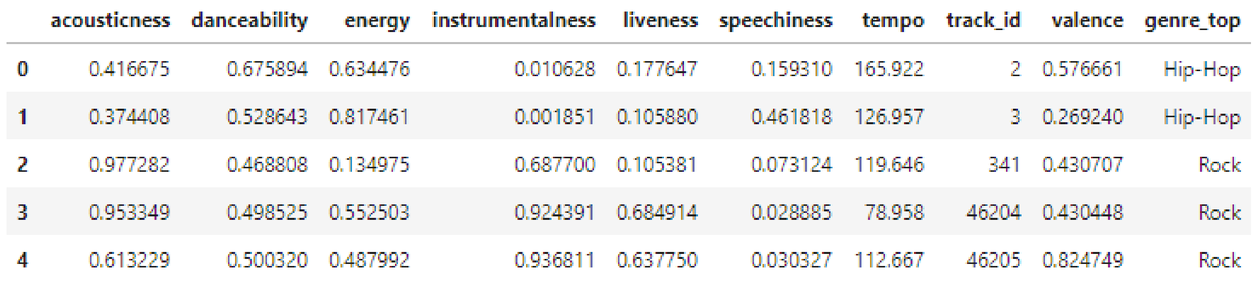
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 13129 entries, 0 to 9999
Data columns (total 9 columns):
acousticness        13129 non-null float64
danceability        13129 non-null float64
energy              13129 non-null float64
instrumentalness    13129 non-null float64
liveness            13129 non-null float64
speechiness         13129 non-null float64
tempo               13129 non-null float64
track_id            13129 non-null int64
valence             13129 non-null float64
dtypes: float64(8), int64(1)
memory usage: 1.0 MB
```

After merging relevant columns of tracks and econest_metrics, we ended up with the dataframe (echo-tracks)



