Data Mining (MD)

Bachelor Degree on Informatics Engineering

Final report

Music genre classifier

Felipe Castro
Théo Fuhrmann
Xavier Gordillo
Javier Rivera
Armando Rodríguez
Hasnain Shafqat

29/12/2021

Q1-Autumn-2021/2022

UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

1. Introduction

1.1. Data selection

For this project we decided to use a music dataset containing plenty of features that were obtained with the Spotify API. We found the dataset on Kaggle, the exact source can be found here: https://www.kaggle.com/vicsuperman/prediction-of-music-genre. We decided to give this dataset a try because the music realm is very interesting to study with machine learning, plus having all the interesting features included in the dataset.

1.2. Dataset description

This dataset contains 50k samples and 18 columns, every sample contains all sorts of information about a track, these are the track features included in the dataset:

- **duration_ms**: duration of the track in milliseconds.
- **key** :estimated overall key of the track. Integers map to pitches using standard Pitch Class notation . E.g. 0 = C, 1 = C♯/D ♭ , 2 = D, and so on.
- **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.
- **acousticness**: confidence measure from 0.0 to 1.0 of whether the track is acoustic. 1.0 represents high confidence the track is acoustic.
- **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.
- **energy**: 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.
- **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.
- **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.
- **loudness**: 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 typical range between -60 and 0 db.
- **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.
- **valence**: 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).
- **tempo**: 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.
- **popularity**: popularity of the track. The value will be between 0 and 100, with 100 being the most popular. The popularity is calculated by algorithm and is based, in the most part, on the total number of plays the track has had and how recent those plays are. Generally speaking, songs that are being played a lot now will have a higher popularity than songs that were played a lot in the past. Artist and album popularity is derived mathematically from track popularity. Note that the popularity value may lag actual popularity by a few days: the value is not updated in real time.

The dataset also includes columns for the Spotify's track ID (instance_id), the date when the track was obtained from the API (obtained_date), the track name (track_name), the track's artist name (artist_name) and the genre of the track (music_genre).

1.3. Goal

The end goal of our project will be to find the best performing model among the ones taught in class that can classify a song by its genre within the following ones: classical, electronic, hip-hop and rock. So we will be facing a classification problem.

2. Preprocessing

Before implementing the models seen in class, we need to have a usable dataset, in order to do so we had to go through several preprocessing phases:

2.1. Dataset simplification (label drop)

In order to have a smaller dataset so that we can work with our models without having astronomically high running times, we decided to keep the 4 genres mentioned earlier. This will also allow more interpretable visualisations when it comes to understanding what every model is doing.

Dropping the rest of labels from the dataset decreased our dataset size to 20k samples, having exactly 5k per genre.

2.2. Treatment of missings

According to the Spotify API, some features might have encoded missings, therefore we first checked if there were any NaNs in the dataset, but none were found. So we proceeded to identify the possible encoded missings that the Spotify API had documented. We found two features with encoded missings:

Feature	Encoded missing	# of missings
duration_ms	-1	1991
tempo	'?'	1953

The missings in both features represent about 10% of the dataset each. To impute the missings we decided to compute the mean of each feature per genre and impute each missing's feature with the mean according to its genre, i.e. if there was a sample labeled as "rock" and had its tempo missing, we would impute its tempo with the mean of the "rock" tempo in the dataset.

Thankfully the dataset didn't have a large amount of missings overall, letting us work with more robust material.

2.3. Feature selection

After analyzing every feature in our dataset, we removed some irrelevant columns:

- **instance_id:** this column provides a unique ID per song., therefore it has no correlation with the genre.
- **obtained_date:** the date the song was obtained from the API, therefore it has no correlation with the genre.
- **artist_name:** the song's artist name. We decided to remove this column because spotify links every artist to a specific genre, thus, if an artist has plenty of songs in the dataset (which happens), a huge overfitting can happen.
- **track_name:** the song's name. We decided to drop this column because there isn't much correlation with the genre, plus the amount of unique values would generate an extremely large amount of columns with the one hot encoding technique.

2.4. Model preparation

Every model has its own dataset tweakings in order to obtain the best results, therefore we decided to do the last preprocessing step on each separate model notebook.

We have performed a standardisation on the data using the standard scaler in the following models:

- KNN: the model works with a distance based algorithm, therefore standardising the data is important to avoid misclassifications due to scale.
- Naive bayes: since we're performing a gaussian naive bayes, we need the dataset with a gaussian distribution.
- SVM: we used distance based kernels (linear and RBF), so standardisation is needed.

We also applied a one hot encoding to our categorical variables *key* and *mode*, this will allow us to use the categorical variables in our models, since most of them can only work with numerical data. It's worth mentioning that we will also use the raw categorical variables in the naive bayes model.

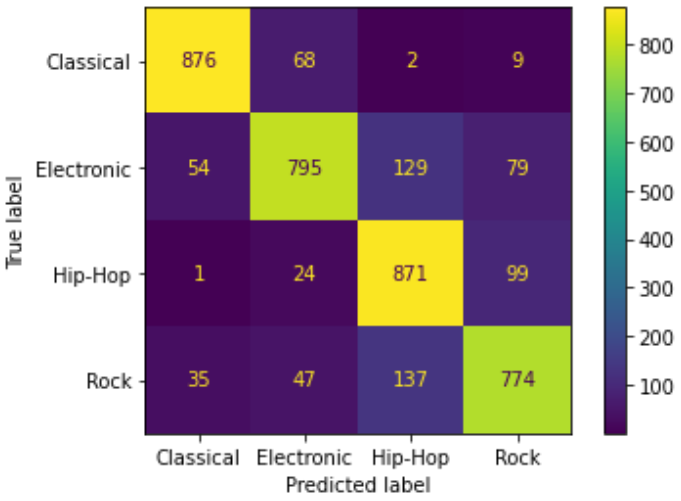
3. Naive bayes

Our data is mixed, it has numerical and categorical attributes, so in order to apply Naive Bayes we will try 2 approaches.

The first one will be to split the attributes between categorical and numerical data and apply a Categorical Naive Bayes to the categorical data and a Gaussian Naive Bayes to the standardized numerical data. Once we've fitted each model with the training data, we will try to predict the test set, obtaining the probabilities of classifying a sample as each of the classes for each model. Later, we will multiply the probabilities obtained by each model, obtaining the probability of classifying a sample by the two models. Selecting the biggest resulting probability, we will have the class that both models combined selected.

