236502 - Artificial Intelligence Project

Music Genres Classification

Tomer Hass - 315328153

tomer.hass@campus.technion.ac.il

Computer Science Faculty
Technion - Israel Institute of Technology
March 23rd 2023

Contents

1 Introduction

1.1 Music - Definition

Music is defined as a form of art concerned with combining vocal or instrumental sounds for beautry of form or emotional expression, usually according to cultural standards of rhythm, melody, and, in most Western music, harmony. Music most often implies sounds with distinct pitches that are arranged into melodies and organized into patterns of rhythm and metre. The melody will usually be in a certain key or mode, and in Western music it will often suggest harmony that may be made explicit as accompanying chords or counterpoints.

1.2 Music Genre - Definition

A music genre is a category or classification system that groups together musical compositions based on their shared characteristics, such as musical form, style, instrumentation, cultural origin, or historical period. These classifications are often used to help listeners identify and organize different types of music, and to understand the stylistic features and cultural contexts of a particular musical tradition or movement. However, music genres are subjective and may be interpreted differently by different people or cultures.

1.3 Different Types Music Genres

There are over 1300 music genres in the world, and some of them are difficult to identify. Throughout the years, some music genres gained more popularity than others. Genres help us understand where the piece is coming from, its style, the instruments selected, and how they are used.

Blues Music

A genre of African-American music that originated in the southern United States, characterized by its simple, repetitive chord progressions and lyrics about personal struggles and emotions.

Classical Music

A genre of music that originated in Europe in the late 18th century and is characterized by its formal structure, use of orchestral instruments, and adherence to specific musical forms and styles.

Opera

A form of musical theater that features singers performing a dramatic story through a combination of music, lyrics, and acting, often with orchestral accompaniment and elaborate stage sets.

Country Music

A genre of American popular music that originated in the rural southern United States, characterized by its storytelling lyrics, use of acoustic instruments such as guitar and fiddle, and twangy vocal style.

Disco

A genre of dance music that originated in the 1970s and is characterized by its use of a 4/4 beat, funky basslines, and use of orchestral instruments and electronic synthesizers.

Folk Music

A genre of music that is rooted in traditional or rural cultures, characterized by its use of acoustic instruments such as guitar and banjo, storytelling lyrics, and emphasis on community and cultural heritage.

Reggae Music

A genre of music that originated in Jamaica and features a distinctive rhythm known as the ”one drop,” along with prominent basslines, skanking guitar rhythms, and socially conscious lyrics.

Hip-hop Music

A genre of music that originated in African-American and Latinx communities in the United States in the 1970s, characterized by its use of rapping, DJing, and sampling.

Jazz Music

A genre of music that originated in the early 20th century in African-American communities and is characterized by improvisation, swing rhythms, and bluesy melodies.

Rock

A genre of music that originated in the United States in the 1950s and features electric guitars, bass, drums, and often vocals, with a strong emphasis on rhythm, melody, and energy.

Metal

A genre of loud, aggressive rock music characterized by heavy use of distorted electric guitars, fast drumming, and powerful vocals.

Pop

A genre of popular music that emphasizes catchy melodies, accessible lyrics, and a polished production style, typically featuring simple song structures and electronic instrumentation.

Latin Music

A genre that encompasses a variety of styles of music from Latin America, including salsa, tango, bossa nova, and more.

Electronic Music

A broad term that encompasses any music that is primarily created using electronic instruments such as synthesizers, drum machines, and computers.

EDM (Electronic Dance Music)

A genre of electronic music that is primarily produced for nightclubs, raves, and festivals, characterized by its use of computer-generated sounds, synthesizers, and a fast tempo.

Deep-House

A subgenre of electronic dance music that originated in the 1980s, characterized by its use of soulful vocals, 4/4 beat, and incorporation of jazz and funk elements.

Dubstep

A subgenre of electronic dance music that originated in the UK in the early 2000s, characterized by its use of heavy basslines, wobbly synths, and chopped-up vocal samples.

1.4 Project Vision

In this project, I tried to find ways to classify music pieces into their genre using machine learning algorithms. There are many approaches to this classification problem. The most common one uses signal processing algorithms to extract data from the music file (WAV or MP3 files) and extract features to train a learning model from the many there are.

But, to make my project more unique, I was using features from the music streaming service Spotify, which I found very intuitive to understand and attractive to play with to see if there is a connection between some of them to the genre of the music piece.

2 Defining the Solution

Music genres by definition are classifications of music pieces, therefore it is a classic case of classification problem to classify which genre each music piece belongs to.

2.1 System Scheme

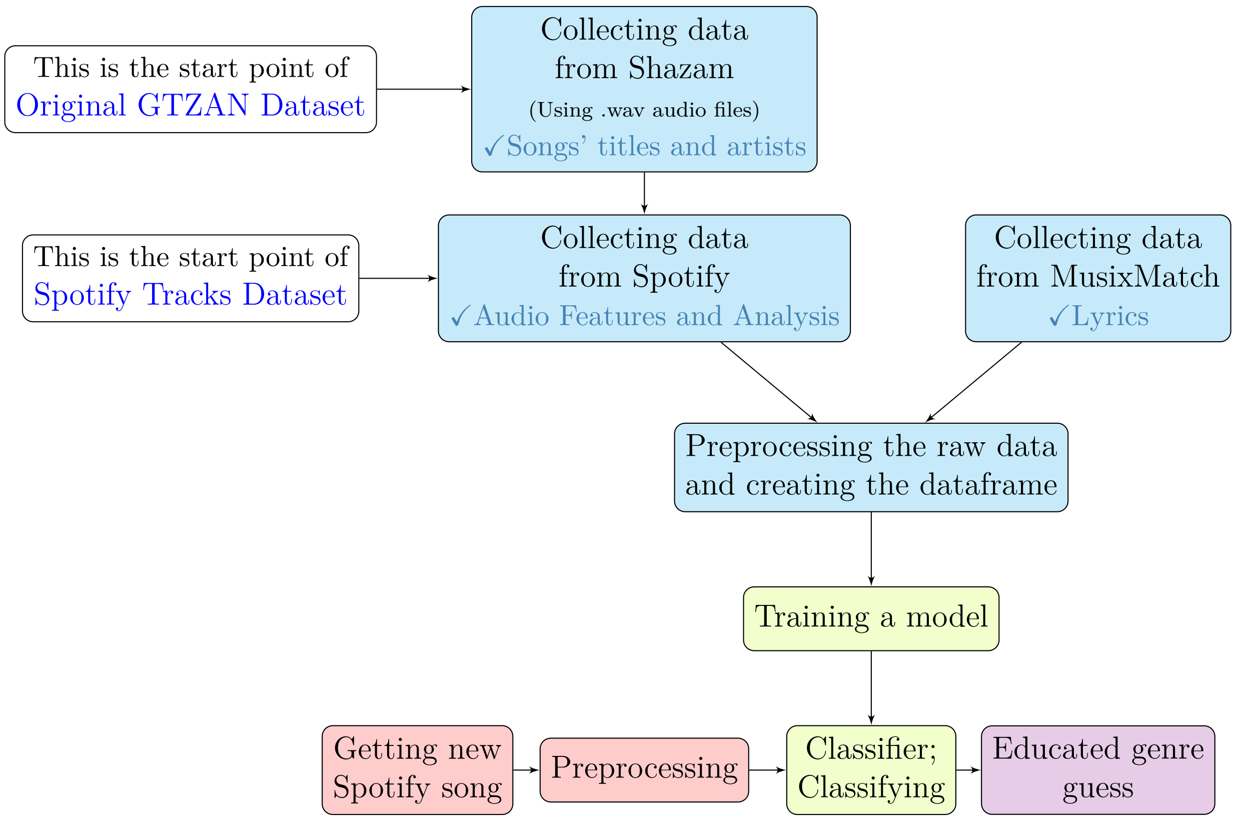

This is the start point of Original GTZAN Dataset
Collecting data from Shazam
(Using .wav audio files)
✓Songs' titles and artists
This is the start point of Spotify Tracks Dataset
Collecting data from Spotify
✓Audio Features and Analysis
Collecting data from MusixMatch
✓Lyrics
Preprocessing the raw data and creating the dataframe
Training a model
Getting new Spotify song
Preprocessing
Classifier; Classifying
Educated genre guess

Figure 1: System Schema

2.2 Datasets

Throughout the project I have used a few datasets to conduct the experiments, all from kaggle.com:

1. Original GTZAN Dataset

   This dataset contains 1000 songs divided into 10 genres: blues, classical, country, disco, hip-hop, jazz, metal, pop, raggae and rock. The dataset contains 1000 audio WAV files each 30 seconds long representing the best part of the song.

2. Spotify Tracks Dataset

   This dataset contains 114 genres, where each genre is represented by 1000 songs (totaling 114000 samples). The dataset is all in a compact CSV file. This dataset was only discovered later after many attempts to improve results occurred in first dataset.

2.3 Features

Features from the Track Audio Feature Spotify API request:

1. `acousticness`: A confidence measure from 0.0 to 1.0 of whether the track is acoustic. 1.0 represents high confidence the track is acoustic.

2. `danceability`: Danceability describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity. A value of 0.0 is least danceable and 1.0 is most danceable.

3. `energy`: Energy is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity. Typically, energetic tracks feel fast, loud, and noisy. For example, death metal has high energy, while a Bach prelude scores low on the scale. Perceptual features contributing to this attribute include dynamic range, perceived loudness, timbre, onset rate, and general entropy.

4. `instrumentalness`: Predicts whether a track contains no vocals. "Ooh" and "aah" sounds are treated as instrumental in this context. Rap or spoken word tracks are clearly "vocal". The closer the instrumentalness value is to 1.0, the greater likelihood the track contains no vocal content. Values above 0.5 are intended to represent instrumental tracks, but confidence is higher as the value approaches 1.0.

5. `key`: The key the track is in. Integers map to pitches using standard Pitch Class notation. E.g. 0 = C, 1 = C♯/D♭, 2 = D, and so on. If no key was detected, the value is -1.

6. `liveness`: Detects the presence of an audience in the recording. Higher liveness values represent an increased probability that the track was performed live. A value above 0.8 provides strong likelihood that the track is live.

7. `loudness`: The overall loudness of a track in decibels (dB). Loudness values are averaged across the entire track and are useful for comparing relative loudness of tracks. Loudness is the quality of a sound that is the primary psychological correlate of physical strength (amplitude). Values typically range between -60 and 0 db.

8. `mode`: Mode indicates the modality (major or minor) of a track, the type of scale from which its melodic content is derived. Major is represented by 1 and minor is 0.

9. `speechiness`: Speechiness detects the presence of spoken words in a track. The more exclusively speech-like the recording (e.g. talk show, audio book, poetry), the closer to 1.0 the attribute value. Values above 0.66 describe tracks that are probably made entirely of spoken words. Values between 0.33 and 0.66 describe tracks that may contain both music and speech, either in sections or layered, including such cases as rap music. Values below 0.33 most likely represent music and other non-speech-like tracks.

10. `tempo`: The overall estimated tempo of a track in beats per minute (BPM). In musical terminology, tempo is the speed or pace of a given piece and derives directly from the average beat duration.

11. `valence`: A measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track. Tracks with high valence sound more positive (e.g. happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g. sad, depressed, angry).

Features from the Basic Track info Spotify API request (`Get Track`):

12. `duration_ms`: The duration of the track in milliseconds.

13. `explicit`: Whether or not the track has explicit lyrics (true = yes it does; false = no it does not OR unknown).

14. `popularity`: The popularity of the album. The value will be between 0 and 100, with 100 being the most popular.

Other features were processed and extracted in several ways via the Audio Analysis API request. Audio analysis structure contains a few parts and the ones that were used are:

15. `sections`: Sections are defined by large variations in rhythm or timbre, e.g. chorus, verse, bridge, guitar solo, etc. Each section contains its own descriptions of tempo, key, mode, time_signature, and loudness. This feature is divided to a few sections where each section has been approximate by Spotify to have the same values of tempo, key, mode etc. Each section has also the duration_ms so we know how long were the values the same, meaning each song (which have different lengths) has its own amount of sections.