```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4802 entries, 0 to 4801
Data columns (total 10 columns):
acousticness        4802 non-null float64
danceability        4802 non-null float64
energy              4802 non-null float64
instrumentalness    4802 non-null float64
liveness            4802 non-null float64
speechiness         4802 non-null float64
tempo               4802 non-null float64
track_id            4802 non-null int64
valence             4802 non-null float64
genre_top           4802 non-null object
dtypes: float64(8), int64(1), object(1)
memory usage: 412.7+ KB
```

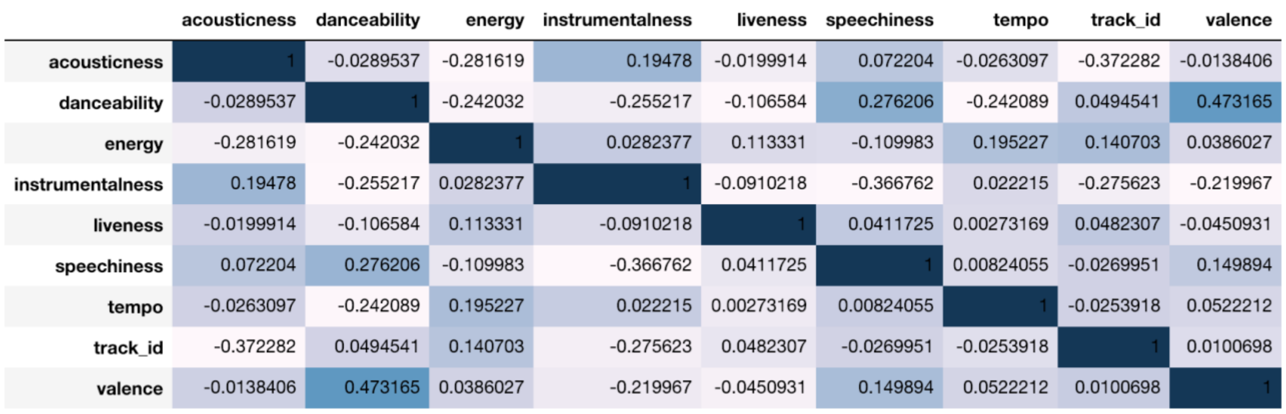
For now, we have our target variable (genre_top) that can be identified by the below features.



	acousticness	danceability	energy	instrumentalness	liveness	speechiness	tempo	track_id	valence	genre_top
0	0.416675	0.675894	0.634476	0.010628	0.177647	0.159310	165.922	2	0.576661	Hip-Hop
1	0.374408	0.528643	0.817461	0.001851	0.105880	0.461818	126.957	3	0.269240	Hip-Hop
2	0.977282	0.468808	0.134975	0.687700	0.105381	0.073124	119.646	341	0.430707	Rock
3	0.953349	0.498525	0.552503	0.924391	0.684914	0.028885	78.958	46204	0.430448	Rock
4	0.613229	0.500320	0.487992	0.936811	0.637750	0.030327	112.667	46205	0.824749	Rock

Pairwise relationships between continuous variables

Starting with pairwise relationship between continuous variables. We typically want to avoid using variables that have strong correlations with each other. Hence, avoiding feature redundancy for two reasons. First, to keep the model simple and improve interpretability; with many features, we run the risk of overfitting. Second, when our datasets are very large, using fewer features can drastically speed up our computation time. To get a sense of whether there are any strongly correlated features in our data, the following correlation metrics was built.