On the other hand, we will apply One Hot Encoding on our categorical data. It's the best approach to transform our categorical data to numerical data because the categorical attributes (key and mode) don't have any particular order between their possible values, they are nominal variables. We will also have standardized the numerical data. After doing this, we will fit a Gausian Naive Bayes over all the training set, and we will validate our results predicting the test data. For the first approach we obtain a test validation accuracy of 0.829 and the next classification report and confusion matrix:

	precision	recall	f1-score	support
Classical	0.91	0.92	0.91	955
Electronic	0.85	0.75	0.80	1057
Hip-Hop	0.76	0.88	0.82	995
Rock	0.81	0.78	0.79	993
accuracy			0.83	4000
macro avg	0.84	0.83	0.83	4000
weighted avg	0.84	0.83	0.83	4000

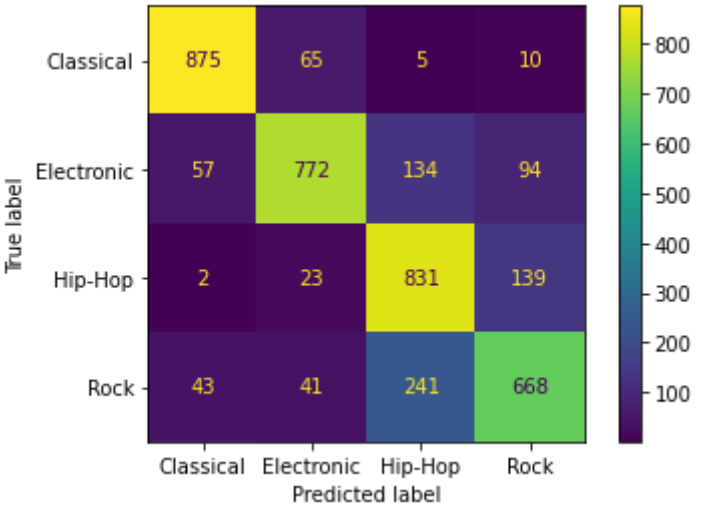


As our test data is balanced enough, accuracy is a good indicator of how good our classifier is, although we can see that the model has good results in recall and precision for all the classes (and therefore a good f1-score), with an average f1-score of 0.83.

The class that is classified significantly better than the others is Classical music, with a recall and a precision of 0.91 and 0.92 respectively.

For the second approach we obtain a test validation accuracy of 0.7865 and the next classification report:

	precision	recall	f1-score	support
Classical	0.90	0.92	0.91	955
Electronic	0.86	0.73	0.79	1057
Hip-Hop	0.69	0.84	0.75	995
Rock	0.73	0.67	0.70	993
accuracy			0.79	4000
macro avg	0.79	0.79	0.79	4000
weighted avg	0.79	0.79	0.79	4000



We can observe that we obtain slightly worse results than with the other approach. But it's interesting to see that Classical music it's also the class that is classified better than the other ones and with a recall and precision similar to the other approach although with all the other classes the values obtained are worse.

Naive Bayes works pretty well despite not all the variables are completely independent of each other. For example, variables such as energy and loudness or danceability and energy are related. That could happen due to the fact that we have a number of examples big enough to get a reliable propabilities that allows Naive Bayes to work pretty well.

4. KNN

In this section we will explain the results and the procedure made with the k-nearest neighbors.

What we did to prepare the data was to read the general clean preprocessed dataset. Then we obtained the one hot encoding of the variables “mode” and “key” achieving a total of 20k rows and 26 features which are maybe too much for KNN, so later we will try to apply PCA to reduce this number and achieve a better result. Then we decided to split the dataset having 20% for the test and 80% for training. We also used the stratify option to obtain balanced datasets and the random_state = 33 to obtain the same splits as the ones obtained with the other models.

Once this was done we normalized the data with the sklearn Normalizer to make the values be rescaled into ones in the range [0,1]. Then, to obtain a baseline, we run the *KNeighborsClassifier* function with the default values: 5 neighbors and the distance ponderated uniformly. We obtained, with a 10-fold cross validation, a validation accuracy of 65,306% with a small deviation among folds of only 0,009.

Checking the test partition we obtained an accuracy of **66,225%** which is even higher than the one obtained with the cross validation. This means that this classifier is not suffering from over or underfitting which is good news. This is the confusion matrix obtained

True\Predicted	Classical	Electronic	Hip-Hop	Rock
Classical	858	116	9	17
Electronic	65	775	100	60
Hip-Hop	7	98	526	369
Rock	14	95	401	490

where we can see some mis-classifications specially in the bottom right part of the matrix. With those results, which are not bad but not the best ones, we wanted to go a bit further and inspect some of the hyperparameters with a grid search with different numbers of neighbors and distances like euclidean or manhattan among others.

Specifically, we test 4 distances: euclidean, manhattan, chebysheva and minkowski (which is the generalization of the euclidean distance) with p being 3. The way of pondering the neighbors was also taken into account using both, the *uniform* and *distance* options, in the weight parameter. Talking about the number of neighbors we test from 5 to 100. This grid search done with 10 folds gives us that the best number of neighbors is 20 using the manhattan metric and the neighbors weighted by the distance.

This combination gives us, using 10 folds, a validation accuracy of 65,86% with a small standard deviation among folds of 0.014. If we take a look at the test partition we obtained an accuracy of **71,23%** which is much better than the obtained with the default classifier. This is the resulting confusion matrix obtained with the best combination explained:

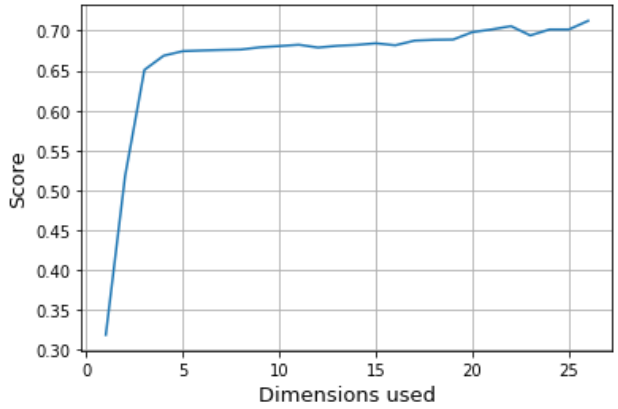
True\Predicted	Classical	Electronic	Hip-Hop	Rock
Classical	1283	164	22	31
Electronic	65	1146	141	148
Hip-Hop	3	90	910	497
Rock	5	80	480	935

We can see, again, good results but not in the bottom right part of the matrix where we can see the greatest part with misclassifications. Nevertheless, let's take a look to the numbers using the classification report provided by the sklearn:

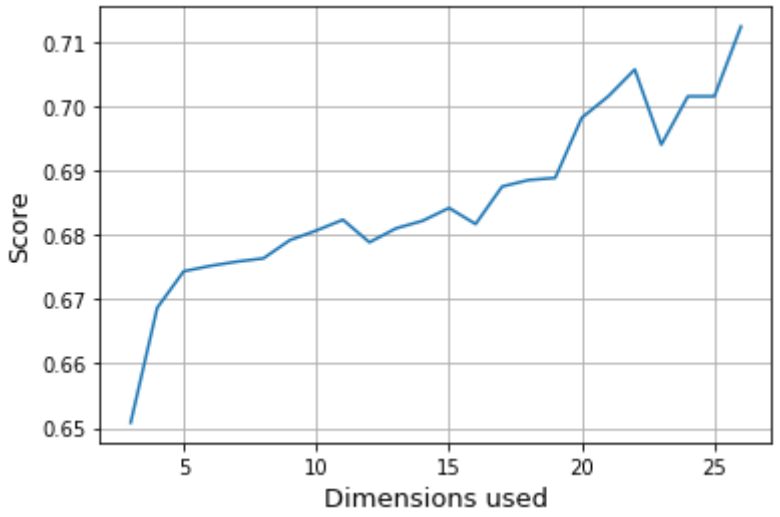
	precision	recall	f1-score	support
Classical	0.95	0.86	0.90	1500
Electronic	0.77	0.76	0.77	1500
Hip-Hop	0.59	0.61	0.60	1500
Rock	0.58	0.62	0.60	1500
accuracy			0.71	6000
macro avg	0.72	0.71	0.72	6000
weighted avg	0.72	0.71	0.72	6000

Here we can see that, as seen before, the classical songs are classified really good. Hip-hop and rock have some issues and just arrive at a 60% of f1-score. However we can see that we have good results with this simple model.

Now let's try to apply PCA to reduce the dimensionality of the problem and provide the KNN only with important dimensions. We did that with different numbers of dimensions to see the results obtained. In the next plot we can see the score obtained searching the best results for every PCA transformed datasets.



However, as the usage of the low dimensions (as 1 and 2) obtain really low results, we plot only from 3 to 26 dimensions. This is the plot:



What we can see over here is that the reductions in the dimensionality didn't give us any improvement as long as we obtain the higher results using all the dimensions. This means that all the variables are important or, at least, that PCA didn't help us make the feature selection in the right way.

5. Decision trees

To apply decision trees to our data, no normalization or standardization is needed, so we will apply the model directly to our data after applying one hot encoding to the categorical variables.

Decision trees are a model that can represent any complex function or data, but tends to overfit the training set and to not generalize very well. That could be seen when we apply decision trees to our data without any consideration in selecting the best hyperparameters.

With the default hyperparameters, we obtain a 0.9986 accuracy with the training data, whereas a 0.83875 with our validation test set. That is a clear indication that our model is overfitting and therefore losing generalizability.

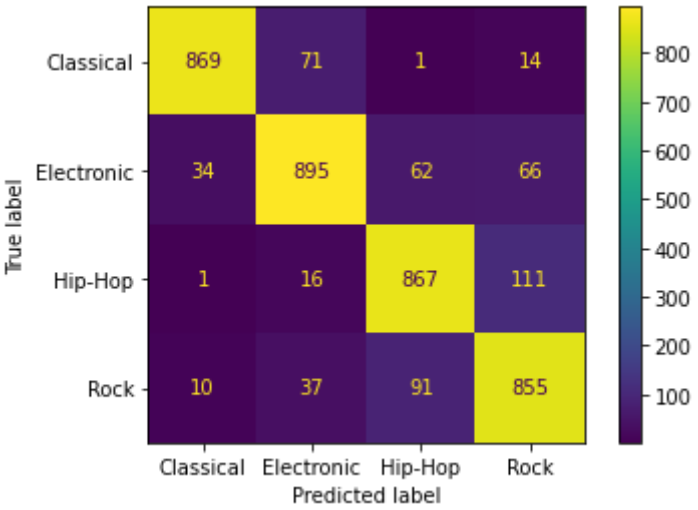
In order to avoid this, we will adjust the hyperparameters that allow us to prune the tree such as max depth, minimum samples to be a leaf and minimum samples to split a node.

We will use GridSearchCV to obtain the best configuration of these hyperparameters. Also, we will explore which of the two criterions for deciding the quality of a split is better, gini or entropy. We get that the best parameters are: entropy criterion, a max depth of 8, a minimum number of samples to be leaf of 6 and a minimum number of samples to split of 15.

We obtained a worse training result than before, but we obtained a better test accuracy of 0.87. That means that we have achieved our goal of increasing the generalizability of the model avoiding overfit.

We obtained the following confusion matrix and classification report:

	precision	**recall**	**f1-score**	**support**
Classical	0.95	0.91	0.93	955
Electronic	0.88	0.85	0.86	1057
Hip-Hop	0.85	0.87	0.86	995
Rock	0.82	0.86	0.84	993
accuracy			0.87	4000
macro avg	0.87	0.87	0.87	4000
weighted avg	0.87	0.87	0.87	4000



As our test data is balanced enough, accuracy is a good indicator of how good our classifier is, although we can see that the model has good results in recall and precision for all the classes (and therefore a good f1-score), with an average f1-score of 0.88.

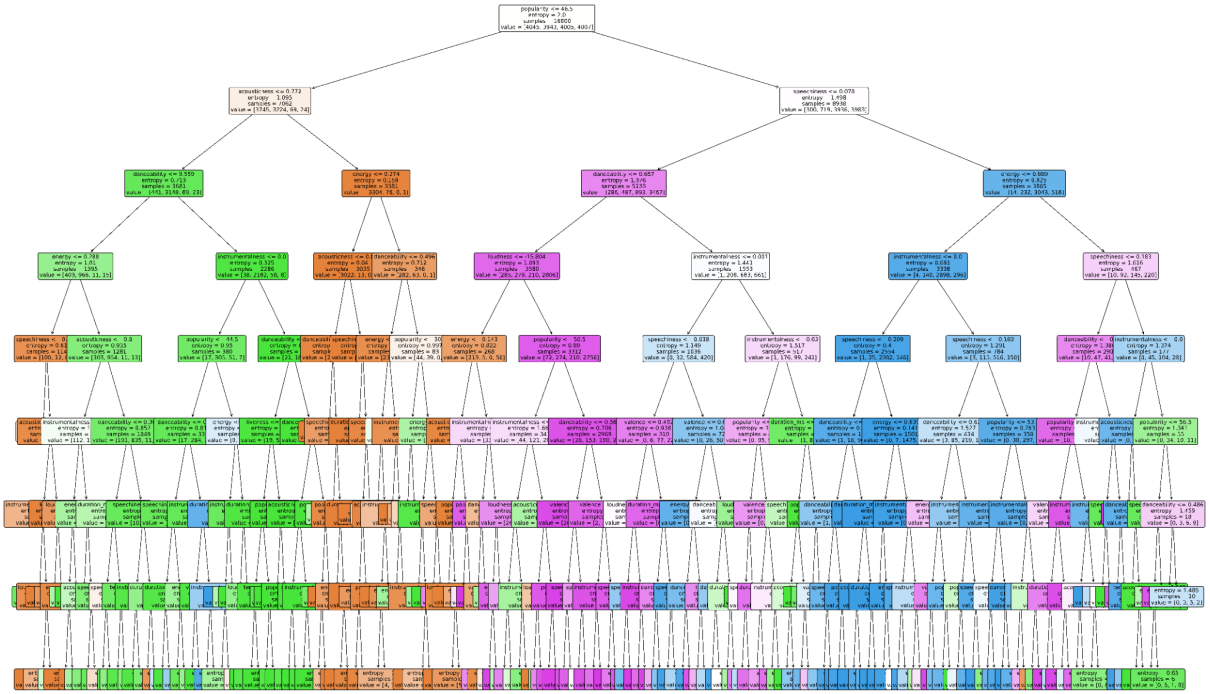
The class that is classified significantly better than the others is Classical music, with a recall and a precision of 0.95 and 0.91 respectively.