16. `segments`: Each segment contains a roughly conisistent sound throughout its duration. Each segment contains its own descriptions of loudness, pitches, and timbre. There are usually a lot more segments than sections, meaning it contains finer details to work with.

17. `bars`: The time intervals of the bars throughout the track. A bar (or measure) is a segment of time defined as a given number of beats.

18. `beats`: The time intervals of beats throughout the track. A beat is the basic time unit of a piece of music; for example, each tick of a metronome. Beats are typically multiples of tatums.

19. `tatums`: A tatum represents the lowest regular pulse train that a listener intuitively infers from the timing of perceived musical events (segments).

These features (15-19) are divided each to a few parts where each part has been approximated by Spotify to have the same measures during its duration. In addition, in features `sections` and `segments`, there is also sub-feature `duration_ms` so we can tell how long were the values the same. One of the main problem with these features is the fact that for each song, there were different amount of parts for each of the features so there was a need to figure out how to even them or bypass this differentiation.

And finally, I have collected lyrics for all the songs that have, except for classical music, assuming it will be 100% instrumental and have no singers. To do so, MusixMatch provided a comfortable free API service that allows you to get 30% of the lyrics.

2.4 Data collection

For the GTZAN Dataset, as it was just the audio files, not even song names and artists, to get the information from Spotify Web API, first, I had to use Shazam API service to get the titles of the songs and who played them, and only then was I able to use Spotify to extract features for the songs.

To make that happen I wrote python scripts to communicate with those services using the packages `shazamio` (for Shazam API) and `requests` (for Spotify API).

In the process of getting the details about the songs two kinds of errors occurred that shrank the dataset:

1. Shazam couldn't recognize the song. Reading the API revealed that Shazam only uses the first few seconds to try to identify it, so playing another part by hand helped a little.
2. Spotify, as vast its data is, didn't acknowledge all the songs in the dataset. It may be due to the rarity of some songs.

Another problem that was discovered using Shazam API is that it identified other songs than it should have (Viewing the resulting dataset revealed some songs' titles were empty or had weird names). Cross-hearing the song Shazam guessed against the original audio file led to the realization that some of the samples contained wrong information about the identification, turning the resulting dataset to less trusted and with only 876 samples out of the initial 1000 begun with.

For the Spotify Dataset, on the other hand, the csv provided `track_ids` that Spotify assigned for each song in the dataset, so, pulling new information from the its API was a lot easier and a lot more reassuring to match the song.

3 Experiments Methodology

The work-flow on this project splits to two major part, each done with a different dataset.

The first part was to test different types of models with a small dataset. The models that were on the check are Decision Trees, Random Forests and Neural Networks. The main conclusion that came from this part is that Random Forest the most ideal learning algorithm to work with in the next part.

The second part, was to test what are the best features to train a Random Forest with that will yield the best accuracy to the given data.

3.1 Official GTZAN Dataset

Throughout this part of the experiment, the features that were used are the Audio Features (Features 1-11) and Basic Track Features (Features 12-14))

3.1.1 Decision Trees

For the first part I was testing a single tree. To make it work a little better I used the feature selection option to make sure only the most relevant features play roles in the model learning. So, using GridSearchCV I trained models where the hyper-parameters are the number of features participating and the decision tree max depth.

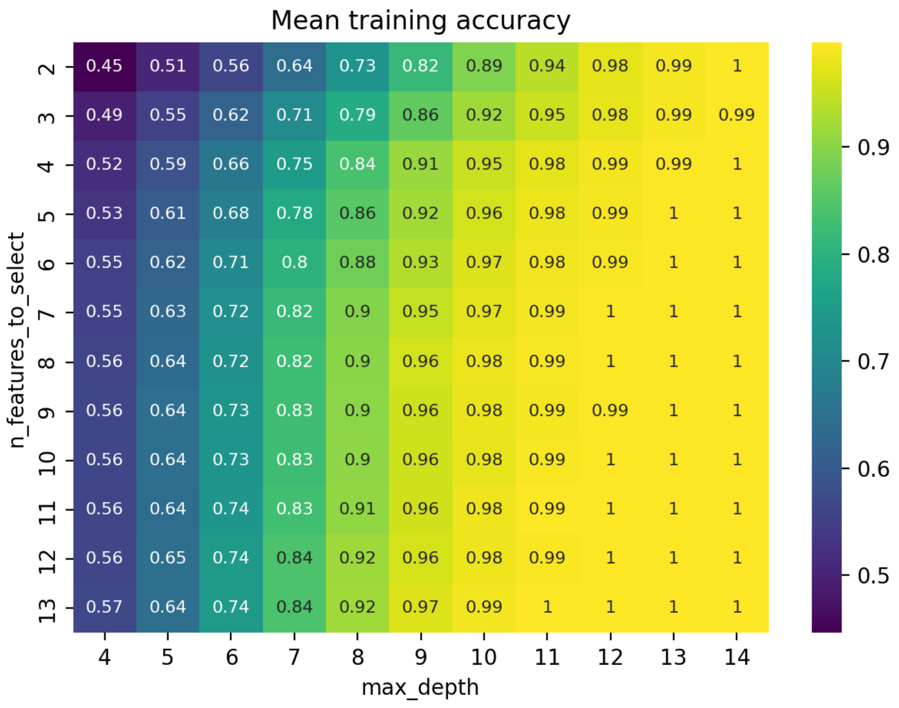


Figure 2: GridSearchCV results for the sequential feature selector's `n_features_to_select` and decision tree's `max_depth` hyperparameters, plotted against train set accuracy

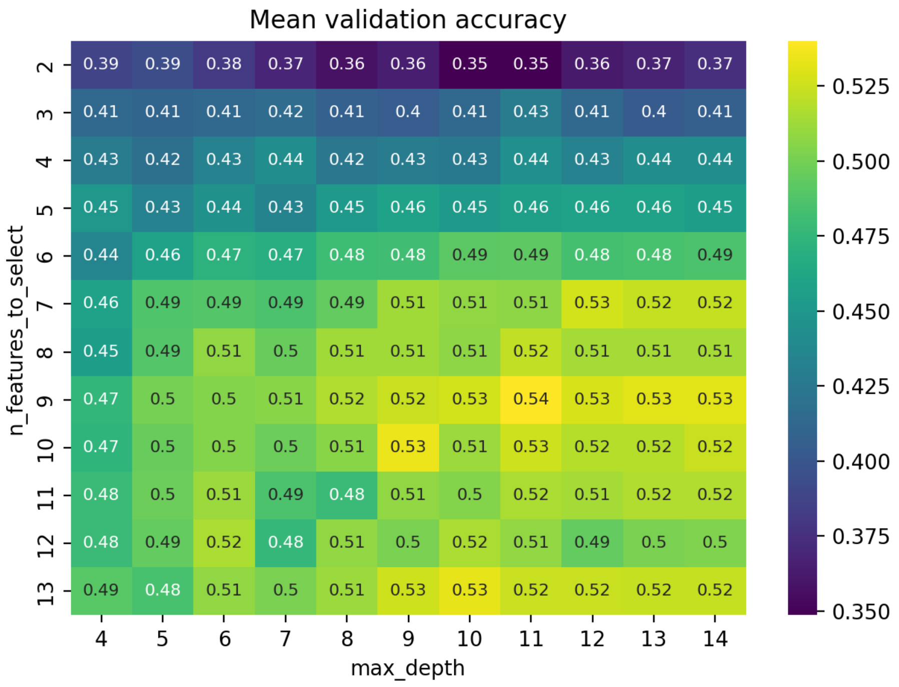


Figure 3: GridSearchCV results for the sequential feature selector's `n_features_to_select` and decision tree's `max_depth` hyperparameters, plotted against test set accuracy

So I decided to train a decision tree that has only 9 features to use, (The sequential feature selector (SFS) concluded that the features should be `duration_ms`, `popularity`, `explicit`, `danceabilitiy`, `loudness`, `speechiness`, `acousticness`, `valence` and `tempo`) and max depth of 11.

The accuracy on the train set was 99.5% and the accuracy on the test set was 48.8%. It seems almost like overfitting, but according to the map, it doesn't look like it.

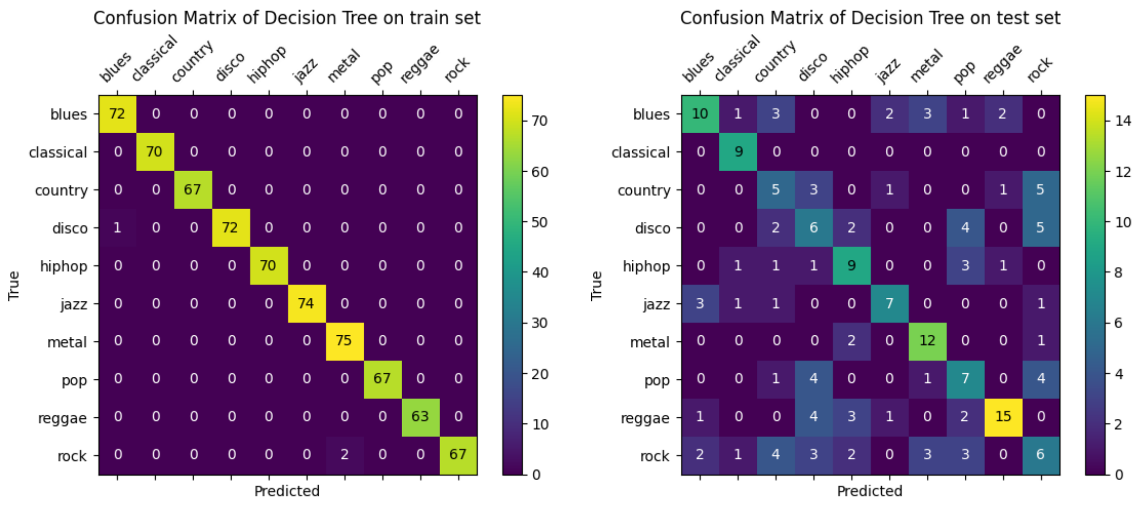


Figure 4: Confusion Matrix of the trained decision tree on the dataset

The confusion matrices in Figure 4 reveal some interesting properties of what the final model is able to do:

1. The model is able to detect very well classical music (it had only a few samples in the test so that is why the blue-ish color, but there are no confusions at all. 100% of the time it was able to detect classical music), metal and reggae.
2. There isn't always a symmetrical relationship between any two genres. It is concluded by the fact that the matrix is not always symmetrical. For example, the country genre is highly misclassified as rock, while rock is almost evenly misclassified as all other genres!

3.1.2 Random Forests

Random forest is a better version of decision trees as it is containing the advantages of ensemble models. As I learned in AdaBoost (in Introduction to Machine Learning - 236756), ensemble and boosting models use weights on the models themselves to get which basic models give better results and this way have more generalized yet accurate models. With this model, we will also use tuning to seek the best hyper-parameters to fit the model with for getting the best results.

Using a script I ran many options for random forests where the hyper-parameters are `n_estimators` and `max_depth`. The search for the best values for these hyper-parameters was done using GridSearchCV with 5-fold cross-validation.

