# Group Project

Course: CS 488

Sem: Spring 2022

Stage: 4

Group: Kerry Forsythe,

Roderick Hutterer,

David Torres

## Motivation

Data is fundamental to the operation of modern cloud-based services. In addition to organizing their catalogs of products or materials, companies have the opportunity to collect data from their users’ interactions with the app. Understanding who, what, when, where, and why can help a company better serve their customers through analysis such as classifying products, identifying purchase trends, identifying patterns in users’ past behavior, predict their future behavior, identify commonalities between users, and recommending products and services.

## Problem

In this project, we propose to data mine a sub-set of the Spotify music catalog and user data to provide descriptive and predictive analytics. Spotify songs include attributes such as the following:

* Beats Per Minute (BPM) — The tempo of the song.
* Energy — The energy of a song, the higher the value, the more energetic.
* Danceability — The higher the value, the easier it is to dance to this song.
* Loudness — The higher the value, the louder the song (in dB).
* Valence — The higher the value, the more positive mood for the song.
* Length — The duration of the song.
* Acousticness — The higher the value the more acoustic the song is.
* Release Year — The year each song was released.
* Popularity — The higher the value the more popular the song is.

Questions:

Do certain characteristics of a song predict popularity? Using a **classification** technique, we can construct a training and testing data set to model and predict song popularity.

What other songs might a user like? Using **clustering**, we can try to identify songs with similar characteristics to recommend to a user.

**Sof**t**ware & Datasets**

For our dataset, we have a few different selections of data to use as training data and testing data. Our first possible option would be to download a dataset from Kaggle.com. There are numerous datasets that have been collected from Spotify, consisting of statistics from many different songs, where each song is an instance in their respective dataset. These options, however, may be limiting to how we can analyze the data. Many of these sets tend to be specialized, such as consisting of strictly unpopular songs or representing only one or two genres of music. Because of these limitations, it would be optimal to use a software package that can collect data for us.

The Spotipy Python package is a useful Python plugin that collects information on Spotify songs by using a URL as input. This could be useful for our project since it allows us to create our own datasets however we wish. We could easily collect a wide variety of songs from all different genres, regardless of popularity. We could choose specific attributes of songs to record as well, such as mode, key, runtime, genre, number of listens, etc. This tool provides us with a large range of dataset options for our project.

## Techniques

After collecting our dataset(s), we would most likely pre-process the data to eliminate noise using means and variances to single out instances that would greatly differ from the rest of the set. Additionally, we would use general statistics of songs and histograms of multiple attributes in order to identify grouping and similarities that we could explore when training a training model. Seeing as a Spotify dataset will have a lot of continuous data, we might even use linear/logistic regression techniques to help with our model. A clustering analysis of different attributes would help us find close correlations that we could exploit when training the data. While developing a decision tree would be ideal, it may not be feasible with such an expansive dataset that has so much continuous data. During the early parts of this project, we will attempt to see how and if we can model a decision tree.

## Data Collection

To collect the data, [Spotipy](https://spotipy.readthedocs.io/en/master/), a lightweight Python library for the Spotify Web API was utilized. After installing Spotipy and obtaining authentication ids from [Spotify for Developers](https://developer.spotify.com/), code was written in Jupyter Notebooks to connect and collect tracks from Spotify. The team explored several methods of collecting tracks including using popular words from songs, using billboard charts, and using user playlists, before settling on a stratified genre and year selection. Using Spotify search(), the Spotify song database was queried to find songs that met the following criteria:

* Genre – the track belonged to one of the 8 specified genres – pop, rock, rap, country, classical, jazz, techno, and world.
* Year – the track belonged to the specified year – 2012-2022
* Market – the track was available to the “US”

A maximum of 1000 distinct tracks were selected for each genre and year. Ten years (2012-2022) of tracks were collected in total producing up to 10,000 observations per genre and a total of ﻿79,379 records:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Classical | Country | Jazz | Pop | Rap | Rock | Techno | World | **Total** |
| # | 9403 | 10000 | 9994 | 10000 | 10000 | 10000 | 9986 | 9996 | **79379** |

For each of these tracks, the Spotipy audio\_features() method was used to collect audio attributes including genre, danceability, energy, key, loudness, mode, speechiness, acousticness, instrumentalness, liveness, valence, tempo, and time signature. After extracting some data from the album information, some initial columns were dropped from the data, and each genre was saved to a CSV file. These files serve as a large dataset of tracks that can be randomly sampled to create representative samples for further analysis.