We also explored the best parameter configuration with the number of features to consider when looking for the best split, the best criterion, and the max depth. We obtained similar results to before with an entropy criterion, a max depth of 7 and considering all the features.

5.1. Visualizing and understanding the decision tree

As we mentioned in the previous section the results obtained by using a decision tree classifier with the optimal parameters have been quite satisfying. The train and test results with the chosen parameters show that the model performs reasonably well with our data. Now, we can visualize and analyze the model that we trained in order to obtain a better understanding of the decisions that it makes.

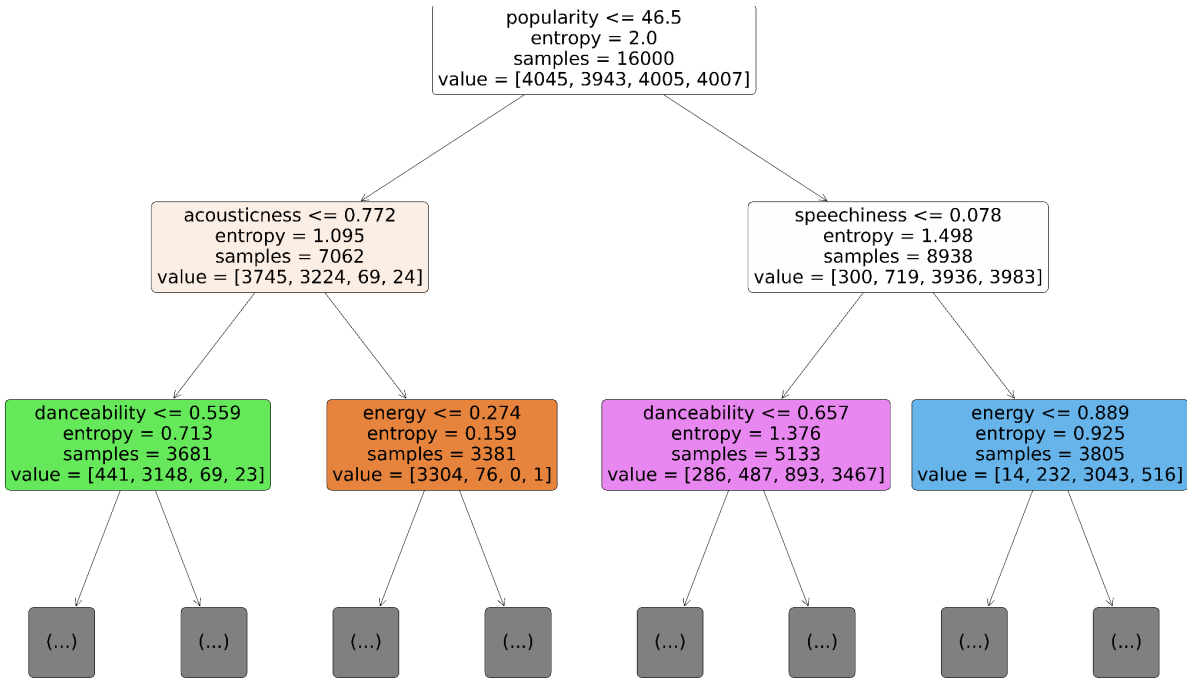
First, we have to consider the `max_depth` parameter that we found best, which is 8. This means that our tree will make 8 decisions before deciding which class the sample belongs to. This number of choices yields the best cross-validation results, since it has a nice balance between training results (the more decisions the tree makes, the better it will fit the data) and testing results (if the tree makes too many decisions, the results for the validation dataset will probably be hindered, since the decisions have been overfitted to the training data), meaning that the tree has not been overfitted to our training data.



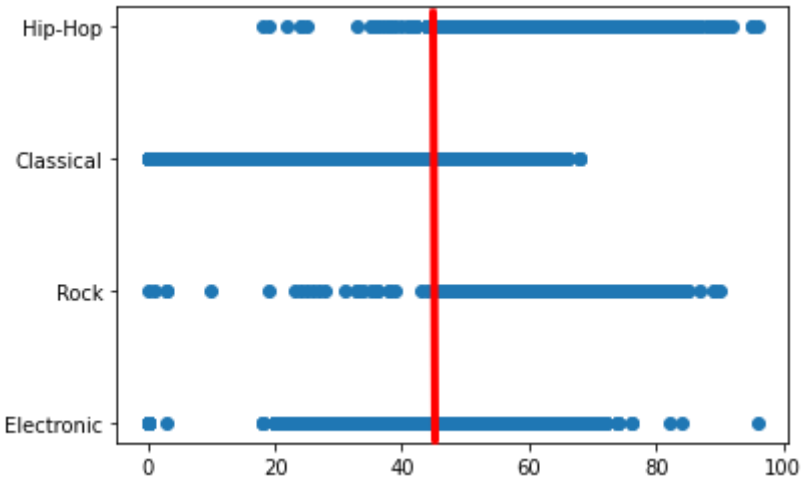
Visual representation of the whole tree.

If the tree made fewer questions (`max_depth` < 8) then the results for both the training and testing data would be worse than the ones we obtained with `max_depth = 8`. This would imply that the tree is generalizing too much and it wouldn't represent our dataset well enough.

In order to visualize and analyze our tree in a comprehensive way, we will take a look at the first three levels of the tree. For the sake of simplicity, we will not analyze the lower levels of the tree, since it can have up to $\sum_{i=0}^{8} 2^{i}$ nodes (or 511 decisions).