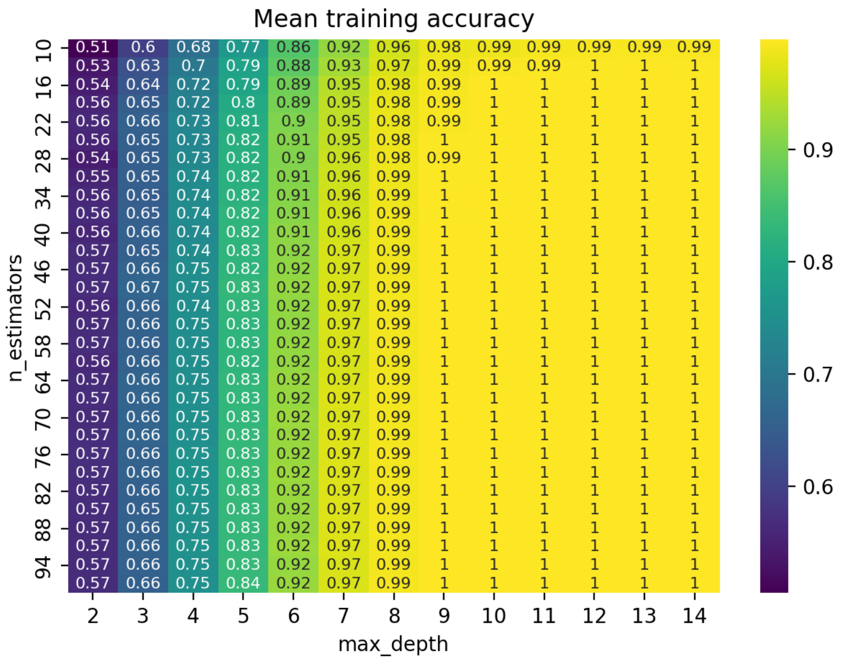


Figure 5: GridSearchCV results for the random forest's `n_estimators` and `max_depth` hyperparameters, plotted against train set accuracy

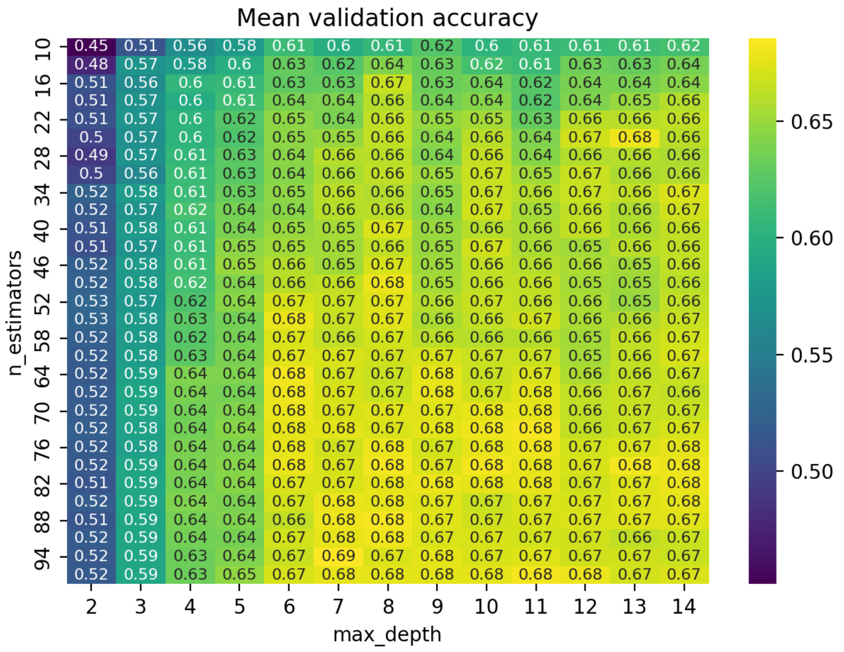


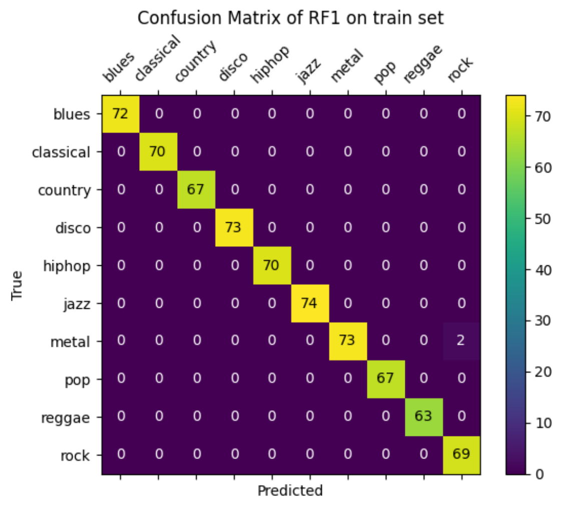
Figure 6: GridSearchCV results for the random forest's `n_estimators` and `max_depth` hyperparameters, plotted against test set accuracy

At first, it seem like Figures 5, 6 have a classic case of overfitting, as we get in many cases 100% accuracy in the training set, but as we see in the validation set, there is a nice stable accuracy of 66%~68%. I think the major factors to the huge boost we get here (about 14% additional accuracy) are:

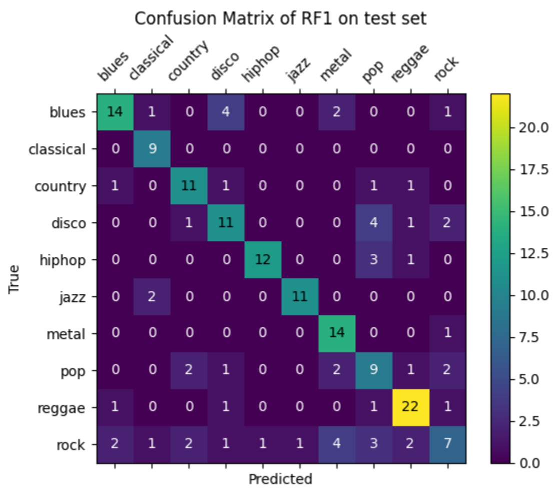
1. The model I used to train the dataset with is an ensemble type of model, which has a high robustness against overfitting, and as we can see in the graphs, there is fairly flat zone in the validation heatmap.
2. The dataset we used contained only 876 samples. That means it is a tiny dataset to work with, and the chance to get all the songs correct and create the illusion of overfitting is relatively easy to get.

Therefore, I chose a specific random forest with "overfitted" parameters to finalize the experiment, which are 73 trees in the forest (`n_estimators = 73`), and each decision tree has max depth of 11 levels.

The accuracy on the train set was 99.7% and the accuracy on the test set was 67.61%.



(a) Accuracy 99.7% on the train set



(b) Accuracy 67.61% on the test set

Figure 7: Confusion Matrix of the trained random tree on the dataset. `n_estimators = 73`, and `max_depth = 11`

It is interesting to see that the random forest is mostly confused on the more 'generic' genres which are pop and rock. By generic I mean their music style is more generic and it is indeed less understandable how to classify them as metal has more strict style than rock, so classifying rock music as metal is something that will never change.

3.1.3 Neural Network

The last model type I wished to check is the neural network. Neural networks are a great way to make a model that can find hidden connections between features and hopefully create a better model. The sad part is that all tries were failures and I ended up choosing random forest to be the 'supreme' learning algorithm to go on with. I know that any function can be created with neural networks but it would take a lot of time to figure out the layer structure to make it better, so I stuck to the Random Forest model because it was a simple yet powerful learning algorithm to use.

The model I chose to work with, after many trials to fit better results is:

```
Model: "model"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
input (InputLayer)         [(None, 30)]              0

dense_1 (Dense)             (None, 128)               3968

dense_2 (Dense)             (None, 64)                8256

dense_3 (Dense)             (None, 32)                2080

dense_4 (Dense)             (None, 10)                330

=================================================================
Total params: 14,634
Trainable params: 14,634
Non-trainable params: 0
```

Other settings were fixed for this model for example, there is an inside split of 20% of the samples for validation, and there was an early stopping condition via the validation, where if the validation loss rose up for 10 epochs then stop and restore best weights. The loss function used is called Sparse Categorical Cross-Entropy Loss.

As seen in the next plot, the best results came on quite early:

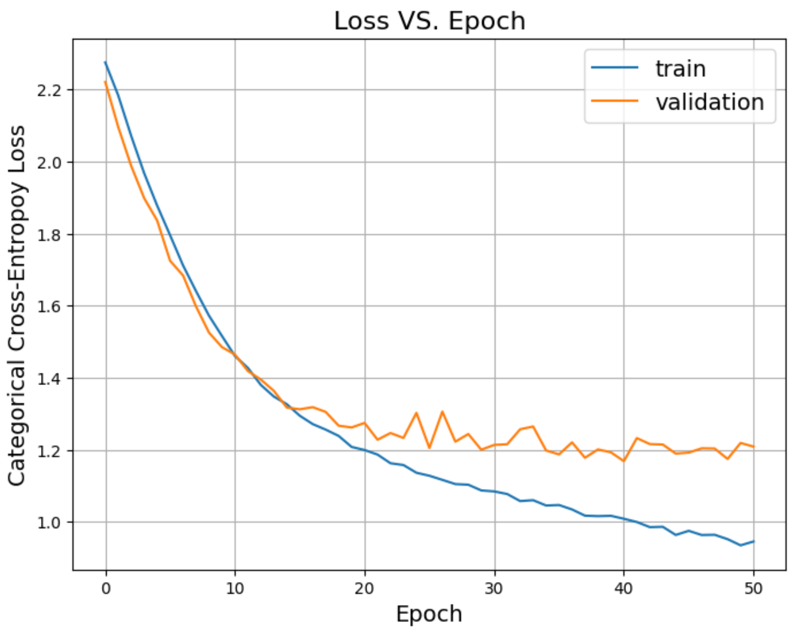
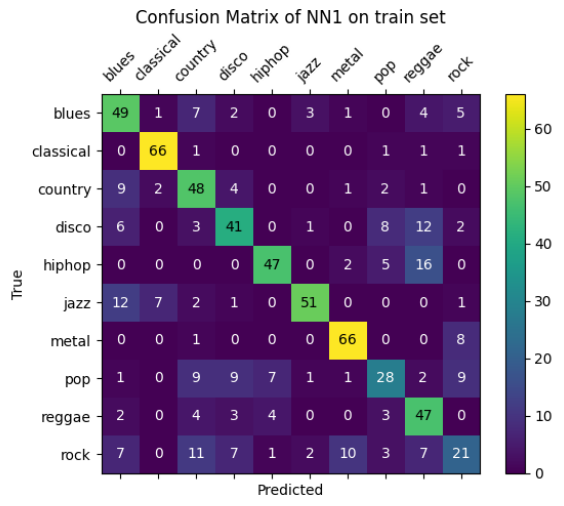
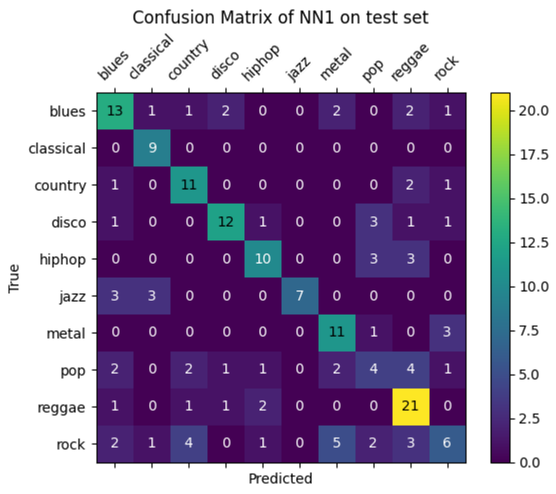


Figure 8: Training process of the neural network



(a) Accuracy 66.3% on the train set



(b) Accuracy 59.1% on the test set

Figure 9: Confusion Matrix of the trained neural network on the dataset

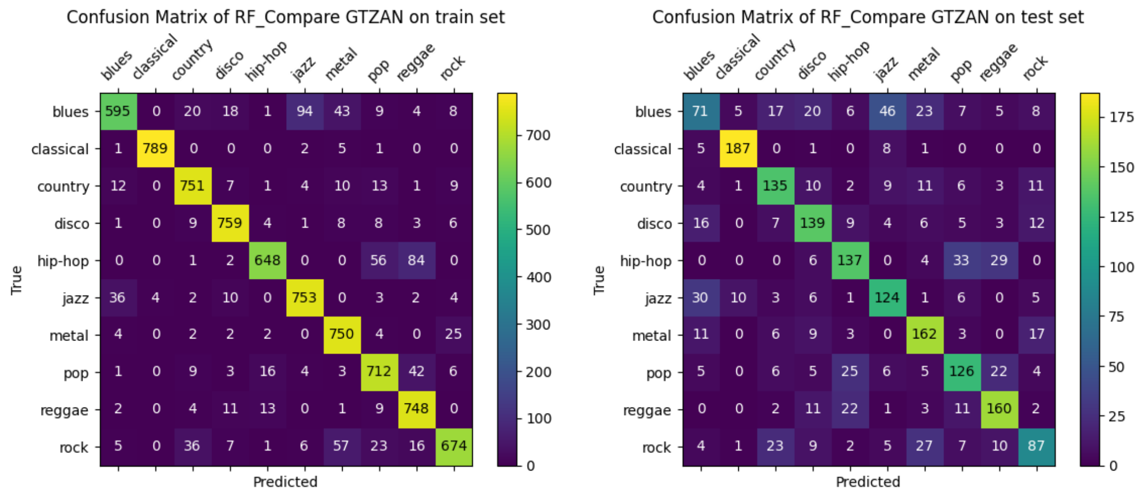
According to the confusion matrices in Figure 9, it seems to have a hard time classifying correctly rock and pop. In differ to the random forest matrix, it seems to misclassify rock to metal, country, and weirdly enough to disco. Actually, it seems that most misclassifications go to disco... ABBA fan maybe?