## Summary Statistics

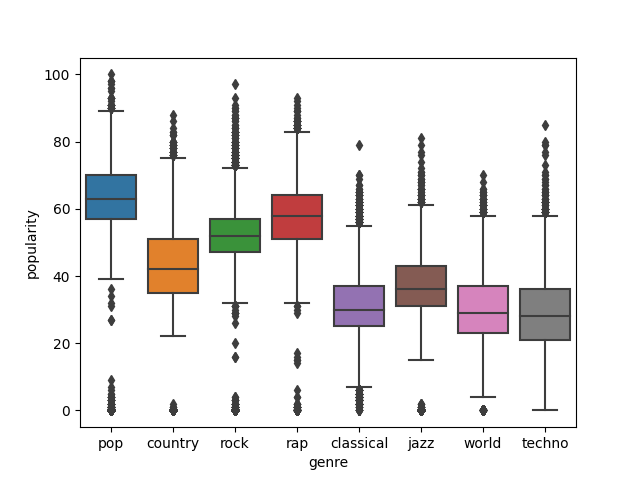
The following statistics on some of the more prominent and significant attributes in the dataset provide a basis on which to compare and evaluate the different genres.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | popularity | danceability | energy | speechiness |
| mean | 42.489865 | 0.564586 | 0.565899 | 0.080097 |
| std | 16.983005 | 0.180643 | 0.265151 | 0.088723 |
| min | 0 | 0.000000 | 0 | 0.000000 |
| max | 100 | 0.987000 | 1 | 0.955000 |

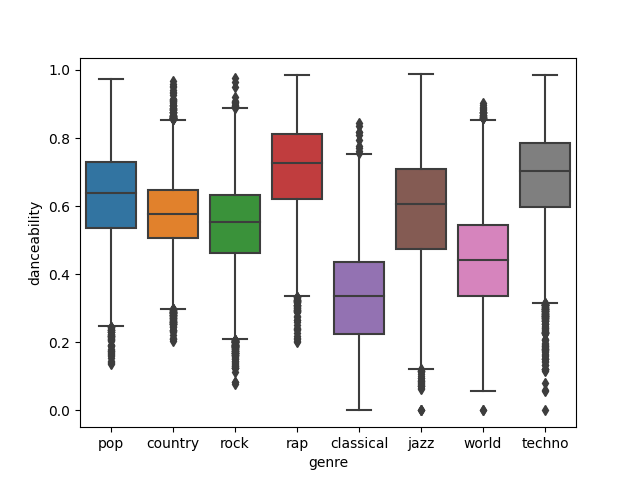
Danceability and energy are very similar, as expected, and will likely be good metrics for determining which songs belong in the pop, rap, techno, and jazz genres. Speechiness, unexpectedly, has a very low mean indicating that it is likely that a significant number of songs have average or minimal vocals.

## Data Preprocessing

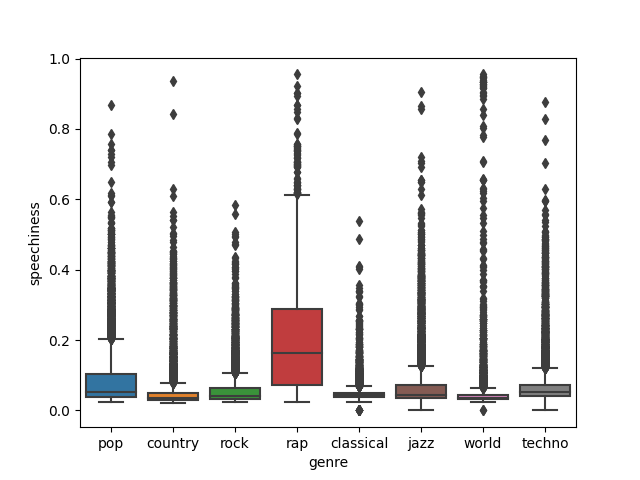
The analysis started with the eight different datasets with about 10,000 songs each. Then, they were all compiled into one large dataset while removing the redundant information, such as disc number, id, track number, album type, album name, is playable, and linked from. Next, various histograms and boxplots were created to try to find the most interesting information about the data. For instance, we found that pop music tends to be the most popular out of the eight chosen genres as shown below:



As expected, the least popular would be classical, techno, and world music. So far, the data that was collected seems to be indicative of current music. Additionally, we found that the danceable music is rap, pop, jazz, and techno:



Another important correlation we found (albeit a rather obvious one) was that rap music tends to be the most vocal type, while classical music is the least as shown below:



We expected the rap and classical genres to be the most distinct from one another among all the genres. The boxplots shown so far have all supported this notion and the same can be seen from looking closely at the summary statistics of the two genres (shown below).

*(danceability and energy of rap)*

Graphical user interface, text

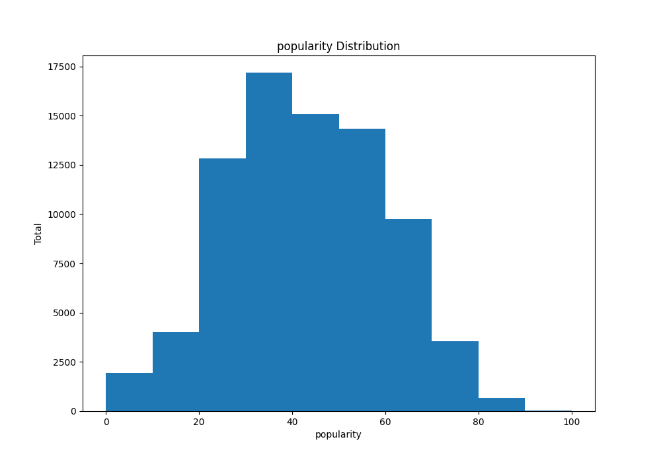
Description automatically generated

*(danceability and energy of classical)*

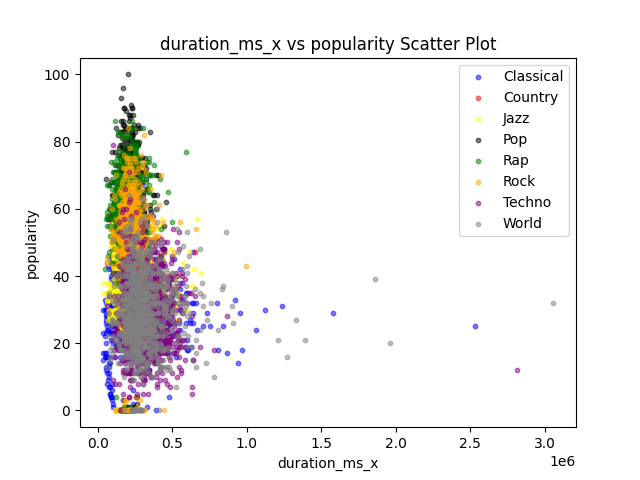
Graphical user interface, text, chat or text message

Description automatically generated

These boxplots show a distinctness between these genres of music, indicating that it may be possible to predict the genres of Spotify tracks. Another important piece of information is that for all eight genres, popularity is most dense in the 30-60 range (on a scale of 0-100) while still having a good amount of variability:

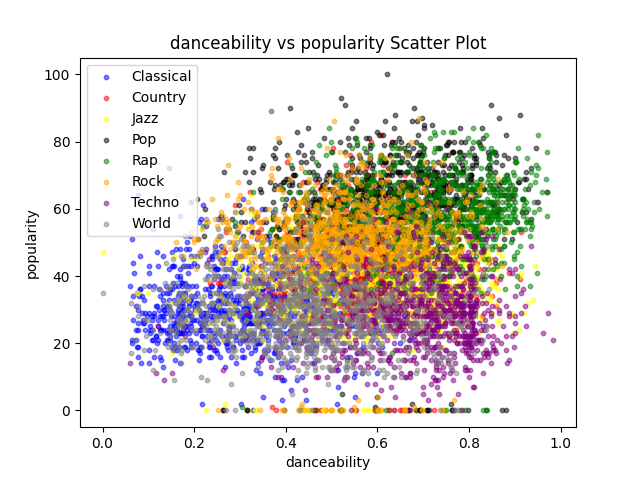
This indicates that our sample data may be useful enough to predict popularity based on all other attributes.

After plotting these initial graphs, a Pearson correlation matrix was created using the attributes of the songs. Some of the highest correlations were between: (popularity, explicit), (popularity, instrumentalness), (valence, danceability), (acousticness, loudness), (genre, duration), (genre, energy), (genre, loudness), and (genre, acousticness). After this analysis, a stratified sample of about 8,000 songs was taken from the dataset (grouped by popularity). This was used to generate scatter plots in order to find any correlation between popularity and genre. One of the correlations found was techno tends to have longer duration and lower popularity than rock music:

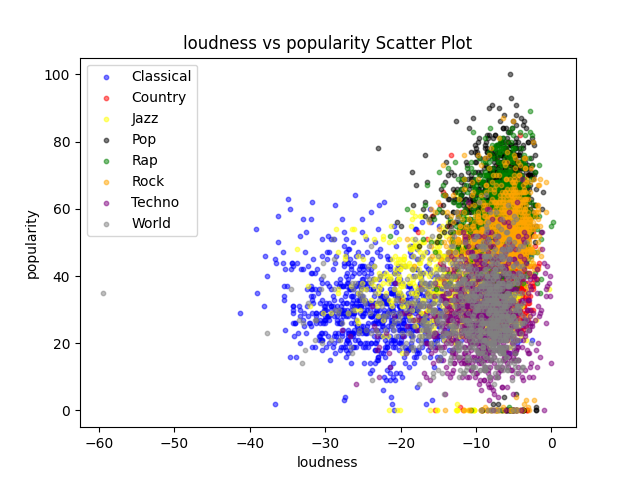


(Make sure to zoom in for clarity)

It can also be noticed that classical has much lower popularity and danceability than rap music:



Another good point to notice is that loudness is more closely related to genre than popularity. In the plot below, it is shown that even though pop, rock, rap, techno and world music generally have the same loudness, but techno and world music have much lower popularity:



With many more scatter plots that visualize similarity and dissimilarity between popularity, genres, and their attributes, we will be able to train and run models using all of this important information. These scatter plots show once more that our data is concrete and that the genres are distinct enough to develop training models with good accuracies that may predict genre or popularity.

**Proposed Solution**

In order to predict the popularity of songs based on their song features and attributes, we will first use the average popularity to determine what is popular. In other words, anything with less than the average popularity will be said to be unpopular, and popular otherwise. Once this value is chosen, we will transform the popularity column in the dataset to be a binary classification, unpopular and popular. Having this new dataset, we will train models to try to accurately predict any song’s popularity classification. The two main models we will focus on first will be the Logistic Regression Model and the Naïve Bayesian Classifier. These two models are ideal since we have so many attributes in the dataset. Though, it is likely that the Bayesian model will outperform the logistic model due to its insensitivity to noise. In the end, we will choose the model with the best statistics (accuracy, precision, etc.). If neither of these models provide desirable results, we will explore other types of classification models and algorithms to achieve our desired results (which would be an accuracy of about 80%).

## Classification Exploration

Ten Classification Algorithms were run using 5-fold cross-validation on a genre-stratified sample of 10% of the full dataset for two binary splitting choices for popularity. Using a splitting choice of Popular = 1 if popularity >= mean (popularity), Bagging, AdaBoost, and Random Forest produced the best accuracy results with 85-87% accuracy. The other 7 models ranged from 55-80% accuracy.