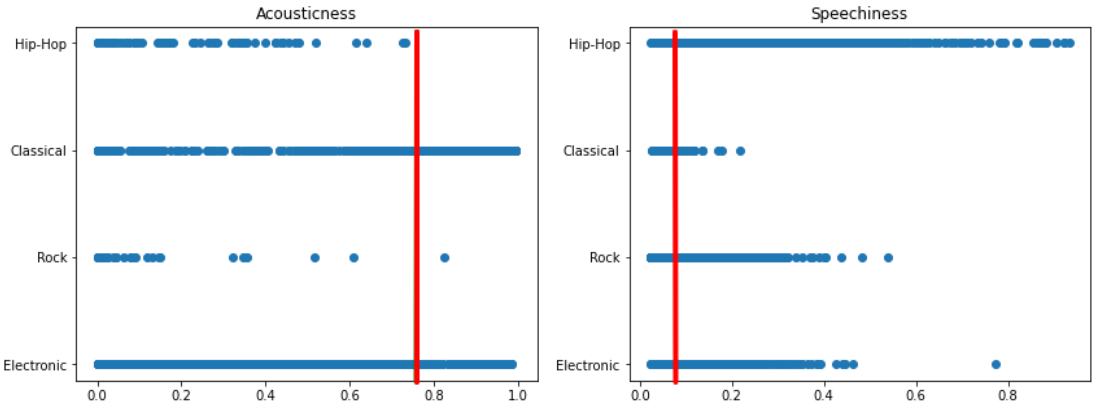


We can see that the first decision the tree makes is based on the popularity feature. It chooses a threshold to split the data. We can visualize the distribution of this variable for each music genre:

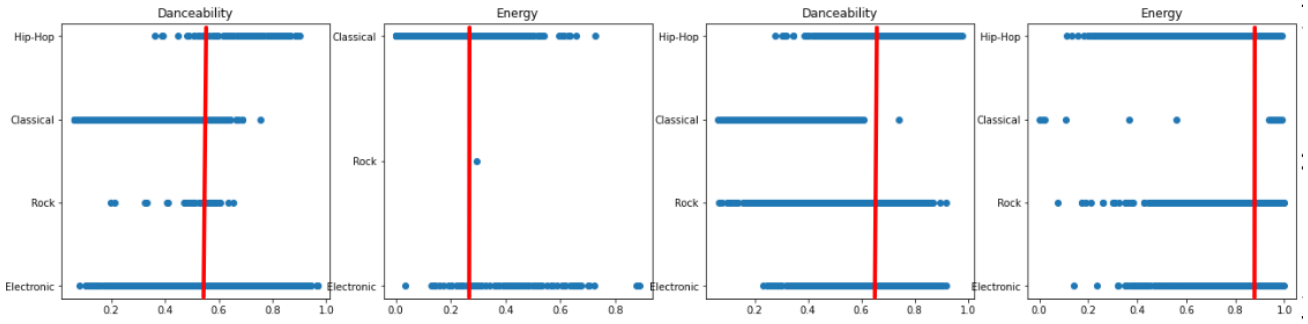


First level of the decision tree. Decision threshold is represented by a vertical red line.

We can start to see how the tree operates. This threshold splits the data such that hip-hop and rock samples will probably fall on the opposite side of classical samples. We can take a look at the following levels:



Level two of the decision tree.



Level three of the decision tree.

The following levels also pick features in order of importance and split the samples into the different classes. The thresholds (shown in red) are computed based on the distribution of the samples, amongst other factors.

We can observe the rules that were used to predict a random sample in order to have a deeper understanding of how the tree works. We have picked the first sample, which our model accurately predicted as a Rock sample. We will see how it got to this conclusion:

```
Rules used to predict sample 0:

decision node 0 : (popularity = 64.0) > 46.5)
decision node 140 : (speechiness = 0.0558) <= 0.07845000177621841)
decision node 141 : (danceability = 0.509) <= 0.6565000116825104)
decision node 142 : (loudness = -4.9030000000000005) > -15.803999900817871)
decision node 162 : (popularity = 64.0) > 50.5)
decision node 178 : (danceability = 0.509) <= 0.5664999783039093)
decision node 179 : (valence = 0.239) <= 0.3375000059604645)
decision node 180 : (popularity = 64.0) > 53.5)
```

Prediction rules used to classify a random sample.

As we mentioned before, the first feature to look at is popularity. Since it is above the given threshold, the tree then goes to node 140 (the right branch of the tree, since nodes are

sorted in a depth-first fashion). By analyzing the popularity feature, we can already determine that this sample will probably belong to either the rock or the hip-hop class.

Subsequent nodes keep *asking questions* to the sample until its class is finally determined in the final node.

6. Support vector machines

In this section we will explain the results and the procedure made with the support vector machines.

6.1. Linear kernel

First of all we tried to see how the linear SVM does. What we did to prepare the data was to read the general clean preprocessed dataset. Then we obtained the one hot encoding of the variables “mode” and “key” achieving a total of 20k rows and 26 features. Then we decided to split the dataset into two halves, in other words, using a train and test sizes of 10k rows each. This decision was made to avoid the extensive training computation times that support vector machines carry with them. We also used the stratify option to obtain balanced datasets and the random_state = 33 to obtain the same splits as the ones obtained with the other models.

Once this was done we standardized the data with the StandardScaler to make the values be rescaled into ones of zero mean and a unit standard deviation. Then, to obtain a baseline, we run the SVC function with the linear kernel and the default values (which for C is 1.0). We obtained, with a 10-fold cross validation, a really good validation accuracy of 87,769% with a small deviation among folds of only 0,0125.

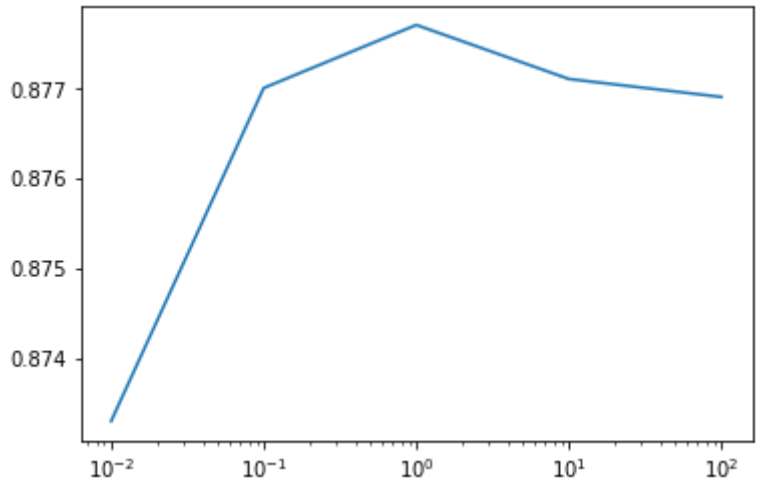
Checking the test partition we obtained an accuracy of 87,92% which is even higher than the one obtained with the cross validation. This means that this classifier is not suffering from over or underfitting which is good news. This is the confusion matrix obtained

True\Predicted	Classical	Electronic	Hip-Hop	Rock
Classical	2278	177	2	43
Electronic	101	2123	131	145
Hip-Hop	2	80	2150	268
Rock	13	59	187	2241

where we can see some mis-classifications. However, in a general view, we can see that we have a lot of values well classified which are the ones from the diagonal. Even seeing that these results were quite good we liked to go a bit further and inspect some hyperparameters.