The same concept seems with blues which the neural network tends to classify either as blues, country, or jazz.

3.2 Spotify Tracks Dataset

First thing I had to do is to filter out the genres I wanted to deal with for the classification. I decided to stick to the 10 GTZAN genres and made a smaller dataset of only the 10 genres mentioned in the GTZAN dataset, but this time instead of initial 100 samples from each genre, I had 1000 sample from each making it a 10000 samples dataset. In addition, I had the right `track_id` for each song, so getting more information from Spotify Web API about each song was easy to get.

First I trained a new random forest on the new dataset, but with the same best hyperparameters (`n_estimators = 73, max_depth = 11`) and start settings (I had both Random Forest to have the same `random_state = 42`)



(a) Accuracy 89.74% on the train set

(b) Accuracy 66.4% on the test set

Figure 10: Confusion Matrix of the trained random forest on the dataset. `n_estimators = 73`, and `max_depth = 11`

The accuracy for the random forest on the train set is 89.74% and the accuracy on the test set is 66.4% which indicates to a case of tiny "underfitting". This makes sense as there is a need to support more samples and get better fitting to a lot more data.

And so, I trained a bunch more random forests to search for the new best hyper-parameters to get the best basic model, before adding more features to the dataset in attempt to improve the statistics:

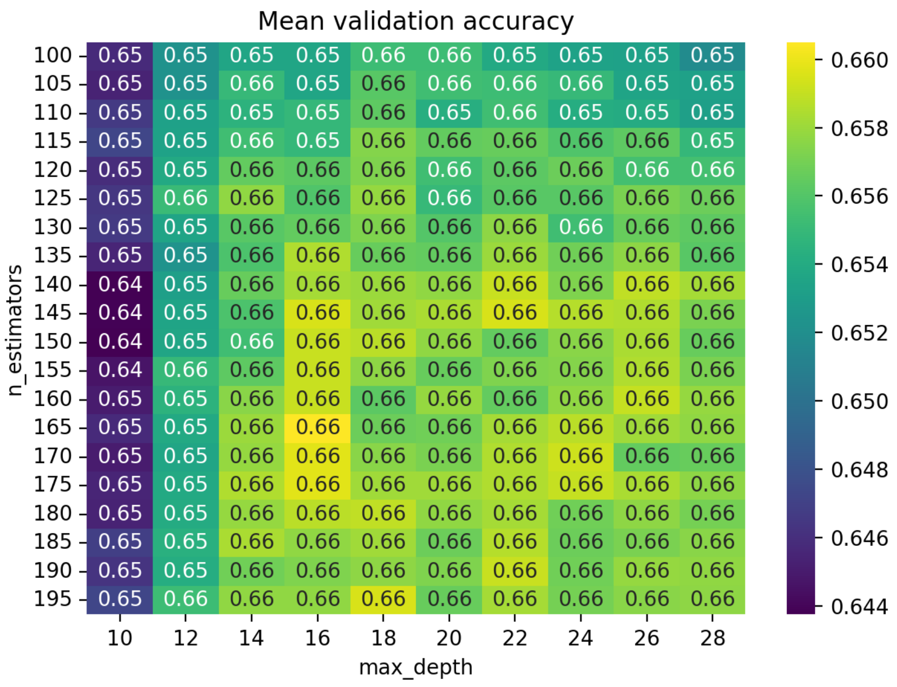
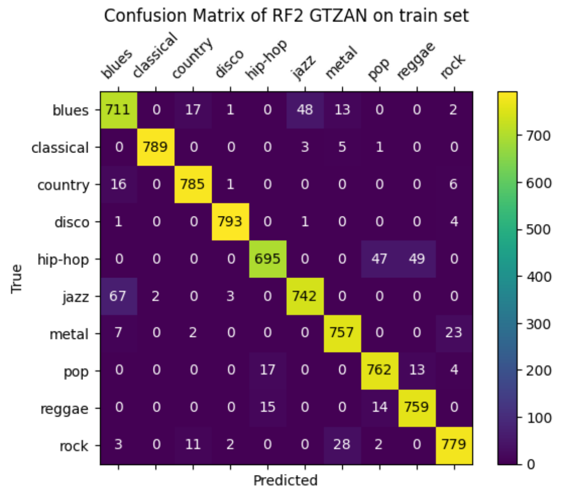
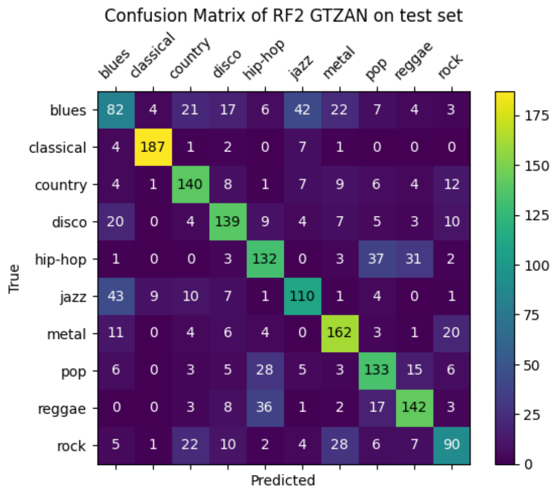


Figure 11: GridSearchCV results for the random forest's `n_estimators` and `max_depth` hyper-parameters, plotted against test set accuracy

I chose to train a Random Forest with specific hyper-parameters. This random forest has 165 trees in the forest and each can go to maximal depth of 16. The accuracy of this model is 94.65% on the training set, and 65.85% on the test set.



(a) Accuracy 94.65% on the train set

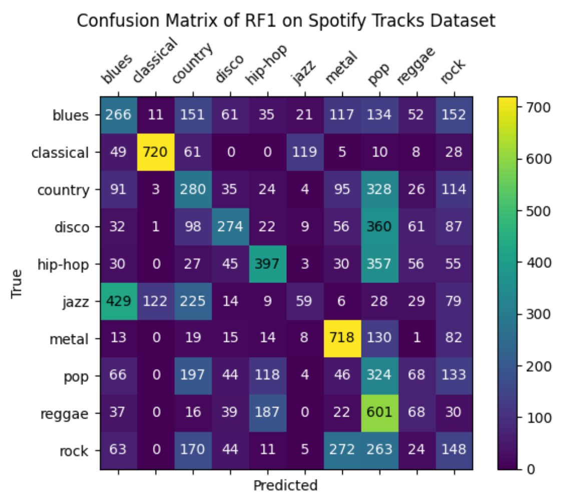


(b) Accuracy 65.85% on the test set

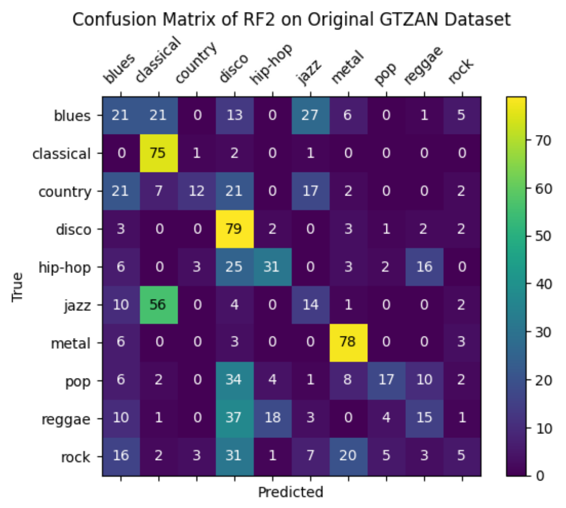
Figure 12: Confusion Matrix of the trained random forest on the dataset. `n_estimators = 165`, and `max_depth = 16`

The question that must be asked is why we got slightly worse accuracy over this dataset when previously we got 3% more accuracy. We can explain it with the reason I mentioned earlier which says that this dataset is a lot bigger than the dataset I played previously, over 10 times bigger (876 samples in the original GTZAN dataset VS. 10000 samples in the Spotify Tracks dataset). Bigger dataset let the learning model to generalize better.

to prove this assumption, I checked the trained random forest RF1 on the Spotify Tracks Dataset, hoping to see really bad results which conclude that the random forest was fitted really well to the small dataset, and on the other hand a good fit (approximately the same ratio) of the original GTZAN dataset on RF2 which is the trained random forest on the big dataset.



(a) Accuracy 32.54% on the Spotify Track Dataset



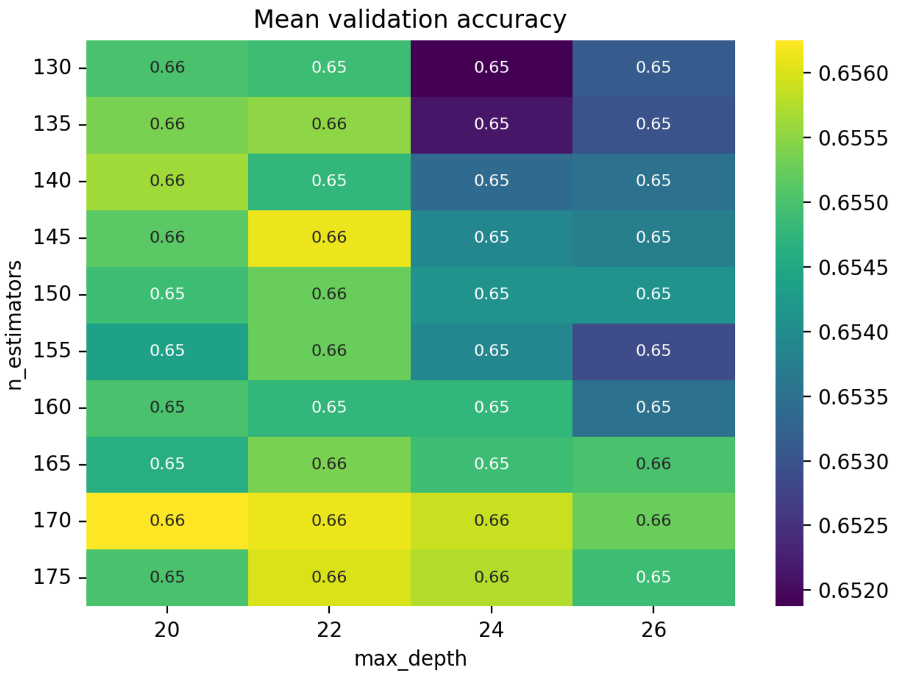
(b) Accuracy 39.61% on the Original GTZAN Dataset

Figure 13: Confusion Matrix of the trained random forest on the different dataset they have been trained on

As we can see in Figure 13, both dataset crossings are very bad, and that means a sample labeled to a specific genre in one dataset, might be labeled differently in the other dataset. Therefore, uniting these datasets will probably cause a noisy and untrue viewing of the real unknown distribution of samples. Nevertheless, it was an interesting experiment to conduct!