Using a second splitting choice of popular =1 if the popularity >= mean(popularity) + standard deviation(popularity) improved the performance for 9 out of 10 models. Bagging, AdaBoost, and Random Forest remained the 3 most accurate models with 2-3% increases in accuracy to 87-88%. The other 7 models' accuracy ranges from 74-83% accuracy with increases from -3% to 18%.

*Table: Accuracy 10 models, 2 splitting points*

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Accuracy** | | |
| **Classifier** | **pop>=mean** | **pop>=mean+std** | **improvement** |
| **Bagging** | 0.848829 | 0.867977 | 0.019148 |
| **AdaBoost** | 0.843536 | 0.874024 | 0.030488 |
| **Random Forest** | 0.861299 | 0.880322 | 0.019023 |
| **Logistic Regression** | 0.661376 | 0.824642 | 0.163266 |
| **SVM** | 0.646889 | 0.826910 | 0.180021 |
| **K-Means** | 0.610105 | 0.788865 | 0.17876 |
| **Decision Tree** | 0.794533 | 0.834467 | 0.039934 |
| **MLP Neural** | 0.499745 | 0.809648 | 0.309903 |
| **Gaussian Process** | 0.570798 | 0.773243 | 0.202445 |
| **Quadratic Discriminant** | 0.774373 | 0.740489 | -0.033884 |

*Figure: Accuracy 10 models, 2 splitting points*

Chart, bar chart

Description automatically generated

Based on these results, further experimentation and analysis focused on the ensemble classifiers – Bagging, AdaBoost, and Random Forest with popular =1 for popularity >= mean + standard deviation.

The Bagging Classifier was able to provide promising results using 5-Fold Validation. It boasted an average accuracy of 88.79% with a standard deviation of 0.00208 of the accuracies, when trained using the entire dataset. Of course, this wasn’t achieved without experimenting with the setting of this classifier. Using the default settings on the classifier, an average accuracy of about 86-87% was achieved. But what proved to be the best method was setting it’s oob\_score to true. Doing this improved the average accuracy to about 88.79%. With such a low deviation, this method so far seemed suitable for predicting the popularity of Spotify songs from these 8 genres. Here are the statistics:

|  |  |
| --- | --- |
| Average Accuracy = | 0.8879 |
| Standard Deviation = | 0.00208 |
| Weighted Average Recall = | 0.89 |
| Weighted Average F1 = | 0.88 |

And here is a classification report of one of the runs:

*Table: Precison, recall, F1 score*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Training** | **Precision** | **Recall** | **F1** | **Total** |
| **0** | 1.0 | 1.0 | 1.0 | 52254 |
| **1** | 1.0 | 0.99 | 0.99 | 11250 |
| **Testing** | **Precision** | **Recall** | **F1** | **Total** |
| **0** | 0.92 | 0.95 | 0.93 | 13103 |
| **1** | 0.72 | 0.60 | 0.65 | 2772 |

As for the AdaBoost Classifier, its accuracy showed good results as well with an average accuracy of 88.3% (using 5-Fold Validation on the full dataset). Its standard deviation of the accuracies was 0.0024. These optimized results were achieved by keeping all of the settings of the classifier as default since any change resulted in degradation of its statistics. While this classifier had reasonable results, both its accuracies and standard deviation were higher (marginally) than those of the Bagging Classifier. Thus, it was clear that of these two, the Bagging Classifier was the more desirable model to use. Here are the statistics:

|  |  |
| --- | --- |
| Average Accuracy = | 0.883 |
| Standard Deviation = | 0.0024 |
| Weighted Average Recall = | 0.88 |
| Weighted Average F1 = | 0.0.88 |

And here is a classification report of one of the runs:

*Table: Precison, recall, F1 score*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Training** | **Precision** | **Recall** | **F1** | **Total** |
| **0** | 0.92 | 0.95 | 0.93 | 52254 |
| **1** | 0.70 | 0.60 | 0.65 | 11250 |
| **Testing** | **Precision** | **Recall** | **F1** | **Total** |
| **0** | 0.92 | 0.94 | 0.93 | 13102 |
| **1** | 0.69 | 0.60 | 0.64 | 2772 |