We used the SVC function of the sklearn doing a grid search among the C values. The C parameter trades off correct classification of training examples against maximization of the decision function’s margin or, in other words, behaves as a regularization parameter in the SVM. We used an array like [0.01,0.1,1,10,100] with different values of C for the grid search.

This is the resulting plot we obtained from it:



As we can see the best results are the ones obtained with C = 1.0. We didn't search further as we can see that the inertia of the plot with higher C's tends to overfit and reduce the test score. As it's the default value we obtained, obviously, the same cross validation accuracy and the same confusion matrix with the related same test accuracy result. Therefore, the final test accuracy for this model is **87,92%**.

Furthermore we checked the number of supports of the model and found that it has 2954 and 966 of them have slacks. We used this classifier as long as it has only around 30% of support which is considered to be a low value. Also it has only 30%, more or less, with slacks which is also a good result.

Calculating more metrics we used the classification_report function and obtained the next table:

	precision	recall	f1-score	support
Classical	0.95	0.91	0.93	2500
Electronic	0.87	0.85	0.86	2500
Hip-Hop	0.87	0.86	0.87	2500
Rock	0.83	0.90	0.86	2500
accuracy			0.88	10000
macro avg	0.88	0.88	0.88	10000
weighted avg	0.88	0.88	0.88	10000

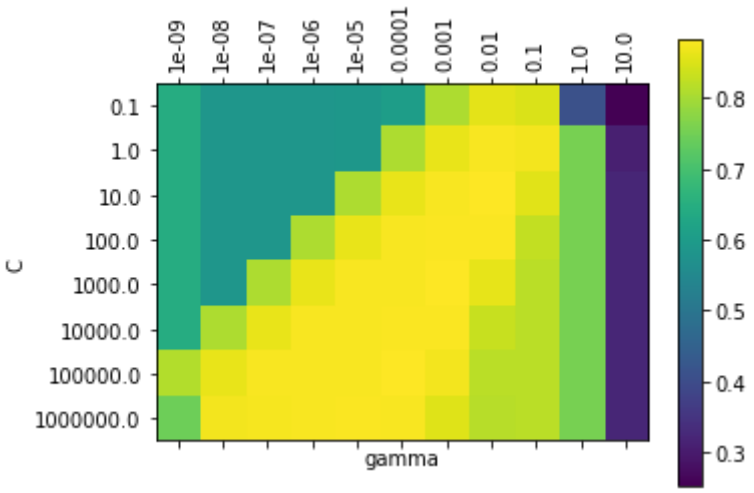
Here we can see that the classification of classical songs is the easiest one or, at least, the one easier for this model to do correctly. It has the higher precision, recall and, therefore, f1-score. More in detail we can see that all the categories have really great results getting an f1-score higher than 85% in all the classes. This shows us that our model predicts correctly without overfitting and with a good precision and recall among classes which shows us the balance in the decisions made. Now let's take a look at the SVM using the RBF kernel.

6.2. RBF kernel

In this case we used the same preprocessing explained with the linear SVM: one hot encoding and standardizing the data. Then we proceed as usual doing a baseline with the default hyperparameters to improve doing a grid search. But first of all let's explain which are the parameters that we will be touching. In this section we will only modify 2 of them, the C's (explained in the linear kernel section) and the gammas that define how far the influence of the support vector samples reach or, in a simple way, the coefficient of the RBF kernel. With that said let's see how the default model did.

We obtained a 10-fold cross validation accuracy of 88,369%, which is higher than the one obtained in the last model, with a small standard deviation of 0.0105. Now checking the test partition we obtain similar results with an accuracy of 88,39% which is even higher than the best results obtained with the linear model. Even obtaining these amazing starting results let's try to enter more in detail and find more adjusted hyperparameters.

For doing that we proceed to do a grid search with C = [10^i with i∈[-1,0,1,2,3,4,5,6]] and gamma = [10^i with i∈[-9,-8,-7,-6,-5,-4,-3,-2,-1,0,1]]. We did that experiment with only 5-fold cross validation as long as the execution times were getting quite large and unaffordable. Once that is said we computed the results and plot them in a grid with a colorbar indicating the validation accuracy obtained:



As we can see there are really good results in the second diagonal of the grid. It's quite difficult at first sight to know which is the best value but we obtained that the best combination is the one given by the usage of C = 10.0 and gamma = 0.01. We didn't search further as we can see that higher C values are not obtaining as good results as the ones in the middle. Furthermore, these higher C values are maybe overfitting to our training dataset.

Let's inspect a bit further this best found combination of parameters. Doing a 10-fold cross validation we obtained an accuracy of 88,59% which is a bit higher than the one obtained with the default values. The standard deviation of these folds is 0.011. This specific classifier obtains 3040 supports (a bit more than with the linear kernel), but, again, about 30% with only 883 of them slacked versus the 966 slacked with the linear kernel.

On the other hand, using the test partition, we obtained an accuracy of **88,62%** improving, again, the results obtained with the linear kernel. This is the resulting confusion matrix:

True\Predicted	Classical	Electronic	Hip-Hop	Rock
Classical	2302	161	3	34
Electronic	73	2143	135	149
Hip-Hop	1	73	2174	252
Rock	14	70	173	2243

We can see some mis-classifications but, in a general view, we can see that we have a lot of values well classified which are the ones from the diagonal. It's clear that the easiest is the classical genre and the hardest is rock. Nevertheless, we think that this is a good confusion matrix because if we indagate a little bit more we can extract the following metrics

	precision	recall	f1-score	support
Classical	0.96	0.92	0.94	2500
Electronic	0.88	0.86	0.87	2500
Hip-Hop	0.87	0.87	0.87	2500
Rock	0.84	0.90	0.87	2500
accuracy			0.89	10000
macro avg	0.89	0.89	0.89	10000
weighted avg	0.89	0.89	0.89	10000

that shows us the correctness and great performance of this classifier. Here we can see in a clear way that the precision to classify rock and classical is quite different with classical being quite easy and rock quite complicated. However we didn't notice before that rock has a great recall. Overall the f1-score shows us, again, that classical is the easiest class to classify and the others are a bit more complicated but still obtaining amazing results.

To conclude we can say that the RBF kernel helped us to get a better model that can separate better our data and classify in a fashion way our data on its correct classes with approximately 89% of accuracy.