3.2.1 Audio Analysis Count features

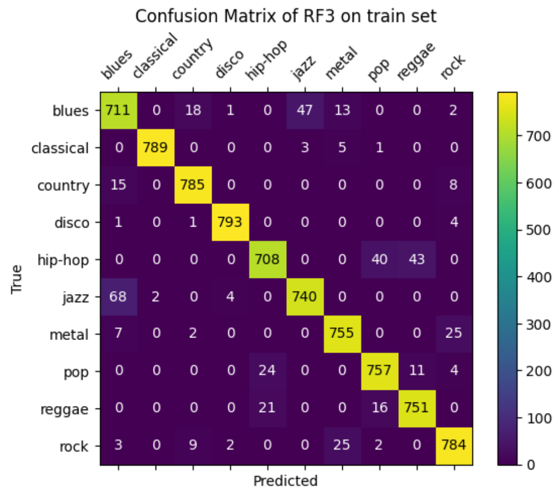
The first try to improve the model, by adding new features to the dataset, is adding the features called bars, beats, tatums, sections and segments. These are just the count of each feature in the audio_analysis.json file. After adding these features to the dataset called `bars`, `beats`, `tatums`, `sections`, `segments` I tested variant combinations of `n_estimators` and `max_depth` values to test which is the best:



n_estimators \ max_depth	20	22	24	26
130	0.66	0.65	0.65	0.65
135	0.66	0.66	0.65	0.65
140	0.66	0.65	0.65	0.65
145	0.66	0.66	0.65	0.65
150	0.65	0.66	0.65	0.65
155	0.65	0.66	0.65	0.65
160	0.65	0.65	0.65	0.65
165	0.65	0.66	0.65	0.66
170	0.66	0.66	0.66	0.66
175	0.65	0.66	0.66	0.65

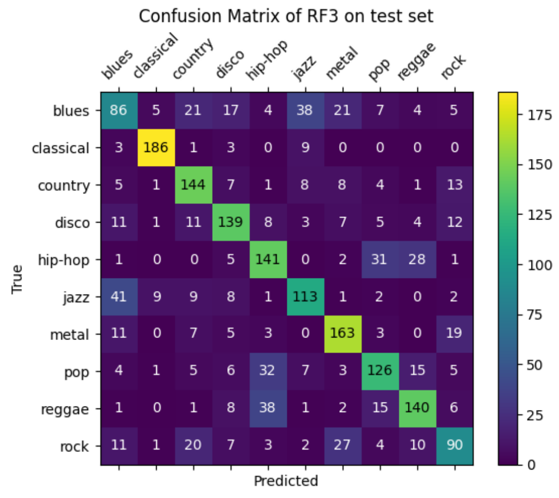
Figure 14: GridSearchCV results for the random forest's `n_estimators` and `max_depth` hyper-parameters, plotted against test set (with counting features) accuracy

Then I decided to train one specific random forest with these features where there are 145 trees in it and each can go to max depth of 22 levels. The train accuracy was 94.7% and the test accuracy was 66.4%.



True \ Predicted	blues	classical	country	disco	hip-hop	jazz	metal	pop	reggae	rock
blues	711	0	18	1	0	47	13	0	0	2
classical	0	789	0	0	0	3	5	1	0	0
country	15	0	785	0	0	0	0	0	0	8
disco	1	0	1	793	0	0	0	0	0	4
hip-hop	0	0	0	0	708	0	0	40	43	0
jazz	68	2	0	4	0	740	0	0	0	0
metal	7	0	2	0	0	0	755	0	0	25
pop	0	0	0	0	24	0	0	757	11	4
reggae	0	0	0	0	21	0	0	16	751	0
rock	3	0	9	2	0	0	25	2	0	784

(a) Accuracy 94.7% on the train set



True \ Predicted	blues	classical	country	disco	hip-hop	jazz	metal	pop	reggae	rock
blues	86	5	21	17	4	38	21	7	4	5
classical	3	186	1	3	0	9	0	0	0	0
country	5	1	144	7	1	8	8	4	1	13
disco	11	1	11	139	8	3	7	5	4	12
hip-hop	1	0	0	5	141	0	2	31	28	1
jazz	41	9	9	8	1	113	1	2	0	2
metal	11	0	7	5	3	0	163	3	0	19
pop	4	1	5	6	32	7	3	126	15	5
reggae	1	0	1	8	38	1	2	15	140	6
rock	11	1	20	7	3	2	27	4	10	90

(b) Accuracy 66.4% on the test set

Figure 15: Confusion Matrix of the trained random forest that supports count features. `n_estimators = 145`, and `max_depth = 22`

This is only slightly better than the RF2 (in Figure 12) without those features. I decided to continue and include them in from this point on and continue improving the model with even ‘more features.

3.2.2 Audio Analysis Sections and Segments features

First Try: Straight Forward

As the first feature seems to improve it by just a bit, I was on to the next feature, which is actually the content of the `sections` and `segments`. The first thing I needed to do is find an algorithm that will make sure that I have the same amount of sections or segments for each song so there won't be NaN values for some samples and make the learning algorithm throw errors. Learning the `sections/segments` I have noticed that each `section/segment` has a sub-feature `duration_ms` where it tells how long the section/segment is going for, and the total is the duration_ms of the entire song.

The pseudo-code (in the appendix) explains the logic of the algorithm I constructed. The input `sections` is a list of `sections/segments` where each `section/segment` is a dictionary of sub-features. `AUDIO_SECTIONS` is a constant that determines to how many parts will the sections split.

The main action of the algorithm is averaging samples that falls ”in-between” real sections. For example, if there are 8 real sections and I want to recreate it to be 100 sections, the algorithm for most of the part just repeat the values of the relevant section at that point of time. if a fake section is falling in the time duration of two or more sections, there was a calculated average taken into consideration that make sense of the values.

Using this algorithm I created for the entire dataset new sections for the audio analysis `sections` and `segments`, this time with an equal amount of sections for each sample.

In the first experiment under this try, I tested the `sections` feature only. The feature count was absurdly big as I made 1000 `sections` (`AUDIO_SECTIONS = 1000`). Moreover, because each `section` has 7 sub-features, I added 7000 brand new features to the dataset, totaling 7020 features to let the random forest train with.

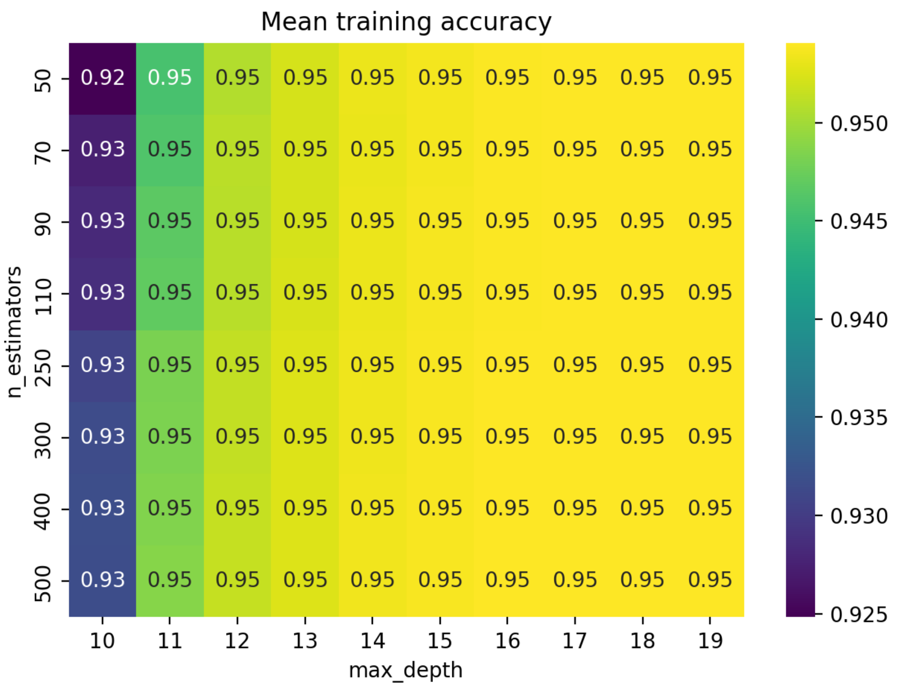


Figure 16: GridSearchCV results for the random forest's `n_estimators` and `max_depth` hyper-parameters, plotted against train set (with 7000 more `sections` features) accuracy

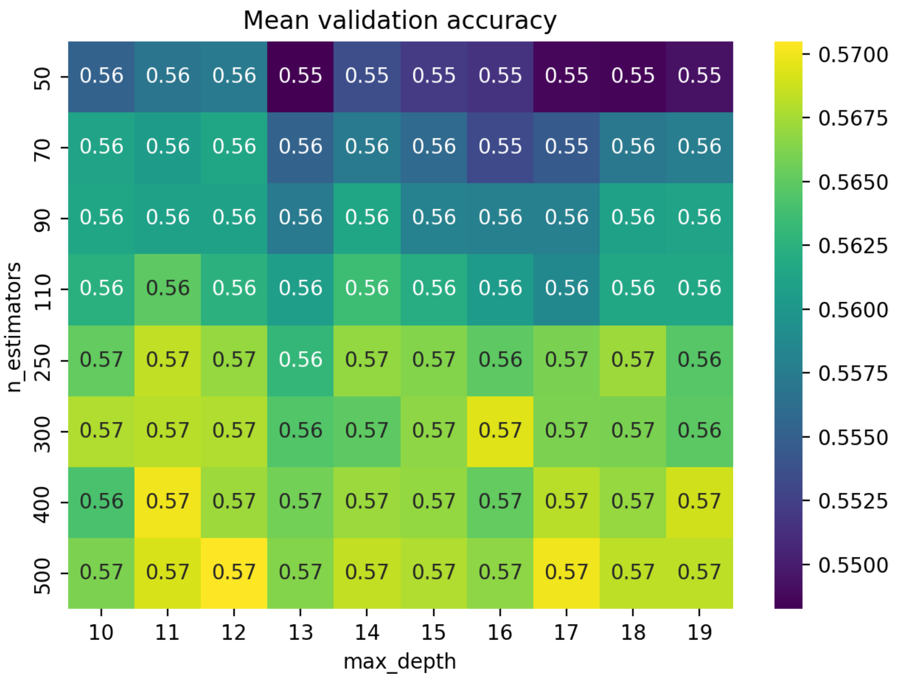


Figure 17: GridSearchCV results for the random forest's `n_estimators` and `max_depth` hyper-parameters, plotted against test set (with 7000 more `sections` features) accuracy

It seems that there was too much noisy features that were added to the dataset that the max accuracy that could be achieved decreased! As it seems, there is 9% less accuracy on the validation set and therefore I didn't bother to make a specific random forest to test a specific case.