	acousticness	danceability	energy	instrumentalness	liveness	speechiness	tempo	track_id	valence
acousticness	1	-0.0289537	-0.281619	0.19478	-0.0199914	0.072204	-0.0263097	-0.372282	-0.0138406
danceability	-0.0289537	1	-0.242032	-0.255217	-0.106584	0.276206	-0.242089	0.0494541	0.473165
energy	-0.281619	-0.242032	1	0.0282377	0.113331	-0.109983	0.195227	0.140703	0.0386027
instrumentalness	0.19478	-0.255217	0.0282377	1	-0.0910218	-0.366762	0.022215	-0.275623	-0.219967
liveness	-0.0199914	-0.106584	0.113331	-0.0910218	1	0.0411725	0.00273169	0.0482307	-0.0450931
speechiness	0.072204	0.276206	-0.109983	-0.366762	0.0411725	1	0.00824055	-0.0269951	0.149894
tempo	-0.0263097	-0.242089	0.195227	0.022215	0.00273169	0.00824055	1	-0.0253918	0.0522212
track_id	-0.372282	0.0494541	0.140703	-0.275623	0.0482307	-0.0269951	-0.0253918	1	0.0100698
valence	-0.0138406	0.473165	0.0386027	-0.219967	-0.0450931	0.149894	0.0522212	0.0100698	1

Normalizing the feature data

As mentioned earlier, it can be particularly useful to simplify our models and use as few features as necessary to achieve the best result. Since we didn't find any particular strong correlations between our features, we can instead use a common approach to reduce the number of features called principal component analysis (PCA).

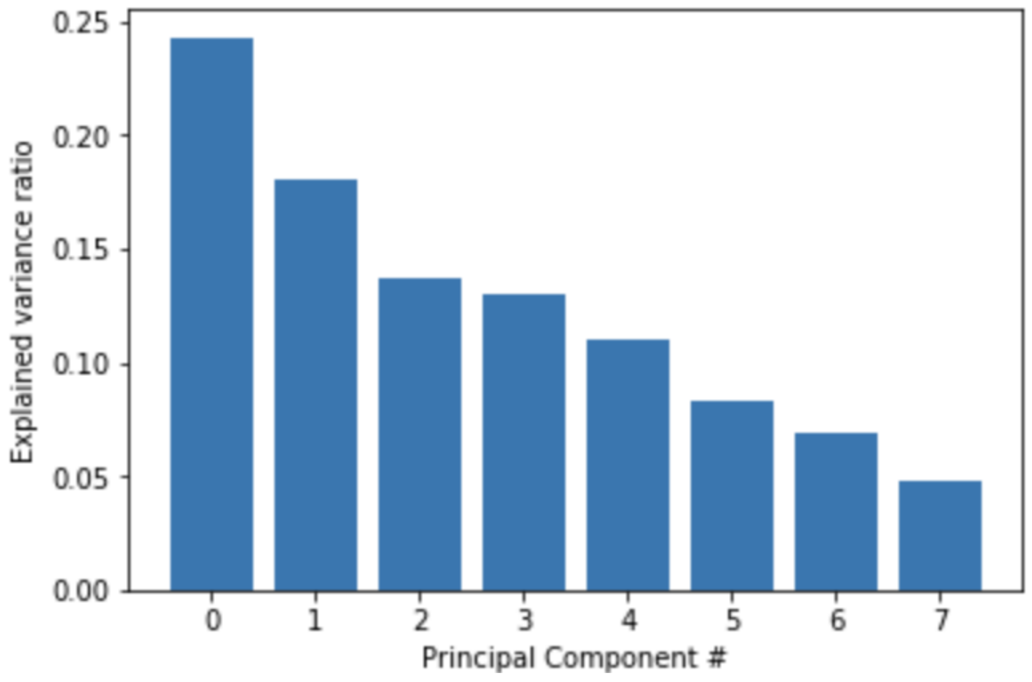
It is possible that the variance between genres can be explained by just a few features in the dataset. PCA rotates the data along the axis of highest variance, thus allowing us to determine the relative contribution of each feature of our data towards the variance between classes.

