Title: MIR Genre Predictor
Name: Christian Lomboy

I. Introduction

Data science is an extensive multi-disciplinary field - its methods and processes are used to extract insights from all sorts of structured and unstructured data. Today, data is being collected on a huge scale. In every industry, we see applications of data science being used to gather information across all platforms, especially the internet. Any information we come across is highly likely to have been used as a data point. When I started learning more and more about this field, I began to question how I could apply it to fuel my curiosity in music. As someone who takes passion in their Spotify playlists, mediocre guitar playing abilities, and concert attendance history; I felt compelled to work on a project relating to music-related data, so I began to consider the many music services and apps that exist out there on our computers and our phones.

Shazam is an app that recognizes songs through the device's microphone and tells you the name and artist. Pandora is a service that receives an artist, song, composer or genre through user input, then uses its engine to analyze your input to provide songs it "thinks" you might like. Spotify and Apple Music have their own ways of recommending music as well. Each of these services have one thing in common: they all use data science methods and processes. How else would they operate? Every service bases itself on a vast amount of information. Each of them has used this information to create models and algorithms that predict and suggest songs. This science of analyzing and categorizing musical data is called **Music Information retrieval** (**MIR**) - it is the backbone of all of these services. Unlike data science, MIR is interdisciplinary, containing several closely related fields such as psychoacoustics, signal processing, machine learning, and computational intelligence.

This project is inspired by the many MIR applications we see today, and I will use its methods to create two functioning models that intend to predict a song's genre. **The first model will receive the song's lyrics as input, and the second model will receive the song's audio attributes as input.** To gather data and create these models, I will use data science methods that I have learned over the past few months, as well as explore new ways of analyzing and manipulating a type of data that I have not yet been exposed to: songs. I will use the internet as my main resource to find the data I will need to create and train these models. Having only a laptop as my tool, I won't have anywhere near as much data as the colossal music services I mentioned before, but I hope to gain some knowledge on how they operate on a smaller scale.

II. Approach

The first step of every successful data science process is the **retrieval of data**. In this case, I'm going to need a lot of songs. Public datasets are available all over the internet. I will be using one song dataset from Kaggle.com, and another from Ranker.com. Both of these datasets contain song titles, artists, and lyrics. Other attributes exist, but we won't need them for training our models. Combining these two datasets should result in enough data to successfully train the lyrics-based model. For the audio-based model, I will be using functions from a package called Spotipy to extract audio features from the songs in our big dataset. Spotipy allows users to gather song data directly from Spotify. Once this is done, I will have to clean and prepare the data before it is used to train the models. This brings me to step two: **preparation and pre-processing**.

My first model will predict a song's genre based on its lyrics. I plan on using a multinomial naive Bayes classifier on the lyric text, so I need to make sure all of the lyrics are scrubbed clean of anything that would throw off the classifier. This includes converting all text to lowercase, as well as eliminating whitespace, special characters, newline characters, etc. After the dataset's text is cleaned, I will also need to consider all of the genres that will be used as classes. In any event - the higher the number of classes,

the more difficult it is to achieve a high classifier score. In order to achieve a desirable score, I will have to group genres together based on their lyrical similarities. Then, I will need to make sure the classifier will be trained on an even distribution of songs from each genre. In order for a model to make truly accurate predictions, it needs to first be fairly exposed to all possible types of data.

As mentioned before, my second model will predict a song's genre based on how the song sounds. I plan on implementing a decision tree classifier. Since I want my model to make predictions based only on sound, I will need to drop all other non-related attributes, such as song titles, artists, and other non-numerical data. Reasons being, for example, if I were to train the model using data that includes the artist, the decision tree would "cheat" and create a node such as "if artist = Kanye West, then genre = hip-hop". I would also like to normalize the audio feature attributes, allowing us to better understand the differences between each genre. This is more so for EDA purposes rather than training purposes. This concludes the preparation and pre-processing phase. It's time to move onto the **processing, EDA, and model creation** phase (this is where the fun begins).