Then, in a second experiment under this try, I tested only 10 sections. I ran the algorithm all over again, and now I had 70 added to the 20 I started with (including the genre feature which right before training the model I removed from the dataset) totaling in 90 features (or more like 89 features). The results were a bit better than the previous run:

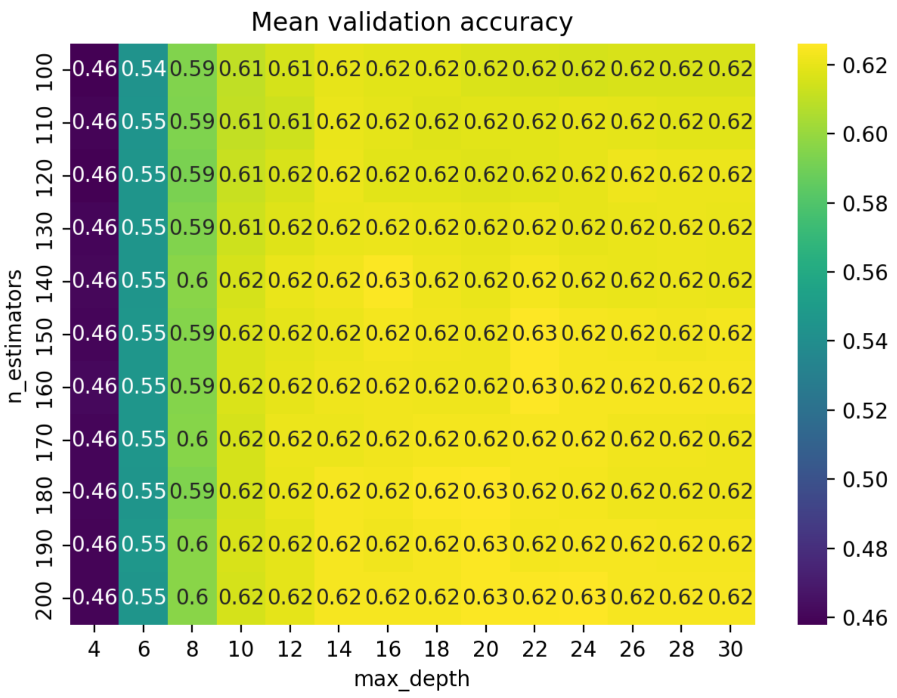


Figure 18: GridSearchCV results for the random forest's `n_estimators` and `max_depth` hyper-parameters, plotted against test set (with 70 more `sections` features) accuracy

Again, it is worse than the start point, which mean the features are too noisy to be informative, so I gave up on going this direction.

The third and last try under this section is using the `segments` feature. Using the same algorithm I made it so every song has the same amount of segments and this time I chose to split/shrink (shrink in case there are more real segments than the amount I required) the real data to 20 segments. Each segment has 24 sub-features yielding total of 480 additional features.

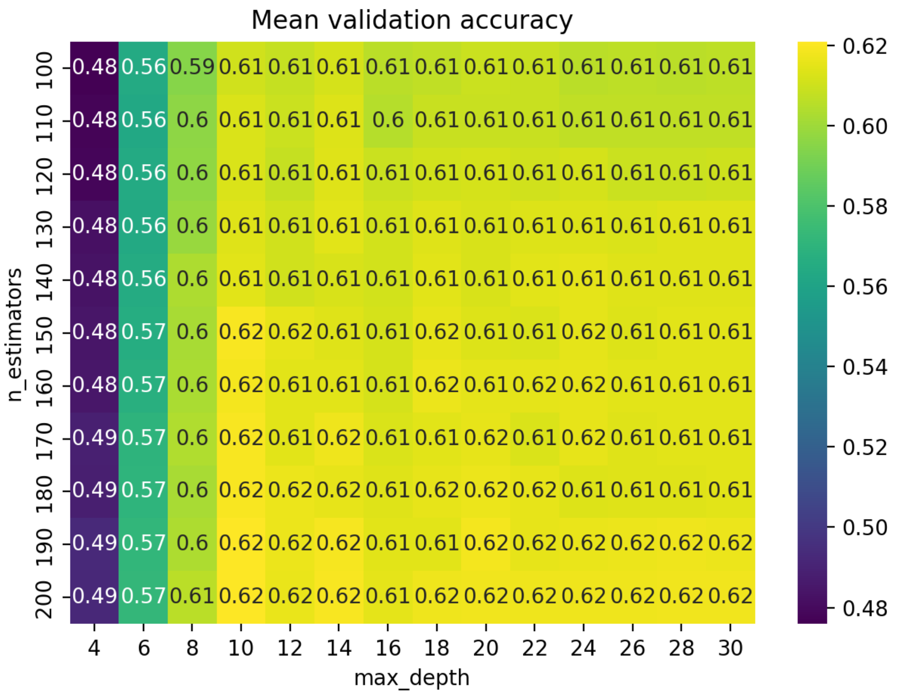


Figure 19: GridSearchCV results for the random forest's `n_estimators` and `max_depth` hyper-parameters, plotted against test set (with 480 more `segments` features) accuracy

Again, it doesn't seem to add more to the accuracy, and therefore I gave up on using these features as they are as additional features.

3.2.3 Audio Analysis Sections and Segments features Second Try: Neural Network

As the first try failed to improve the accuracy, and just made the results worse, I moved to a smarter approach in which I considered the fact that the sections and segments are in a sequence and each section or segment is focusing on a specific portion of the song, and that they are arranged to match the different parts of the song. So it was the perfect opportunity to add it to an RNN model and see if any additional information can be learned this way that a regular model with no memory couldn't figure out.

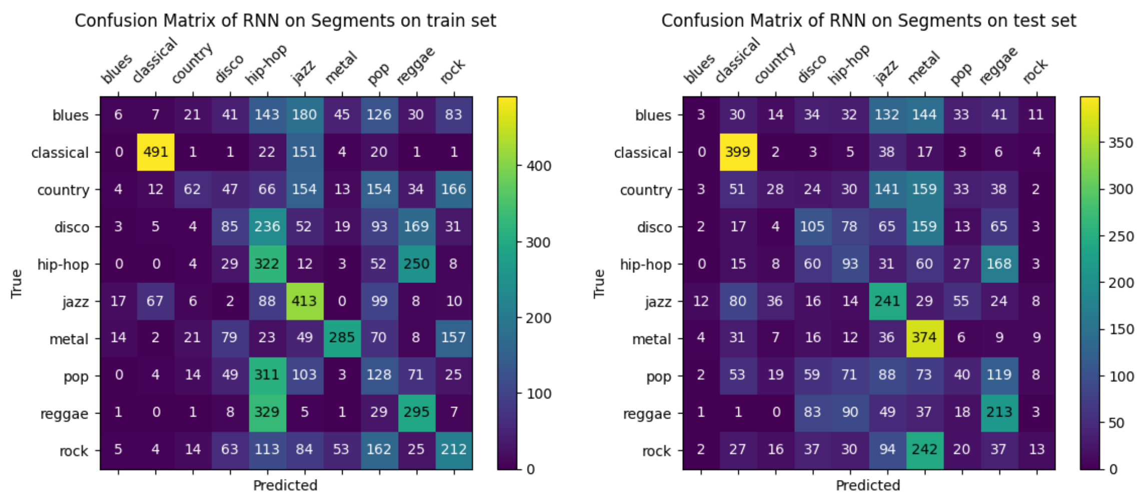
The idea is to train two models: One RNN model on the audio analysis features, then use the predictions of this model as the new features of a random forest! The challenge here this time is to train a 'good enough' RNN model to predict closely enough to the right genre so the random forest will aid these features to some extent.

In the relevant Jupyter Notebook, there are tries to make RNN using only the sections feature once, and only the segments feature once, realizing they won't do any good to the final score, I decided "what the hell", I trained an RNN using both features. The dataset for the RNN is a 3-dimensional matrix in the shape (10000, 200, 31) Where 10000 is the number of samples, 200 is the time-steps available for the RNN to train on, and 31 features total (7 from each section, and 24 from each segment).

After many trials and errors, I got to the following RNN structure:

```
Model: "sequential"

_________________________________________________________________
Layer (type)                    Output Shape              Param #
=================================================================
 bidirectional (Bidirectional)  (None, 200, 64)           16384

 dropout (Dropout)              (None, 200, 64)           0

 bidirectional_1 (Bidirectional) (None, 200, 128)         66048

 dropout_1 (Dropout)            (None, 200, 128)          0

 bidirectional_2 (Bidirectional) (None, 200, 128)         98816

 dropout_2 (Dropout)            (None, 200, 128)          0

 bidirectional_3 (Bidirectional) (None, 64)               41216

 dropout_3 (Dropout)            (None, 64)                0

 dense (Dense)                  (None, 10)                650

=================================================================
Total params: 223,114
Trainable params: 223,114
Non-trainable params: 0s: 0
```

The confusion matrices on the train and test looked interesting enough for me to green light the use of it as the model to predict the whole dataset:



True \ Predicted (train)	blues	classical	country	disco	hip-hop	jazz	metal	pop	reggae	rock
blues	6	7	21	41	143	180	45	126	30	83
classical	0	491	1	1	22	151	4	20	1	1
country	4	12	62	47	66	154	13	154	34	166
disco	3	5	4	85	236	52	19	93	169	31
hip-hop	0	0	4	29	322	12	3	52	250	8
jazz	17	67	6	2	88	413	0	99	8	10
metal	14	2	21	79	23	49	285	70	8	157
pop	0	4	14	49	311	103	3	128	71	25
reggae	1	0	1	8	329	5	1	29	295	7
rock	5	4	14	63	113	84	53	162	25	212

True \ Predicted (test)	blues	classical	country	disco	hip-hop	jazz	metal	pop	reggae	rock
blues	3	30	14	34	32	132	144	33	41	11
classical	0	399	2	3	5	38	17	3	6	4
country	3	51	28	24	30	141	159	33	38	2
disco	2	17	4	105	78	65	159	13	65	3
hip-hop	0	15	8	60	93	31	60	27	168	3
jazz	12	80	36	16	14	241	29	55	24	8
metal	4	31	7	16	12	36	374	6	9	9
pop	2	53	19	59	71	88	73	40	119	8
reggae	1	1	0	83	90	49	37	18	213	3
rock	2	27	16	37	30	94	242	20	37	13

Figure 20: Confusion Matrix of the trained RNN that supports `sections` and `segments` features

It is clear that the RNN is very noisy and I probably could find a better model with more accuracy, but at this point in time, I got tired of trying so I stuck to the best I could train. It is interesting to see that the RNN nailed the classical genre and had the least amount of errors in metal as well. What is funnily annoying to see is that in both train and test sets, the RNN predicts metal for country, disco, rock, and blues, and predicts reggae for hip-hop, and pop.

As described in the scheme of the RNN, the final layer is calculated with softmax activation function to 10 perceptrons which gives the probability of the input to be each of the genres. These 10 perceptron are used as an output to the RNN and as input to the new features of the dataset.

Using GridSearchCV I trained yet another bunch of random forests with different values for `n_estimators` and `max_depth`.

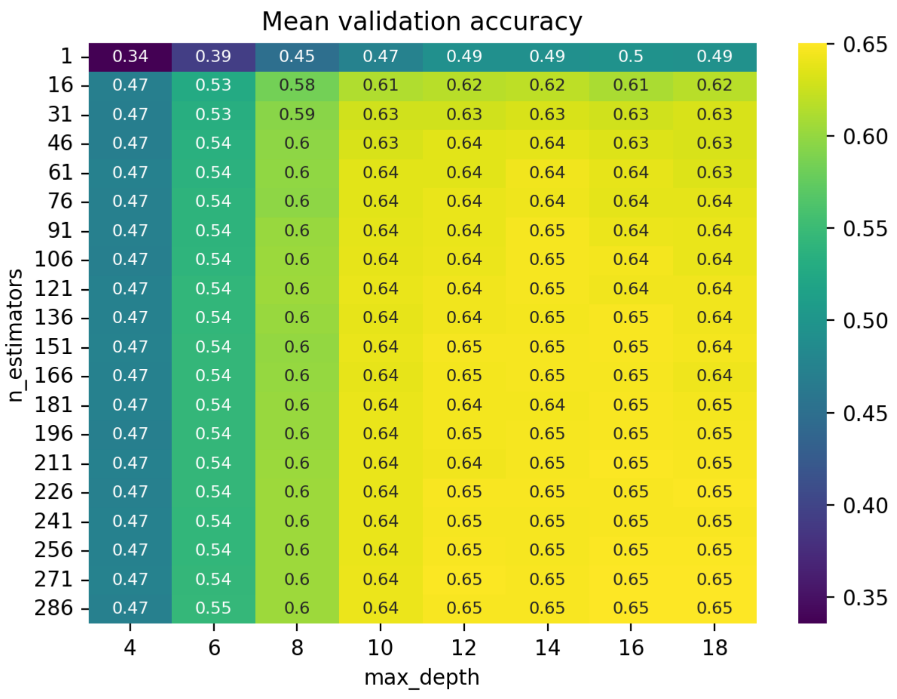
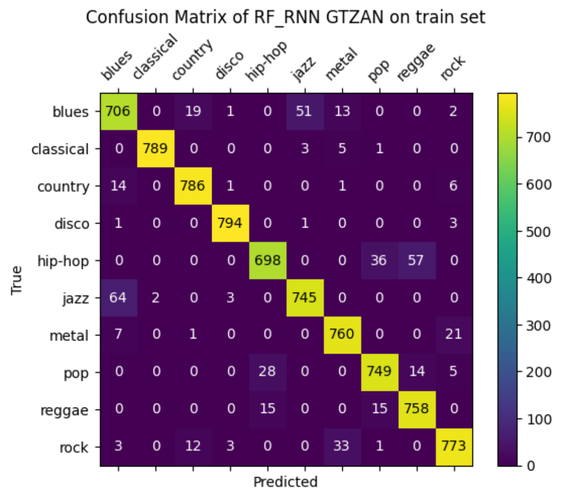
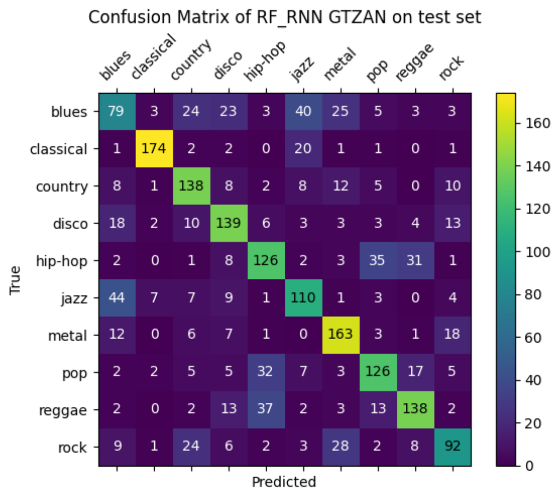