However, since PCA uses the absolute variance of a feature to rotate the data, a feature with a broader range of values will overpower and bias the algorithm relative to the other features. To avoid this, we must first normalize our data. There are a few methods to do this, but a common way is through standardization, such that all features have a mean = 0 and standard deviation = 1 (the resultant is a z-score).

Principal Component Analysis on our scaled data

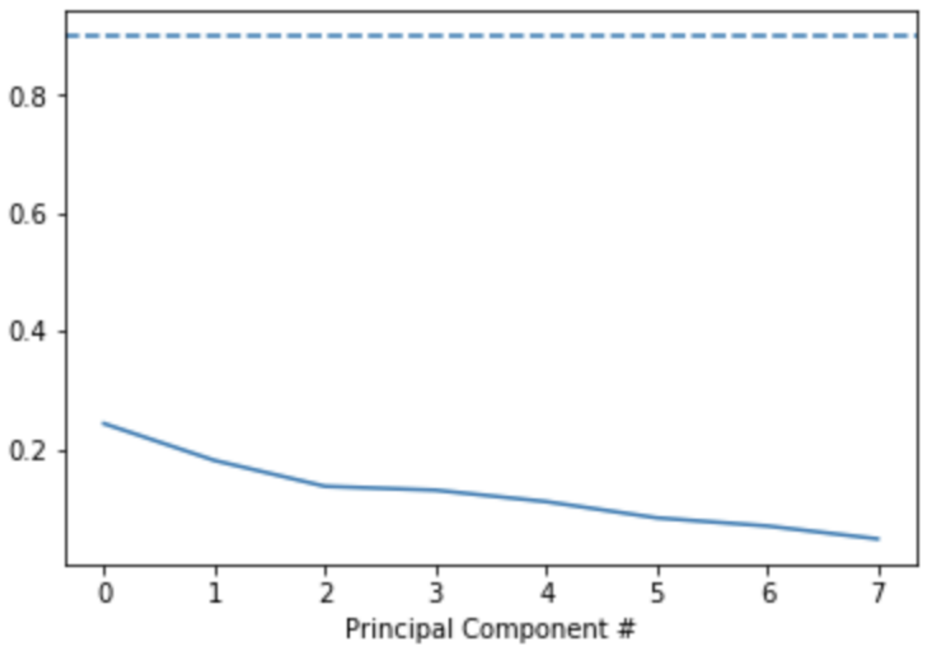
Now that we have preprocessed our data, we are ready to use PCA to determine by how much we can reduce the dimensionality of our data. We can use scree-plots and cumulative explained ratio plots to find the number of components to use in further analyses.

Scree-plots display the number of components against the variance explained by each component, sorted in descending order of variance. Scree-plots help us get a better sense of which components explain a sufficient amount of variance in our data. When using scree plots, an 'elbow' (a steep drop from one data point to the next) in the plot is typically used to decide on an appropriate cutoff.



Further visualization of PCA

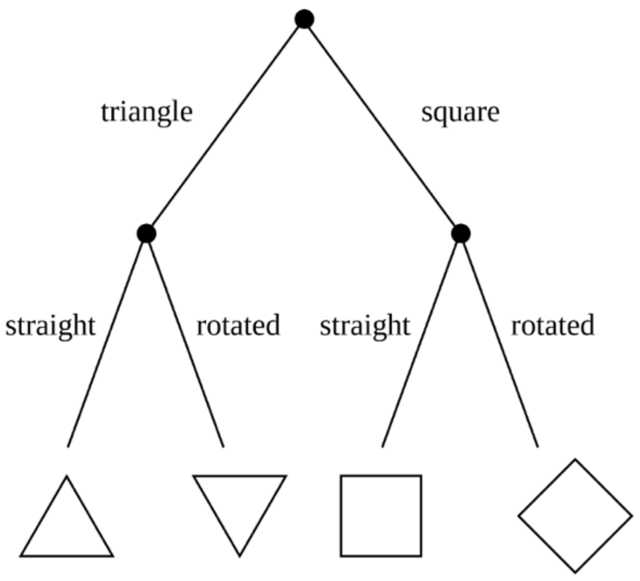
Unfortunately, there does not appear to be a clear elbow in this scree plot, which means it is not straightforward to find the number of intrinsic dimensions using this method. But all is not lost! Instead, we can also look at the cumulative explained variance plot to determine how many features are required to explain, say, about 90% of the variance (cutoffs are somewhat arbitrary here, and usually decided upon by 'rules of thumb'). Once we determine the appropriate number of components, we can perform PCA with that many components, ideally reducing the dimensionality of our data.