My first model is based on the naive Bayes classifier. This classifier is popular for its text classification capabilities. It's relatively simple to use and it is known for its accuracy and speed. If you have ever taken a probability and statistics class, you've probably heard of Bayes' theorem:

$$P(A|B) = \frac{P(B|A)\ P(A)}{P(B)}$$

"*the probability that an event A occurs given that another event B has already occurred is equal to the probability that the event B occurs given that A has already occurred multiplied by the probability of occurrence of event A and divided by the probability of occurrence of event B*."

To support an intuitive understanding of how this theorem applies to our model, let's think about the genres and lyrics that are used to train the model. Each song's genre and lyrics can be thought of as "events". The model knows that these events have occurred and will use that knowledge to predict the probability of another "event" occurring when presented with a song's lyrics. If the model could talk during its predictions, it would say something like, "Okay, so I see this song's lyrics contain references to tractors and drinking beer in the sunset. **I remember seeing these words before. I know that they were in many country songs. Therefore, I predict this song to be a country song**."

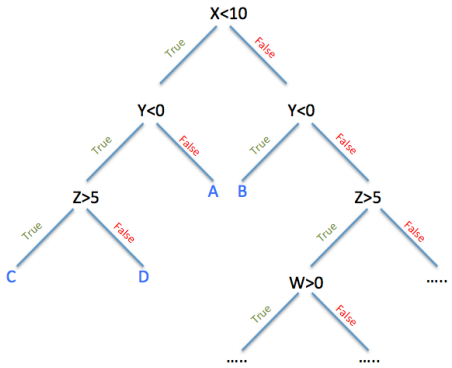
There are many different distributions that are used with naive Bayes. Event frequency is important in this case, so I chose multinomial distribution. I want my model to keep track of how many times it sees the word "beer" in a country song. This will allow my model to consider term frequency when calculating probabilities of a song being a distinct genre.

I will create the model using the multinomial naive Bayes classifier, then I will use a count vectorizer on the training lyrics data. The count vectorizer converts the lyrics into simple word counts. When the model finishes training on the vectorized lyrics, I will calculate its score using the testing lyrics data. If I'm satisfied with my model's score, I'll finish up by testing it out on three songs I know.

Next, I will focus on processing the audio features. Before building the second sound-based model, I'm curious to see the differences between each genre's audio features, so I will perform an **exploratory data analysis** (**EDA**) on the data I gathered using Spotipy. For this EDA, I will group the attributes by genre. Then, I will find the mean of each genre's attribute. Next, I will find the variance between each of the genre's attribute's means, then display them sorted greatest to least. This will allow us to see which attribute differs the most between genres, as well as which attribute is the most similar.

After performing the EDA, I will begin working on my second model. This model will be based on a decision tree classifier. **I chose a decision tree classifier because I believe that each of the song's audio**

features can be considered in a decision tree node. We can assume that a hip-hop song with clear, spoken lyrics will have a high speechiness attribute. For example, let's consider this song in a simple made up decision tree. In the tree, we can imagine one of the nodes holding the condition "speechiness >= 0.85". In this case, the song being considered could have a speechiness of 0.97, and it will hold true when the node's condition is reached. The song will pass through the node with a higher chance of landing on a hip-hop leaf. If, however, a song's speechiness is under 0.85, the probability that it is a hip-hop song will be reduced. This example only takes one attribute into account. The decision tree classifier will consider all attributes.



When creating the model, I plan on tuning the classifier parameters until I reach a desired score. I will try using the entropy information gain function - though I don't expect it to make much of a difference considering its measurement is very similar to the gini function. In the event that the previously mentioned EDA results in some genre's attributes having little variance between each other, I may also reduce the maximum number of features to be considered, as this could reduce unneeded complexity in the tree. I will tweak the rest of the parameters based on trial-and-error. Once the model is complete, I will test the model on the same three songs I tested the first model on. This concludes the approach.