Figure 21: GridSearchCV results for the random forest's `n_estimators` and `max_depth` hyper-parameters, plotted against test set (with RNN features) accuracy

Although it didn't improve the accuracy I was curious to see the influence the new features had on the random forest, so I trained a new random forest with specific hyper-parameters: `n_estimators = 100, max_depth = 14`. The resulting random forest had an accuracy of 94.48% on the train set and 64.25% on the test set, which is not good but not bad either.



(a) Accuracy 94.48% on the train set



(b) Accuracy 64.25% on the test set

Figure 22: Confusion Matrix of the trained RNN that supports `sections` and `segments` features. `n_estimators = 100`, and `max_depth = 14`

The new information just didn't add that new dimension that will significantly improve the accuracy. Interestingly enough it seems the random forest did take a lot of the new features into consideration totaling 30% of the tree branches contained in one way or another the probabilities of these features! It means there was some information hidden in these statistics, but that also tells us that the new data is just linearly correlated to previous features, which kind of makes sense as I suspect that the audio features Spotify evaluates, based on the audio analysis it ran.

```
Variable: acousticness           Importance: 0.09
Variable: danceability           Importance: 0.08
Variable: popularity             Importance: 0.07
Variable: energy                 Importance: 0.06
Variable: metal_probability      Importance: 0.05
Variable: loudness               Importance: 0.05
Variable: speechiness            Importance: 0.05
Variable: segments               Importance: 0.04
Variable: classical_probability  Importance: 0.03
Variable: country_probability    Importance: 0.03
Variable: disco_probability      Importance: 0.03
Variable: hip-hop_probability    Importance: 0.03
Variable: pop_probability        Importance: 0.03
Variable: reggae_probability     Importance: 0.03
Variable: rock_probability       Importance: 0.03
```

```
Variable: duration_ms            Importance: 0.03
Variable: instrumentalness       Importance: 0.03
Variable: valence                Importance: 0.03
Variable: tempo                  Importance: 0.03
Variable: blues_probability      Importance: 0.02
Variable: jazz_probability       Importance: 0.02
Variable: bars                   Importance: 0.02
Variable: beats                  Importance: 0.02
Variable: tatums                 Importance: 0.02
Variable: liveness               Importance: 0.02
Variable: sections               Importance: 0.01
Variable: explicit               Importance: 0.01
Variable: key                    Importance: 0.01
Variable: mode                   Importance: 0.01
```

3.2.4 Lyrics as features

After the previous attempts haven't improved the accuracy of the model, I moved on to see if there is a connection between the lyrics of the song to the genre. In this experiment, I excluded the classical genre as most of the music pieces under this genre are purely instrumental and there was no use to train a model with so little information about lyrics that connected somehow to classical music (I did check and there were 87 out of the 1000 songs that had lyrics. In the rest of the genres there were over 700 songs with lyrics).

The main problem with the lyrics was MusixMatch's free subscription to their API. It only allowed me to ask for only 30% of the lyrics as there is copyright protection on these words. So features like how many times the chorus is repeating, or how many lines were in the full song, were a bit hard to determine.

Nevertheless, I found a way to use the words and to do so I used a python package named `NRCLex`. This package has a simple method that processes text and rates the general emotions expressed in it. It is not something fancy that uses NLP and it is very technical with understandable functionality, but the idea of turning text into an array of emotions with numbers that give the percentage of each emotion in the given text was a great idea.

`NRCLex` rates the text according to 10 emotions: Fear, Anger, Anticipation, Trust, Surprise, Positive, Negative, Sadness, Disgusts, and Joy. So I decided to get these rates as the new features.

As usual I tested the data on a bunch of random forests where I tune the hyper-parameters to give a good mapping on the accuracy of both train and test sets:

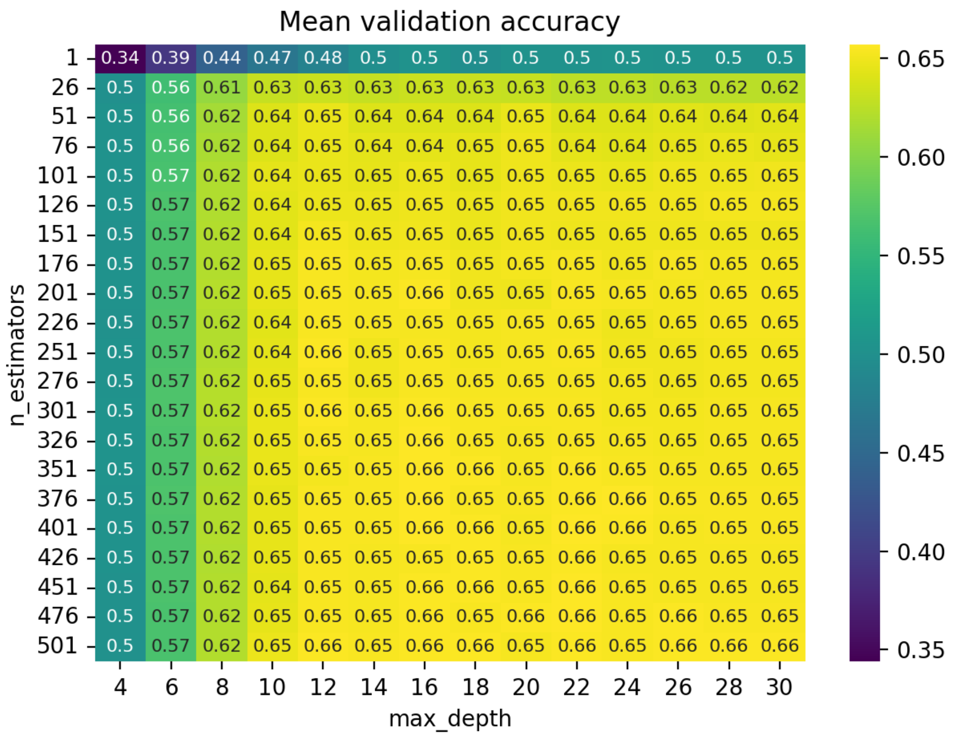
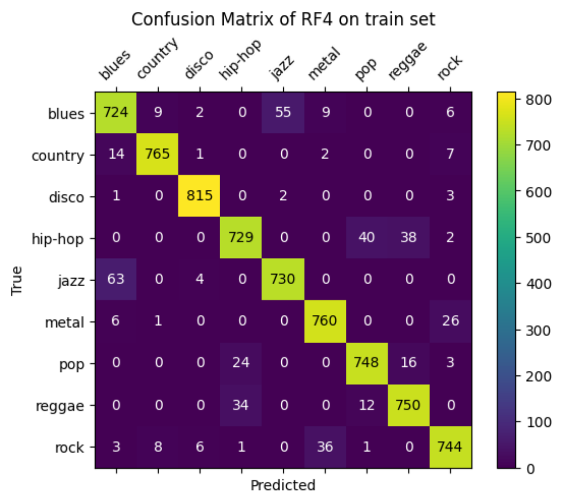
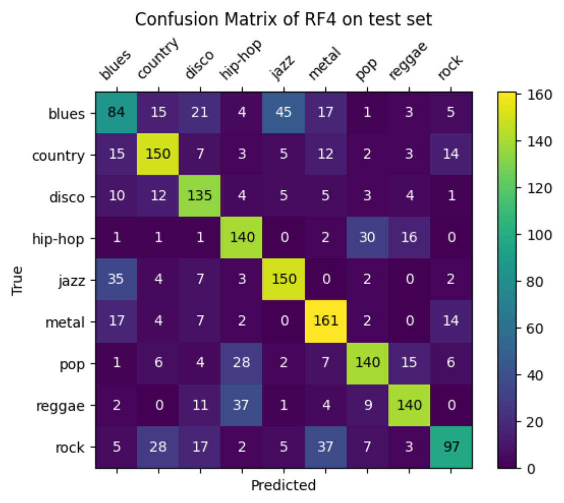


Figure 23: GridSearchCV results for the random forest's `n_estimators` and `max_depth` hyperparameters, plotted against test set (with emotions features) accuracy

There is not any improvement on first glimpse, so I tested a new random forest with specific parameters to see if there is better results on the test set. The accuracy for this random forest, which had `n_estimators = 351, max_depth = 18` was 93.96% on the train set and 66.5% on the test set.



(a) Accuracy 93.96% on the train set



(b) Accuracy 66.5% on the test set

Figure 24: Confusion Matrix of the trained RNN that supports `sections` and `segments` features. `n_estimators = 351`, and `max_depth = 18`

Which is better by 0.1% of the accuracy on the same test set for RF3. Comparing it to the confusion matrix of the random tree it seems the current forest is guessing the same amount of songs for each genre, which is interesting observation, to my opinion.

According to this random forest, the emotions took a total of 19% of the decisions in the forest, which is quite impressive!

```
Variable: popularity          Importance: 0.09        Variable: liveness            Importance: 0.03
Variable: danceability        Importance: 0.09        Variable: fear                Importance: 0.02
Variable: acousticness        Importance: 0.09        Variable: anger               Importance: 0.02
Variable: energy              Importance: 0.07        Variable: trust               Importance: 0.02
Variable: speechiness         Importance: 0.06        Variable: positive            Importance: 0.02
Variable: segments            Importance: 0.05        Variable: negative            Importance: 0.02
Variable: loudness            Importance: 0.05        Variable: sadness             Importance: 0.02
Variable: duration_ms         Importance: 0.04        Variable: joy                 Importance: 0.02
Variable: valence             Importance: 0.04        Variable: sections            Importance: 0.02
Variable: tempo               Importance: 0.04        Variable: key                 Importance: 0.02
Variable: anticipation        Importance: 0.03        Variable: surprise            Importance: 0.01
Variable: bars                Importance: 0.03        Variable: disgust             Importance: 0.01
Variable: beats               Importance: 0.03        Variable: explicit            Importance: 0.01
Variable: tatums              Importance: 0.03        Variable: mode                Importance: 0.01
Variable: instrumentalness    Importance: 0.03
```

4 Summary

4.1 Future research direction

1. Sections and Segments:

   (a) I believe the features of sections and segments contain a lot more information than I tend to extract, and I was using linear algorithms to even the amount of sections and segments for each song. The problem is that the values in those features might not be able to connect linearly, and therefore I might have made noise where I thought I was making sense of the values. So understanding how Spotify extracted those values might help to understand which way is the best to make sense of the values in between which I might have added noise.