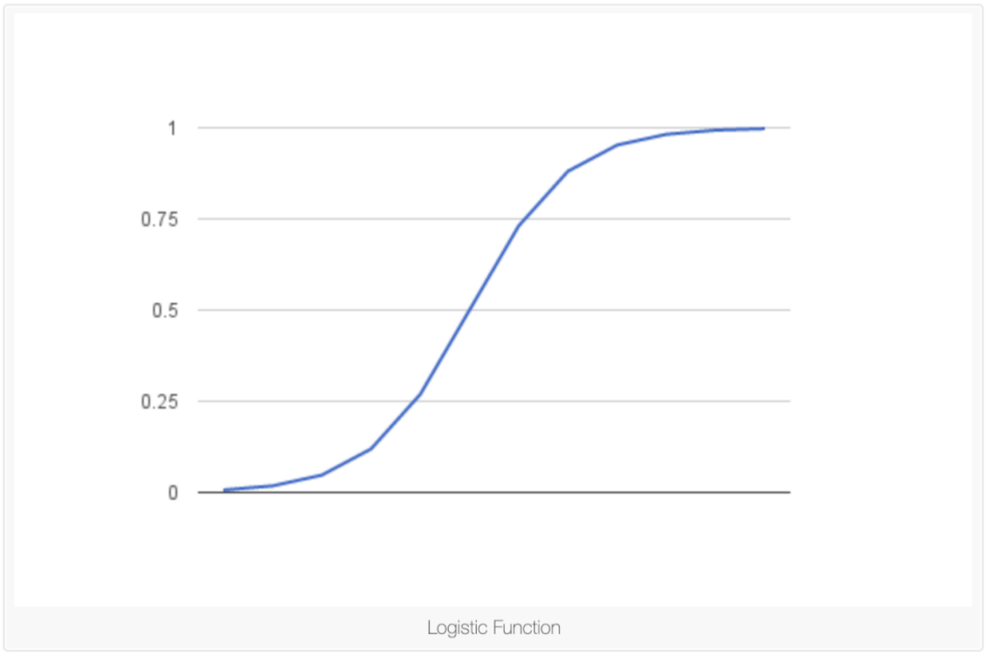
Algorithms and Techniques

The classification technique is a systematic approach to build classification models from an input dataset. In this project case, decision tree classifiers and logistic regression which are different technique to solve a classification problem. Each technique adopts a learning algorithm to identify a model that best fits the relationship between the attribute set and class label of the input data. Therefore, a key objective of the learning algorithm is to build predictive model that accurately predict the class labels of previously unknown records.

Decision Tree Classifier is a simple and widely used classification technique. It applies a straightforward idea to solve the classification problem. Decision Tree Classifier poses a series of carefully crafted questions about the attributes of the test record. Each time it receives an answer, a follow-up question is asked until a conclusion about the class label of the record is reached.

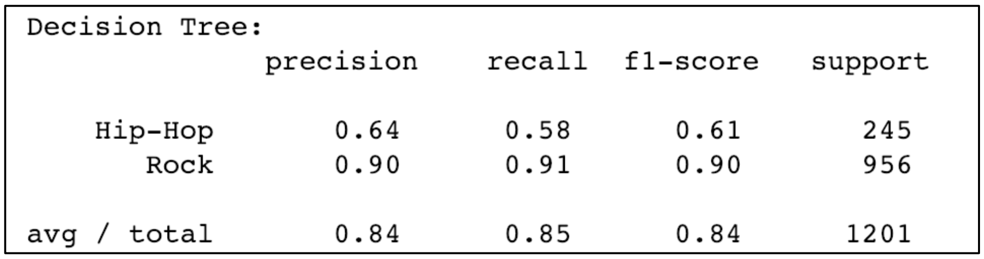


The logistic function, also called the sigmoid function was developed by statisticians to describe properties of population growth in ecology, rising quickly and maxing out at the carrying capacity of the environment. It’s an S-shaped curve that can take any real-valued number and map it into a value between 0 and 1, but never exactly at those limits.



Benchmark Model

AS previously mentioned, Precision, Recall, and F1-score are going to be used as an evaluation metric to verify if our model can be improved. After applying the benchmark model, we ended up with the results below:



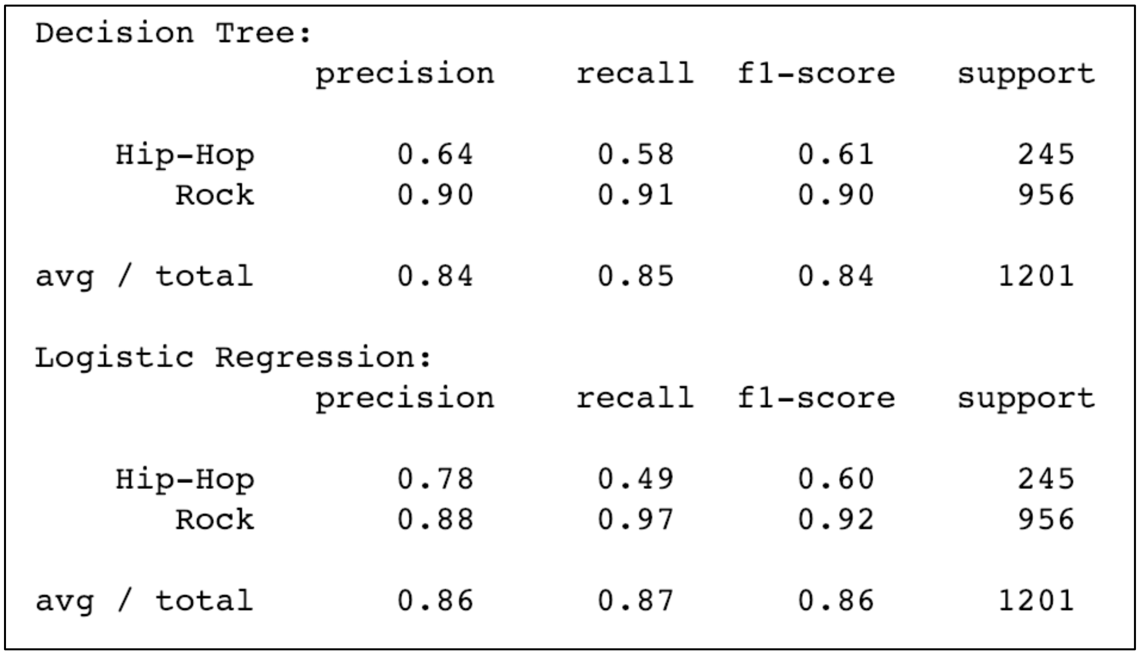
```
Decision Tree:
              precision    recall  f1-score   support