III. Results

I started my project by importing three datasets I found on the internet – two from Ranker.com, and one from Kaggle.com. Ranker's datasets were a bit odd. There were several rows with duplicate song names that contained different parts of the song's lyrics. To fix this, I grouped the songs together by their names then joined the lyrics together. Afterwards, I cleaned up the lyrics, getting rid of whitespace and special characters. Next, I dropped three columns: 'year', 'album', and 'genre'. We won't be needing these columns to train our models. For my classifier, I will instead be using the 'ranker_genre' column. This column's attributes contain more generalized genres that are easy to classify, whereas the 'genre' column contains very specific genres that are sometimes combined with others. For example, one song's genre is labeled as "Soul|R&B|Gospel|Vocal", whereas its ranker genre is labeled simply as "R&B". After dropping the 'genre' column, I renamed the 'ranker_genre' column to 'genre'.

Next, I imported the Kaggle dataset. This dataset is almost three times larger than the two Ranker datasets combined. I cleaned up the lyrics text by removing special characters, whitespace, and newline characters; then converted all the text to lowercase. I noticed that some of the song titles and artists had dash characters in them, so I replaced the dashes with space characters. Next, I dropped the 'year' column, as I won't be needing it to train my models.

Once all of the datasets were imported and cleaned, I merged them together to create one giant song dataset containing almost 500,000 songs - though I doubt this is anywhere close to the amount of data held by popular music services such as Apple Music. In fact, my dataset is small when compared to others. Some datasets can be over a terabyte in size, whereas this one is not even one gigabyte. For my purposes, however, this should be plenty of data to train a couple models. Due to the size of my dataset,

I had to reduce the number of genres in order to train a successful classifier. I found 22 unique genres amongst all of the songs. There were also a few genres that were actually duplicates, but I had forgot to clean the column's attributes before previously merging the datasets. For example, one website's dataset contains the genre 'Hip Hop', while another website has it listed as 'hip hop'. The capitalization causes the attributes to appear as two totally different values. I fixed this issue when grouping the genres.

When grouping different genres together, I used what I knew about each genre to group them together based on their sound and lyrical similarities. For instance, one group I created, 'punk/metal', was formed from four different genres: 'screamo', 'punk rock', 'heavy metal', and 'metal'. All four of these genres have very similar characteristics: emo lyrics, heavy guitars and drums, and high energy. I was able to reduce the number of genres from 22 to 3. Other than grouping similar genres together, I also dropped genres that didn't have much of a presence in the data, or genres that were difficult to classify.

After grouping the genres, I began creating the training and testing subsets. As mentioned earlier, I needed to make sure the model is trained using an even distribution of genres, so I decided to select 50,000 songs from each genre group. I also wanted to make sure my model gets trained on enough information, so I made sure that each song fed to the model contained at least 200 words in its lyrics. I used a for-loop to populate the subsets. Once the subsets were populated, I began creating and testing the first model.

I created the lyrics-based model using the pipeline function, a function that's part of sklearn's library. This function sequentially applies a list of transforms and an estimator. Using the pipeline, I transformed the training data using a count vectorizer, then fitted a multinomial naive Bayes classifier using the transformed data. After fitting, I tested the score of the model using the testing subset. For this model, I was able to achieve a high accuracy score of roughly 83%. To follow up on the model's accuracy, I wanted to test some songs and see if the model could predict their genres. The first song I chose was "Band on the Run" by the great Paul McCartney. I called the model's predict function, fed it the lyrics, and ran the code. Success! My model printed the result, 'country/folk/rock'. Next, I fed the model "Hotline Bling" by Drake. Again, the model was correct, printing 'hip hop'. Lastly, I tested a song by a punk band I used to listen to: "Symmetry" by Title Fight. Correct again: 'punk/metal'.

```
song = pd.Series(["Stuck inside these four walls Sent inside fo
print(model.predict(song))
```

```
['country/folk/rock']
```

```
...
```

```
song = pd.Series(["You used to call me on my cell phone Late ni
print(model.predict(song))
```

```
['hip hop']
```

```
...
```

```
song = pd.Series(["There's symmetry in the way you cut me strai
print(model.predict(song))
```

```
['punk/metal']
```

After creating a successful lyric-based genre estimator, I started focusing on obtaining and analyzing audio attribute data. I chose to use Spotipy to gather audio attributes for each song from our previous dataset. To see how I gathered the audio attributes, I have included a separate notebook, 'MIR_data_collection.ipynb', which showcases how Spotipy works, as well as the code I wrote to extract the data. To get started using Spotipy, first I had to authorize my application with Spotify. In order to do this, I had to log in to Spotify's developer website using my Spotify account, then retrieve a client and secret ID that allows my application to work with Spotify's API. Once authorized, I was able to freely use all of Spotipy's functions in order to retrieve data for the purpose of this project.