   (b) In addition, sections and segments might have certain values that might connect to certain genre (as in `sections[0]` is rock-ish section, `sections[1]` is more jazzy section and so on). It might be smart way to make a better probability using information like so to get better results.

2. Lyrics:

   (a) As I mentioned before, the package I used to process the lyrics is very technical and the algorithm is very native and simple to understand, there was no NLP involved in the process and that might have led to the poor rate of the lyrics. So another aspect to go with it is actually running a smarter model that uses NLP to understand the text and use it to express more accurate emotions that express in the text.

   (b) Not to mention NLP might actually be able to understand lyrics that are not originally written in English, which were quite a lot to my surprise. I recognized in the dataset many Punjabi, Indian, and Chinese texts that lowered the rate of actual useful information that was presented in the dataset.

   (c) Another technical aspect is that the lexicon that was used in the NRCLex package was very limited. Many songs use unique words, slang, and expressions that can be understandable but as an expression and not as separate words. I don't want to give an example here, because the one I think of is quite rude but I will leave a link here for such a word. (It is a short, funny, YouTube reel)

   (d) The lyrical features are very raw, and there might be some other features that can be extracted from the lyrics, such as chorus repetition, and the use of certain words might reveal more information about the genre. For example, many blues songs actually contain the word "blues" in them as part of the lyrics. This detail might be the same for other genres (I can think of the song "we will rock you" by Queen, which is a classic rock song).

3. Another aspect that might be taken into consideration is the fact there might be other sites that have their own rates and their own features for songs. For example while exploring MusixMatch for lyrics and learning its platform I discovered it

has other features within it (for example `explicit`). I wonder what other music stream services (YouTube Music, SoundCloud, etc.) are using to match favorable songs to each user.

4. Finally, there might be other group of genres that will turn out to have better results just as the styles are more unique, giving more variety, more distinguishable values for the features. Just as a try I actually did one try, constructing the basic dataset (Only audio features and basic track features): First, the dataset contains 12,000 samples split equally to 12 different genres: Classical, Deep-House, Dubstep, EDM, Folk, Hip-Hop, Jazz, Metal, Opera, Pop, Sleep, and World-Music. Then, I trained a bunch of random forests to find the best hyper-parameters for the given dataset:

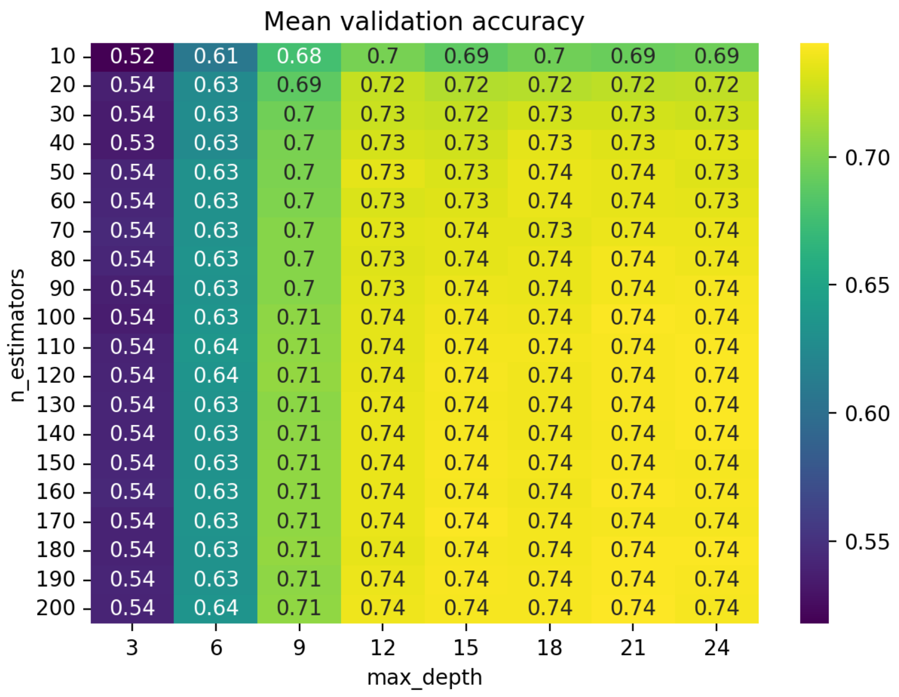
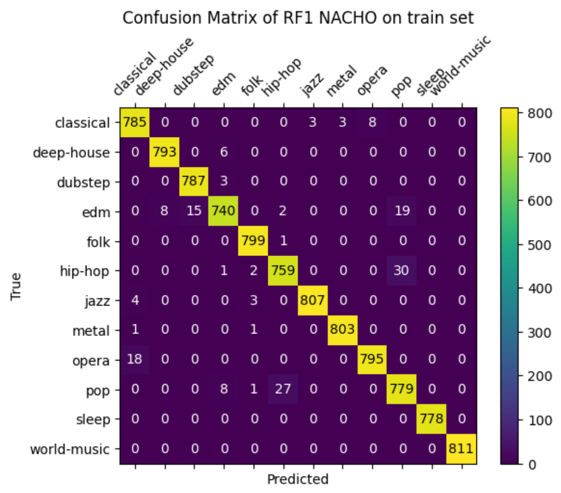


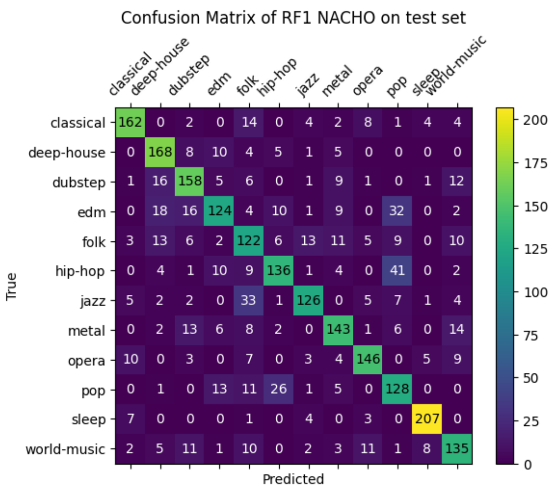
Figure 25: GridSearchCV results for the random forest's `n_estimators` and `max_depth` hyper-parameters, plotted against test set accuracy

As you can see there is a significant improvement of 9%. I then trained a random forest with hyper-parameters set to `n_estimators = 165` and `max_depth = 17`. The accuracy on the train set is 98.3% and the accuracy on the test set is 73.13%



True \ Predicted	classical	deep-house	dubstep	edm	folk	hip-hop	jazz	metal	opera	pop	sleep	world-music
classical	785	0	0	0	0	0	3	3	8	0	0	0
deep-house	0	793	0	6	0	0	0	0	0	0	0	0
dubstep	0	0	787	3	0	0	0	0	0	0	0	0
edm	0	8	15	740	0	2	0	0	0	19	0	0
folk	0	0	0	0	799	1	0	0	0	0	0	0
hip-hop	0	0	0	1	2	759	0	0	0	30	0	0
jazz	4	0	0	0	3	0	807	0	0	0	0	0
metal	1	0	0	0	1	0	0	803	0	0	0	0
opera	18	0	0	0	0	0	0	0	795	0	0	0
pop	0	0	0	8	1	27	0	0	0	779	0	0
sleep	0	0	0	0	0	0	0	0	0	0	778	0
world-music	0	0	0	0	0	0	0	0	0	0	0	811

(a) Accuracy 98.3% on the train set



True \ Predicted	classical	deep-house	dubstep	edm	folk	hip-hop	jazz	metal	opera	pop	sleep	world-music
classical	162	0	2	0	14	0	4	2	8	1	4	4
deep-house	0	168	8	10	4	5	1	5	0	0	0	0
dubstep	1	16	158	5	6	0	1	9	1	0	1	12
edm	0	18	16	124	4	10	1	9	0	32	0	2
folk	3	13	6	2	122	6	13	11	5	9	0	10
hip-hop	0	4	1	10	9	136	1	4	0	41	0	2
jazz	5	2	2	0	33	1	126	0	5	7	1	4
metal	0	2	13	6	8	2	0	143	1	6	0	14
opera	10	0	3	0	7	0	3	4	146	0	5	9
pop	0	1	0	13	11	26	1	5	0	128	0	0
sleep	7	0	0	0	1	0	4	0	3	0	207	0
world-music	2	5	11	1	10	0	2	3	11	1	8	135

(b) Accuracy 73.13% on the test set

Figure 26: Confusion Matrix of the trained RNN that supports `sections` and `segments` features. `n_estimators = 165`, and `max_depth = 17`

The interesting points in these confusion matrix is that there is still (symmetrical) misclassification of pop and hip-hop. Another misclassification to point out that is understandable is of the three genres: Deep-House, Dubstep, and EDM as they all comes from the Electronic genre.

4.2 Final Words

In this project I have tried to find a way to classify music into genres bypassing signal processing the audio file itself. To do so I have collected interesting data from many sources including Spotify and MusixMatch. As I collected information I learned how to connect with different APIs services and how python makes it so much easier to use and implement.

I have learned many different ways to extract information from the raw data, hoping to find interesting features to use for the learning algorithm to train well and get good results. For example the use of RNN for adding the time dimension to data was a great challenge that took days to overcome and understand, use and extract the features from.

Sadly, the project revealed it is not very easy to find features that will drastically improve the accuracy of the model in classifying the genre according to them, if at all! Nevertheless, I had great fun doing so, hearing lots of music, discovering other genres and finding awesome databases to play with!

5 Appendix

5.1 Algorithm Pseudo-code

```python
def analyze_sections(sections):
    total_duration = sections[-1]["start"] + sections[-1]["duration"]
    gain_duration = 0
    audio_sections_count = 0
    intersection_analysis = {}
    sections_analysis = []

    # Analyze Sections
    for section in sections:
      # Divide section into parts and analyze each part
        while gain_duration < section['duration'] / total_duration:
          # If intersection analysis data is available, add it to the current part's analysis data
            if intersection_analysis:
                sections_analysis.append({
                    "sub-feature": gain_duration / (1 / AUDIO_SECTIONS) * section["sub-feature"] + intersection_analysis["sub-feature"],
                    ...
                })
                intersection_analysis = {}
                continue

            # Otherwise, analyze the current part without intersection analysis data
          sections_analysis.append({
                "sub-feature": section["sub-feature"],
                ...
            })
            gain_duration += 1 / AUDIO_SECTIONS

        # If the current part is larger than the audio section size, split the remaining part and save the intersection analysis data
      if gain_duration - (1 / AUDIO_SECTIONS) >= 0:
            left_over_duration = section['duration'] / total_duration - (gain_duration - 1 / AUDIO_SECTIONS)
            gain_duration = 1 / AUDIO_SECTIONS - left_over_duration
        # If the current part is smaller than the audio section size, save the intersection analysis data
      else: # gain_duration - (1 / SECTION_PARTS) < 0:
            left_over_duration = section['duration'] / total_duration
            gain_duration -= left_over_duration

        intersection_analysis = {
            "sub-feature": round(left_over_duration / (1 / AUDIO_SECTIONS) * section["sub-feature"] + (intersection_analysis["sub-feature"] if "sub-feature" in intersection_analysis else 0), 3),
            ...
        }

    # Getting rid or adding missing parts
    while len(sections_analysis) > AUDIO_SECTIONS:
        if len(sections_analysis) % 2 == 0:
            sections_analysis = sections_analysis[:-1]
        else:
            sections_analysis = sections_analysis[1:]
    while len(sections_analysis) < AUDIO_SECTIONS:
        if len(sections_analysis) % 2 == 0:
            sections_analysis.append(sections_analysis[-1])
        else:
            sections_analysis.insert(0, sections_analysis[0])

    return sections_analysis
```