    Hip-Hop       0.64      0.58      0.61       245
       Rock       0.90      0.91      0.90       956

avg / total       0.84      0.85      0.84      1201
```

III. Methodology

Data Preprocessing

Although our tree's performance is decent, it's a bad idea to immediately assume that it's therefore the perfect tool for this job -- there's always the possibility of other models that will perform even better! It's always a worthwhile idea to at least test a few other algorithms and find the one that's best for our data.

Sometimes simplest is best, and so we will start by applying logistic regression. Logistic regression makes use of what's called the logistic function to calculate the odds that a given data point belongs to a given class. Once we have both models, we can compare them on a few performance metrics, such as false positive and false negative rate (or how many points are inaccurately classified).



```
Decision Tree:
               precision    recall  f1-score   support

    Hip-Hop       0.64      0.58      0.61       245
       Rock       0.90      0.91      0.90       956

avg / total       0.84      0.85      0.84      1201

Logistic Regression:
               precision    recall  f1-score   support

    Hip-Hop       0.78      0.49      0.60       245
       Rock       0.88      0.97      0.92       956

avg / total       0.86      0.87      0.86      1201
```

Implementation

In order to balance our data for greater performance. Both our models do similarly well, boasting an average precision of 85% each. However, looking at our classification report, we can see that rock songs are fairly well classified, but hip-hop songs are disproportionately misclassified as rock songs.

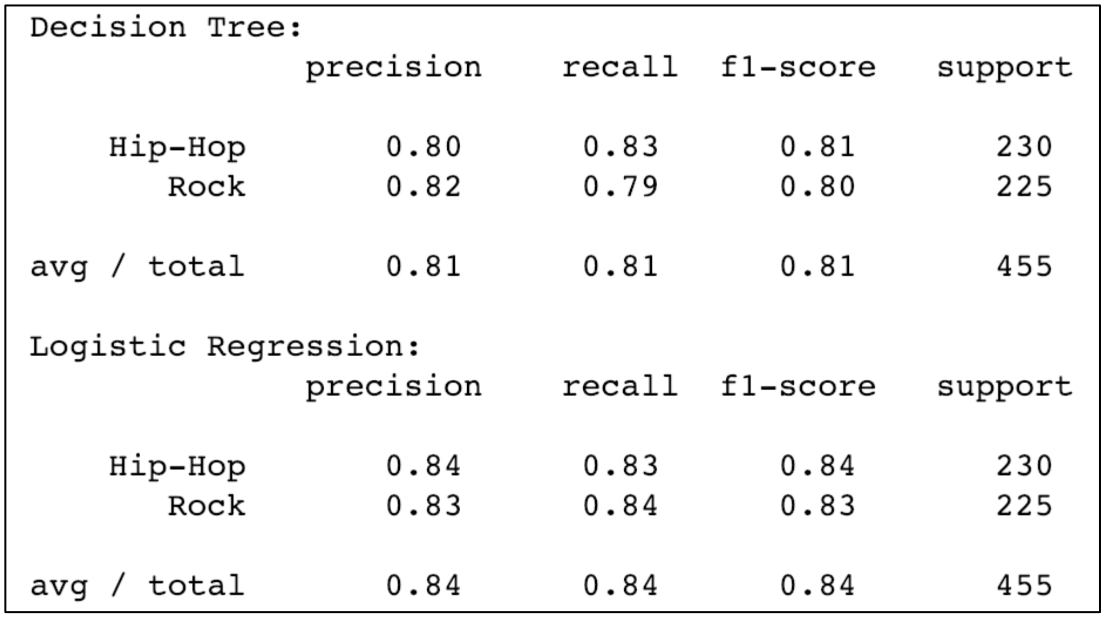
Why might this be the case? Well, just by looking at the number of data points we have for each class, we see that we have far more data points for the rock classification than for hip-hop, potentially skewing our model's ability to distinguish between classes. This also tells us that most of our model's accuracy is driven by its ability to classify just rock songs, which is less than ideal.

To account for this, we can weigh the value of a correct classification in each class inversely to the occurrence of data points for each class. Since a correct classification for "Rock" is not more important than a correct classification for "Hip-Hop" (and vice versa), we only need to account for differences in sample size of our data points when weighting our classes here, and not relative importance of each class.