The Random Forest Classifier consistently provided the highest accuracy, so it was run on the full data set. It yielded a training accuracy of 99.7% and a testing accuracy of 89.9%. It had a standard deviation of its accuracies of 0.00218. These results were achieved by turning on the oob\_score and setting the criterion to use Gini Index. These were the most optimized setting using this classifier. Here are the statistics:

|  |  |
| --- | --- |
| Average Accuracy = | 0.8974 |
| Standard Deviation = | 0.00218 |
| Weighted Average Recall = | 0.90 |
| Weighted Average F1 = | 0.89 |

The tables and chart provide the confusion matrix, precision, recall, and F1 score, and feature importance for the classification.

*Table: Confusion matrix*

|  |  |  |
| --- | --- | --- |
| **Training** | **0** | **1** |
| **0** | 52193 | 43 |
| **1** | 148 | 11119 |
| **Testing** | 0 | 1 |
| **0** | 12563 | 558 |
| **1** | 1053 | 1702 |

*Table: Precison, recall, F1 score*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Training** | **Precision** | **Recall** | **F1** | **Total** |
| **0** | 1.0 | 1.0 | 1.0 | 52236 |
| **1** | 1.0 | 0.99 | 0.99 | 11267 |
| **Testing** | **Precision** | **Recall** | **F1** | **Total** |
| **0** | 0.92 | 0.96 | 0.94 | 13121 |
| **1** | 0.75 | 0.90 | 0.68 | 2755 |

*Figure: Feature Importance*

Chart, bar chart

Description automatically generated

After running these ensemble models using 5-Fold Validation, I wanted to attempt to run them using warm start, where I partitioned the full dataset into 5 partitions, used four of them to train and one of them to test. Due to the singular nature of the AdaBoost Classifier, I could not run warm start on it. So, I only ran warm start on the Bagging and Random Forest Classifiers. Here are the statistics for the Bagging Classifier:

|  |  |
| --- | --- |
| Accuracy = | 0.8944 |
| Weighted Average Recall = | 0.89 |
| F1 = | 0.89 |

*Table: Precison, recall, F1 score*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Training** | **Precision** | **Recall** | **F1** | **Total** |
| **0** | 0.93 | 0.97 | 0.95 | 52234 |
| **1** | 0.85 | 0.67 | 0.75 | 11270 |
| **Testing** | **Precision** | **Recall** | **F1** | **Total** |
| **0** | 0.92 | 0.96 | 0.94 | 13123 |
| **1** | 0.75 | 0.59 | 0.66 | 2752 |

Here are the statistics for the Random Forest Classifier:

|  |  |
| --- | --- |
| Accuracy = | 0.8956 |
| Weighted Average Recall = | 0.90 |
| F1 = | 0.89 |

*Table: Precison, recall, F1 score*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Training** | **Precision** | **Recall** | **F1** | **Total** |
| **0** | 0.93 | 0.98 | 0.95 | 52234 |
| **1** | 0.87 | 0.67 | 0.75 | 11270 |
| **Testing** | **Precision** | **Recall** | **F1** | **Total** |
| **0** | 0.92 | 0.96 | 0.94 | 13123 |
| **1** | 0.76 | 0.58 | 0.66 | 2752 |

As can be seen in the tables above, the Bagging Classifier improves using warm start by about 0.6%, but the Random Forest gets worse using warm start by about 0.2%. Overall, it seems the top performing classification model will always be the Random Forest Classifier using 5-Fold Validation. This model consistently has the highest accuracy (89.74%), the some of the top precision, recall, and f1 scores among all of the classification models used so far. In conclusion, the preferred model to predict the popularity of Spotify songs from the chosen eight genres is the Random Forest Classifier using 5-Fold Validation using the parameters: n\_estimators=100, criterion=’gini’, bootstrap=True, oob\_score=True, warm\_start=False, max\_depth=None, max\_samples=None.