7. Ensembles

7.1. Voting Scheme

This method is based on using other weaker models to classify our task and then using the results of those models to obtain the final result. The first parameter we had to adjust is which models we are going to use. We could use all the models we had done in the previous sections but we wanted to check if using weaker models such as Naive Bayes, Decision Trees and KNN we could improve the performance of them and build a strong model.

Of course we could add stronger models such as Support Vector Machines or maybe Neural Networks but these models are so good by themselves that most of the time we could just use the result obtained from them and we can rest assured that it is with high probability fine. So at the end we used 4 models, Naive Bayes, KNN with the best hyperparameters we found in the previous sections, decision trees with the best hyperparameters and we also added a decision tree with the base parameters.

We added the decision trees with the base parameters because with the normalized data, this tree performs better than the one with hyperparameters, although it is probably due to overfitting. The next hyperparameter we had to choose is what type of voting we want to perform, hard voting or soft voting. We first tried hard voting and we got an accuracy of 0.778 which is not a bad result but knowing that all the weaker models perform slightly worse this result disappointed us.

Next we tried soft voting, so we need to assign weights to each model. We expect that soft voting performs better because it is recommended when we already have well-calibrated classifiers. We could just assign the same weight to every model but we thought that assigning weights based on their performance using cross validation would be a better idea. If we observe the accuracy of the models:

Accuracy: 0.617 [Naive Bayes]

Accuracy: 0.710 [Knn]

Accuracy: 0.714 [Dec. Tree with param]

Accuracy: 0.769 [Dec. Tree]

We can observe that naive bayes perform worse than the others, so we might assign a lower weight to it. Also we predicted that even though the decision tree performs better than the others, it is probably due to overfitting so it’s weight will be the same as KNN or the other decision tree. The accuracy obtained in the previous table is not the same we have obtained with the models in the previous sections because we had to train them using normalized data. At the end we decided to give naive a weight of 1 and all the others a weight of 2. As we expected, soft voting obtained an accuracy of 0.8 which is a way better result but worse than what we have obtained

using other meta-methods. Now we are going to take a brief look at the classification report obtained.

	precision	recall	f1-score	support
Classical	0.93	0.90	0.91	955
Electronic	0.83	0.81	0.82	1057
Hip-Hop	0.74	0.75	0.74	995
Rock	0.71	0.75	0.73	993
accuracy			0.80	4000
macro avg	0.80	0.80	0.80	4000
weighted avg	0.80	0.80	0.80	4000

As we have seen in other models, we have very good results when it comes to classifying classical music, but classifying hip-hop and rock our model does not perform as well. Even though we have good results, this model is not nearly close to being the best one, primarily because of that low f1 score in hip hop and rock.

7.2. Bagging

For this meta method we tried different approaches to get the best accuracy possible with our dataset. We will try to find out which number of estimators gives us the maximum accuracy. So, for doing this, we will choose two weak models and do bagging with them. In our case, we chose 2 specific models for bagging, KNN with the best hyperparameters found in the KNN section and Decision trees with the default parameters. With both we will do a grid search of the number of estimators. We will create the models with the best hyperparameters and we will compare them.

The possible number of estimators that we chose were the following: [1,2,5,10,20,50,100,200, 250, 300, 350, 400]. So, when we did the grid search with KNN, we found out that the best number of estimators for bagging was 350, obtaining an accuracy of 0.721. Then, we did a grid search with Decision trees and we found out that the best hyperparameter with decision trees is 300 estimators, obtaining an accuracy of 0.8975. We can appreciate that the Decision model is a lot

better than the Knn model, which is logical because if we compare just one model of Decision trees with one model of Knn we can say that obviously the Decision tree model is better. We did an extra grid search with Decision trees looking nearby 300. Trying to find a better hyperparameter near the value 300. We found 315 estimators with an accuracy of 0.899. We can see that the difference is not very big but we have made an improvement. So, we have found the number of hyperparameters that gives the best accuracy with bagging, which is 315 estimators. Obviously we can keep increasing the number of estimators and we will find another value with better accuracy but that will take a lot of time. We consider that 0.899 is a model with amazing results. So, our final accuracy is **0.899**

Now, we will study the Decision Trees Bagging model with estimators. Now, let's see the confusion matrix of the model with 315 estimators.

True\Predicted	Classical	Electronic	Hip-Hop	Rock
Classical	**903**	46	1	5
Electronic	17	**940**	42	58
Hip-Hop	1	16	**888**	90
Rock	10	30	91	**862**

We can appreciate that we have mis-classifications for every genre but, looking at the big picture, we can see that the majority of the values are well classified, and these are the values in the diagonal. One curious thing about this matrix is that classical music and hip hop don't have mis-classification between them. There is only one case where we predict hip hop when it should be classical music. We tried this several times and we always had this quality. Now, we will study the classification report obtained with this model.

	precision	recall	f1-score	support
Classical	0.97	0.95	0.96	955
Electronic	0.91	0.89	0.90	1057
Hip-Hop	0.87	0.89	0.88	995
Rock	0.85	0.87	0.86	993
accuracy			0.90	4000
macro avg	0.90	0.90	0.90	4000
weighted avg	0.90	0.90	0.90	4000

We can appreciate that the genre that we best classify is classical music, which is very similar to previous models. Another thing that repeats is that we have a lower precision to classify rock in relation with other genres. So we can say that our model classifies classical music with ease, but it struggles more with the rock genre. But in general we have a very high probability to classify in the good genre. Finally the f1-score confirms us one more time

that classical is the easiest class to classify and the others are a bit more complicated but our model has better results than models seen in previous sections.

7.3. Random Forest

Now, we will try to find the best hyperparameters for randomforest. For this meta method we did a similar process as bagging. As said before, we chose random state 33 to create our training and testing dataset. We tried to find the best hyperparameters. More precisely we chose the following hyperparameters: number of estimators, criterion used, maximum depth for the decision trees, and maximum features. First of all we did a normal Random forest classifier which had the default hyperparameters. And then we did the search for the best hyperparameters for this type of classifier.

With the default random classifier we got an accuracy of 0.901 with the testing dataset, which is a pretty good value if we compare to other strong models like SVM and decision trees. We can see that the default random forest classifier is better than SVM. Now let's see if we can improve this model. We did a grid search with the different hyperparameter and we obtained that the best hyperparameter where the following:

criterion : entropy

maximum depth : 90

maximum features : sqrt

number of estimators : 250

With these values we got an accuracy of 0.9035, which has improved in comparison to the default one. We can appreciate that we are limiting the maximum depth of the trees, which could seem like it is incorrect because if we don't limit we could get more accurate results. But that is not true. We did a model with the same hyperparameters as we have mentioned before but without limiting the maximum depth. With this model we got an accuracy of 0.902, which is better than the default model but worse than the model that limits the maximum depth. The reason for this is because we can get better accuracy on the training dataset, but worse results on the testing dataset. The reason for this is overfitting. So to avoid overfitting and get a better model, we have to restrict the maximum depth. We could increase the number of estimators and see if we can improve even more the accuracy but with our resources, it would take too much time. So, the best model obtained with Random forest has an accuracy of **0.9035**.