Refinement

We've now balanced our dataset, but in doing so, we've removed a lot of data points that might have been crucial to training our models. Let's test to see if balancing our data improves model bias towards the "Rock" classification while retaining overall classification performance.

Note that we have already reduced the size of our dataset and will go forward without applying any dimensionality reduction. In practice, we would consider dimensionality reduction more rigorously when dealing with vastly large datasets and when computation times become prohibitively large.



```
Decision Tree:
              precision    recall  f1-score   support

    Hip-Hop       0.80      0.83      0.81       230
       Rock       0.82      0.79      0.80       225

avg / total       0.81      0.81      0.81       455

Logistic Regression:
              precision    recall  f1-score   support

    Hip-Hop       0.84      0.83      0.84       230
       Rock       0.83      0.84      0.83       225

avg / total       0.84      0.84      0.84       455
```

IV. Results

Model Evaluation and Validation

Success! Balancing our data has removed bias towards the more prevalent class. To get a good sense of how well our models are actually performing, we can apply what's called cross-validation (CV). This step allows us to compare models in a more rigorous fashion.

Since the way our data is split into train and test sets can impact model performance, CV attempts to split the data multiple ways and test the model on each of the splits. Although there are many different CV methods, all with their own advantages and disadvantages, we will use what's known as K-fold

CV here. K-fold first splits the data into K different, equally sized subsets. Then, it iteratively uses each subset as a test set while using the remainder of the data as train sets. Finally, we can then aggregate the results from each fold for a final model performance score.

Decision Tree: 0.7719780219780219

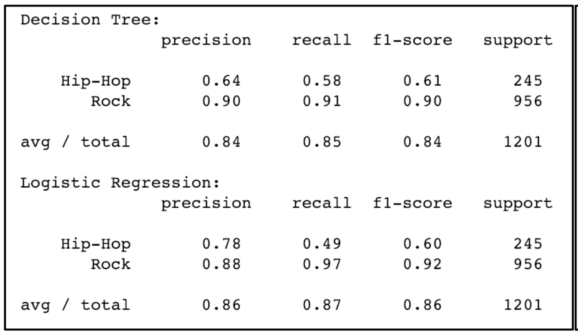
Logistic Regression: 0.7901098901098901

Justification

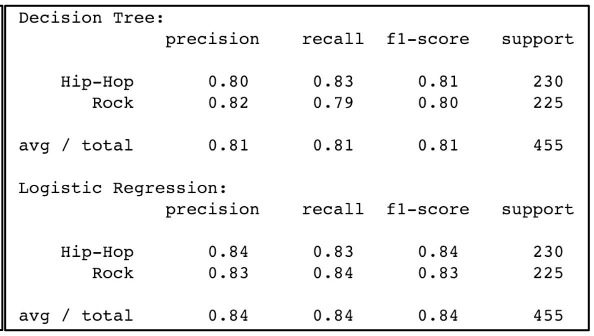
Initially looking at the results of our model and K-Fold Cross Validation, we might not see how the models have improved. However, the bench mark model was biased toward Rock songs which might affected the average scores. Given that, I believe that the logistic regression classifier has performed very well with an F1-Score of 0.84.

Benchmark model:

Resulted balanced model:



```
Decision Tree:
                 precision    recall  f1-score   support

     Hip-Hop       0.64      0.58      0.61       245
        Rock       0.90      0.91      0.90       956

avg / total       0.84      0.85      0.84      1201

Logistic Regression:
                 precision    recall  f1-score   support

     Hip-Hop       0.78      0.49      0.60       245
        Rock       0.88      0.97      0.92       956

avg / total       0.86      0.87      0.86      1201
```



```
Decision Tree:
                 precision    recall  f1-score   support

     Hip-Hop       0.80      0.83      0.81       230
        Rock       0.82      0.79      0.80       225

avg / total       0.81      0.81      0.81       455

Logistic Regression:
                 precision    recall  f1-score   support

     Hip-Hop       0.84      0.83      0.84       230
        Rock       0.83      0.84      0.83       225

avg / total       0.84      0.84      0.84       455
```

V. Conclusion

In conclusion, the logistic regression classifier performed very well for a simple model. One question to ask, is how well other advanced models such as random forest or AdaBoost classifier might perform. However, these advanced models come with a price such as higher load. For the sake of the project and keeping things simple I can say that I'm satisfied with the logistic regression classifier.

Reflection

Personally, I listen to music all the time weather its work, study, gym, even while I'm writing this report. With new songs coming every day, I don't really have a perfect genre, I mostly play these already made up list based on my history. Surprisingly, I rarely have to skip a song. In recent years, streaming services with huge catalogs have become the primary means through which most people listen to their favorite music. But at the same time, the sheer amount of music on offer can mean users might be a bit overwhelmed when trying to look for newer music that suits their tastes

The goal of this project is to distinguish between rock and Hip-hop music based only on the track information. A song is about more than its title, artist, and number of listens. We have another dataset that has musical features of each track such as energy, danceability, and acousticness on a scale from -1 to 1. My hypothesis was to first merge the two datasets together by the track ID and remove any irrelevant information my model before starting my analysis. This took me a bit to figure out. Regarding the model performance, again, I believe it performed very well given that the it was easy to implement once scaling and balancing the data was performed.

Improvement

One improvement I would love to work on if had the time on this is to add more visualization to explain the data. Sadly, I'm only an expert when it comes to listing to music only. Also, A larger dataset could be useful on the long term. There is one competition on Kaggle called One Million Songs Challenge. Hopefully, I will try to work on future work.

References

- For the full code please look into the notebook attached in the (.zip) file.
- https://machinelearningmastery.com/logistic-regression-for-machine-learning/
- http://mines.humanoriented.com/classes/2010/fall/csci568/portfolio_exports/lguo/decisionTree.html
- FMA: A Dataset for Music Analysis : https://arxiv.org/pdf/1612.01840.pdf
- https://www.kaggle.com/aniruddhachoudhury/classify-song-genres-from-audio-data#echonest-metrics.json