To fetch the audio feature data, I first created a new dataframe that was meant to hold the song title, artist, genre, and all of the audio features associated with the song ('danceability', 'energy', 'key', etc.). Then, I shuffled the dataframe we created earlier and iterated through each row in the dataframe, getting each song's title and artist. Next, I used Spotipy's search function to find the song's associated data in Spotify's database. Included in this data is the song's audio features. This is what I'm after.

I parsed the search result to extract each audio feature, then created a new row using these features as attributes. This row is then added to the new audio feature dataframe, along with the song title, artist, and genre. If a song isn't well known, the search function sometimes won't return any results. I used a try-catch block to fix this occurrence, as it would normally end the loop and return an error. This fetching process takes a very long time because each iteration fetches data over the internet from Spotify. Gathering enough features to train a model took days. Each time I worked on this project; I would start the loop in the background in order to hopefully end up with enough data by the time I began creating the second model. I ended up with a dataset containing just under 30,000 songs.

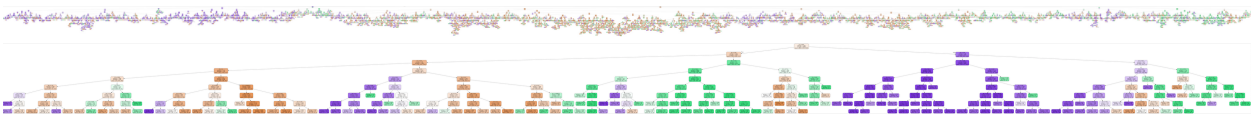
After collecting the data, I began my EDA. I grouped the dataset by genre, then found each attribute's mean. Next, I normalized the dataset's attributes to better understand the differences between each genre's audio features. I used sklearn's variance threshold function to find the variance of each column, then put the variances in a row to be appended in the grouped dataframe. Once added, I sorted the columns in descending order by variance. I ended up with a dataframe that showed me the mean audio feature attribute for each genre, and which features had the most and least similarities between genres.

	instrumentalness	acousticness	speechiness	loudness	danceability	energy	valence	mode	liveness	duration_ms	tempo	key	time_signature
country/folk/rock	0.308914	1.000000	0.283664	1.555187	0.814457	0.615140	0.935537	1.000000	0.862954	0.879286	0.944173	0.956548	0.980070
hip hop	0.150847	0.541163	1.000000	1.204675	1.000000	0.759230	1.000000	0.718757	0.929058	0.951136	0.907905	0.998390	1.000000
punk/metal	1.000000	0.142855	0.523285	1.000000	0.610422	1.000000	0.652038	0.743958	1.000000	1.000000	1.000000	1.000000	0.987927
variance	0.135961	0.122653	0.088646	0.052554	0.025314	0.025205	0.022845	0.016143	0.003132	0.002458	0.001435	0.000405	0.000067

We can see that **'instrumentalness', 'acousticness', and 'speechiness' have the highest variances between each genre, while 'time_signature', 'key', and 'tempo' have the lowest variances**. Using some common knowledge regarding music, we can understand this result by thinking about what makes each genre unique. Punk/metal has very high instrumentalness due to the often presence of screaming guitars, heavy bass, misunderstood lyrics, and thunderous drums; whereas hip hop has very low instrumentalness due to its often usage of electronic beats and synth basses. When we consider the attributes with low variance, it's easy to understand that a song's length often has nothing to do with its genre. I've heard hip hop songs that last over eight minutes, as well as ones that last barely two minutes. This occurs in every genre, which is why we see it reflected in the variances above. There are a number of valid assumptions that could be made from this EDA, but for now I would like to conclude my EDA and begin building the second sound-based model.