Now, let's look the confusion matrix of this model, which is the following one:

True\Predicted	Classical	Electronic	Hip-Hop	Rock
Classical	**893**	55	1	6
Electronic	18	**939**	39	61
Hip-Hop	0	9	**897**	89
Rock	8	19	83	**883**

We can appreciate that we have mis-classifications for every genre but, in general, classify correctly the majority of them. We can appreciate it is pretty similar to Bagging with Decision trees which makes sense because Random Forest is kind of a bagging with Decision trees but with an extra parameter a. Again, classical and hip hop have just one mis-classification between them. There is only one case where we predict hip hop when it should be classical music. Very similar to what we obtained before. Now we are going to take a brief look at the classification report obtained.

	precision	recall	f1-score	support
Classical	0.97	0.94	0.95	955
Electronic	0.92	0.89	0.90	1057
Hip-Hop	0.88	0.90	0.89	995
Rock	0.85	0.89	0.87	993
accuracy			0.90	4000
macro avg	0.90	0.90	0.90	4000
weighted avg	0.90	0.90	0.90	4000

We can appreciate that the precision to classify rock is the worst and to classify classical is the best among all the models. So we can say that our model classifies classical music with ease, but it struggles more with the rock genre. But, we have to say that the results are still pretty good. Finally the f1-score shows us, again, that classical is the easiest class to classify and the others are a bit more complicated but our model has better results than models seen in previous sections.

7.4. Boosting

The last meta-method we had to try was AdaBoosting. As we have explained before, we just tried this method using weak learners, such as naive bayes or decision trees. This method in particular has 2 hyperparameters, which learner is going to be used and how many iterations of boosting we want to use. Selecting the learner was easy because we just have to choose between two: Naive Bayes and Decision tree. We couldn't use knn because it is not supported by the sklearn library. So what we did was try the method with both learners. And when we were trying decision trees, we used the one fine-tuned and the base decision tree. The next step was selecting the number of iterations, so we proceed as we have done until now. We used a grid

search with a huge range of iterations [1,2,5,10,20,50,100,200] for each method. This are the result we have obtained:

Fine-tuned decision tree: 0.89075 n_estimators: 200

Base decision tree: 0.9015 n_estimators: 200

As we noted in the previous section, the base decision tree performed better than the fine-tuned but the difference is not significant, we could have looked for the best number of estimators but it took very long and also the improvement is insignificant. And comparing these results with the decision tree without boosting, we can see it has got better performance, so the boosting method is a good strategy for our dataset. Surprisingly, Naive Bayes didn't benefit from boosting at all, the best performance it has is just with the base training. Now let's take a look at the classification report of the best model we have obtained using boosting i.e. base decision tree.

	precision	**recall**	**f1-score**	**support**
Classical	0.97	0.93	0.95	955
Electronic	0.89	0.89	0.89	1057
Hip-Hop	0.88	0.89	0.88	995
Rock	0.86	0.88	0.87	993
accuracy			0.9	4000
macro avg	0.9	0.9	0.9	4000
weighted avg	0.9	0.9	0.9	4000

Here we have one of the best models so far we have created. The results compared to the original model which is decision trees are almost indistinguishable. As in the other models, classifying Classical music is very easy but electronic, hip-hop and rock are not that easy, even though the difference this time is not that huge. The f1 scores of all of them are greater than 0.85, so we can rest assured that our model almost always classifies every song correctly.

8. Comparison and conclusions

After performing experiments with the models mentioned above, here are the results that we obtained when measuring the accuracy of the predictions that the models made against our test dataset:

Classifier	Test accuracy
Categorical Naive Bayes + Gaussian Naive Bayes	83%
One Hot Encodig + Gaussian Naive Bayes	78,6%
KNN	71,23%
Decision trees	87%
SVM lineal	87,92%
SVM RBF	88,62%
Ensembles Voting	80%
Ensembles Bagging	89.9%
Ensembles Random Forests	90.35%
Ensembles Boosting	90%

In these experiments we have used a wide variety of the models and techniques studied in the course. The way in which we presented our experiments goes more or less in ascending order of their complexity. We can notice how KNN obtains the poorest results with about 20% of difference with the best models that achieve around 90%. We think that this is because KNN could not find a good metric to define a good distance among samples which is a known problem for this method. Even so, the other methods perform better and we are quite proud of the results we obtained with them.

The first models are fairly straightforward and easy to understand and interpret. They can be really suitable for simple datasets with linear correlations between variables. These models are also relatively easy to implement and don't require much time to train. Although most of them present acceptable results, the more complex models perform much better with our data.

We can observe a dramatic increase in accuracy when using Decision Trees and Support Vector Machines against other simpler models such as Naive Bayes and KNN. Although these models do perform significantly better, they also have more hyperparameters that need to be adjusted in order to get the most out of the models. They are usually more complex than linear models, and we need to perform a deep analysis in order to understand why they behave the way they do.

Lastly, we used ensembles to try to improve our results. For most of the ensembles used, we observed that the results did show an improvement over the previous results. These results match our expectations for the Ensembles part of the experiments, since our intuition led us to believe that the combination of multiple good models would yield better results than the models making predictions individually.

To conclude, we would like to mention that the best method selected is the Random Forest. This selection was made considering the performance of this method and the straightforward usage of it. This classifier is known to work fine with different amounts of data and different numbers of dimensions. It is also known that it didn't have problems with dimensionality as KNN or other models. Furthermore, the data can be introduced without any normalization or standardization as the model didn't need the values to be in certain or similar intervals.

These are the statistics results we obtained from 100 executions getting the test score with this final model selected.

```
Mean:  0.9017878787878788
Median:  0.90175
Population standard deviation:  0.0011603528483967135
Population variance:  1.3464187327823665e-06
Sample standard deviation:  0.001166257990087704
Sample variance:  1.3601576994434113e-06
```

We can see a mean about 90% with really small deviations and variances meaning that we are quite confident about the mentioned mean result. Hence, the best results we obtained are about 90% accuracy.

For all these things, and the fact that obtains the best results, we consider this model the perfect one for this type of problem we had and our final decision was made.

With this report we learn a lot about many classification models and how to deal with complicated problems as the one that occupies this report. Finally we would like to mention that we are very proud of the results obtained because we were not sure at the start if we would be able to find a feasible solution.