Now that I have the audio attribute data, I can begin working on my decision tree classifier model. I created training and testing data using sklearn's simple train-test-split function, using 30% of the data for testing. I created a basic decision tree classifier with no parameters, then fitted it using the training data. The basic classifier achieved an accuracy score of roughly 70%, which actually is not bad at all, given that I haven't provided any parameters yet. To improve this score, I eventually added some parameters to the classifier. I used the entropy function and set the maximum features to 11, maximum leaf nodes to 150, max depth to 7, and the minimum samples split to 100. I refitted the model and ran the score function. 77% accuracy! The parameters I added made a huge difference. When analyzing the tree structure, I noticed that the added parameters significantly reduced the tree's complexity. When I visualized the decision tree with no parameters, it was an overly complex mess containing tons of nodes.

This tree clearly showed signs of overfitting. Making the tree less complex allows the tree to perform better when exposed to data it hasn't seen before.



Tree without parameters (top) versus tree with parameters (bottom)

I initially planned on finishing my model here, but after a spike of curiosity drove me to the internet looking for ways to increase my score, I learned about sklearn's bagging classifier, which is defined as, "*an ensemble meta-estimator that fits base classifiers each on random subsets of the original dataset and then aggregate their individual predictions (either by voting or by averaging) to form a final prediction.*" I created a bagging classifier and used my modified decision tree as its base estimator. I added parameters, setting the number of estimators to 100 and maximum samples to 0.8. I fitted the tree and ran the scoring function against my test data. To my surprise, I was able to further increase my accuracy score to just under 80%. The bagging classifier was able to construct a more accurate version of my decision tree by using samples from my testing data. Although fitting the bagging classifier takes much longer than fitting the decision tree classifier, it is worth the boost in accuracy.

Now that I have achieved a desirable score with my sound-based model, I will test it on the same songs I mentioned earlier when testing the lyrics-based model. I fetched the three songs' attributes using Spotipy, put all the data in a dataframe, then ran the predictor function on it. Each prediction was correct!

```
for i in range(3):
    print(songs[i], "=", bag.predict(testsongsdf)[i], "\n")
```

```
paul mccartney band on the run = country/folk/rock

drake hotline bling = hip hop

title fight symmetry = punk/metal
```

IV. Conclusion

When I first started working on this project, there were times where I thought that I was crazy for trying to come up with not one, but *two* predictor models. When you compare the two, they aren't even very similar. One deals with a single text attribute, while another deals with many numerical attributes. The only similarity is the classes. Throughout this entire process, I've met countless difficulties and technical frustrations. But all of these moments were overcome due to my curiosity stemming from music and data science.

As someone who is driven by passion and curiosity, I was drawn to the challenge. **I knew that this project would be worth the time and effort, just to see and experience how so many music services are able to work their magic**. To me, these two models are achievements. I was able to train a model to classify a song's genre just by looking at its lyrics, and the second model is able to predict a song's genre just by looking at a handful of attributes.

I wouldn't consider these models perfect by any means. After all, I only have access to so much data. And given that all these computations were done on a laptop, I would need much more time and resources if I were to create a "perfect" predictor model. Personally, if I had more time, I would've liked to somehow combine the two models to create a highly accurate predictor model that is trained on both the lyrics and the audio attributes. Nevertheless, the outcome of this project was highly successful, as my models do exactly what I envisioned them doing. I plan on taking this project a step further in the future by possibly creating some sort of web application with user-friendly input options. Just for fun!

V. Acknowledgements

This project could not have been realized without the resources mentioned below:

Websites:

- Medium.com
- Kaggle.com
- Ranker.com

Tools and services:

- Spotify
- Spotipy (https://spotipy.readthedocs.io/en/latest/)
- Scikit-learn (https://scikit-learn.org/)

Articles:

- https://www.wintellect.com/data-science-data-science-process/
- https://towardsdatascience.com/multinomial-naive-bayes-classifier-for-text-analysis-python-8dd6825ece67
- https://tmthyjames.github.io/2018/february/Predicting-Musical-Genres/
- https://www.mathsisfun.com/data/bayes-theorem.html

Reports and presentations:

- https://www3.nd.edu/~rjohns15/cse40647.sp14/www/content/lectures/24%20-%20Decision%20Trees%203.pdf