POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

A comparison of ML models for musical genre classification

Numerical Analysis for Machine Learning
Computer Science and Engineering

Author: **Alberto Pirillo**

Professor: **Prof. Edie Miglio**

Academic Year: **2022-2023**

Contents

Introduction

The goal of this project is to recreate and explore further the study on the automatic classification of musical genres originally proposed in the paper *Classification of musical genre: a machine learning approach* [1].

The analysis was recreated using a different data set, the GTZAN data set [2].

Furthermore, given that Java is no longer the language of choice for machine learning applications, this work will rely on the Python programming language, and in particular mostly on the libraries *scikit-learn* [3] and *keras* [4], which are among the most popular ML/DL Python libraries at the moment of writing.

Data set

The GTZAN data set is the most used public data set for evaluation in machine listening research for music genre recognition. The files were collected in 2000-2001 from a variety of sources including personal CDs, radio, and microphone recordings, in order to represent a variety of recording conditions.

The data set consists of a collection of 10 genres (blues, classical, country, disco, hip-hop, jazz, metal, pop, reggae and rock) with 100 audio files each, all having a length of 30 seconds. This means that the original data set is composed of just 1000 samples. In order to increase the amount of data at our disposal, we split each song into 10 tracks each one with a length of 3 seconds. In this way, we are able to work with 10000 samples. This is beneficial since it is known that a larger number of samples helps improve the performance of our model.

It is worth noting that this splitting has some drawbacks since we are introducing some dependency among the samples, given that many samples were coming from the same song. However, we will neglect this aspect in our analysis since increasing the number of samples brings more significant benefits w.r.t. the disadvantages introduced by this correlation.

In our analysis, we will consider the file *features_3_sec.csv* available in the data set. This file has an entry for each one of the tracks of 3 seconds we will consider. For each track, it contains the mean and the variance of a list of features extracted from the corresponding audio file.

Problem description

Supervised classification is one of the core tasks solved by Machine Learning.

Following the lead of the paper [1], two categorization models were studied:

- Multi-class classification: we are considering all the samples and all the musical genres at the same time. Our goal is to assign each one of the samples to one of the genres.
- Binary classification: we are considering the problem of determining whether a single sample belongs or not to a given genre, therefore we are considering only one genre at a time. We will need to train multiple independent models (one for each genre) considering only a subset of the samples. This is because in classification problems it is a good practice to have a balanced amount of data between the classes. Each model was trained considering all the samples of the corresponding genre and an equivalent amount of samples randomly selected from the other classes.

The most common metrics used in classification problems are accuracy, precision and recall. In order to evaluate the performance of the classification models, we will use accuracy. This is in order to be coherent with the original paper, and potentially to compare our results with theirs.

Project steps

As in any other data science work, the investigation begins with an Exploratory Data Analysis (EDA) to study the information contained in the GTZAN data set. It is covered in Chapter 1.

Then, following the example of the original paper [1], we trained multiple machine learning models and evaluated their performance using both static partitioning (i.e. train-validation split) and dynamic partitioning (i.e. K-fold cross-validation). Machine learning models are covered in Chapter 2

Finally, it was decided to tackle the problem using a more complex and more modern

model not present in the original paper: feed-forward fully-connected neural networks. This is covered in Chapter 3.

A comparison between the performance of all those models is then presented in Chapter 4.

All materials related to this project, including the source code, the final models, and the data that was stored in the database, are available in the project's GitHub repository [5].

1 | Exploratory Data Analysis

The analysis was conducted in a *Jupyter Notebook*, relying on the *pandas* library for data manipulation and on the *matplotlib* and *seaborn* libraries for data visualization.

The goals of this EDA are the following:

- Look for missing values and inconsistencies
- Find the most suitable data types
- Perform a correlation analysis to understand how to predict the label
- Visualize the data through the use of Principal Component Analysis (PCA)

The first step was checking for missing or duplicate data inside the data set.
Neither missing nor duplicate data was found.

Then, the target variable, i.e. the musical genre of the songs, was studied. It can only assume 10 different values, which are the 10 musical genres we are considering. This means that the target variable is a categorical variable. We can also observe that the classes are only slightly imbalanced w.r.t. one another, therefore we do not expect this to be an issue in the training of the models.

1.1. Setting data types

The data types automatically inferred by *pandas* are known to be sub-optimal. In order to use the least amount of memory possible, an analysis of the values taken by each feature was conducted. In particular, the absolute value of the maximum value and of the minimum value of each numerical variable was plotted, as shown in the bar plot below.

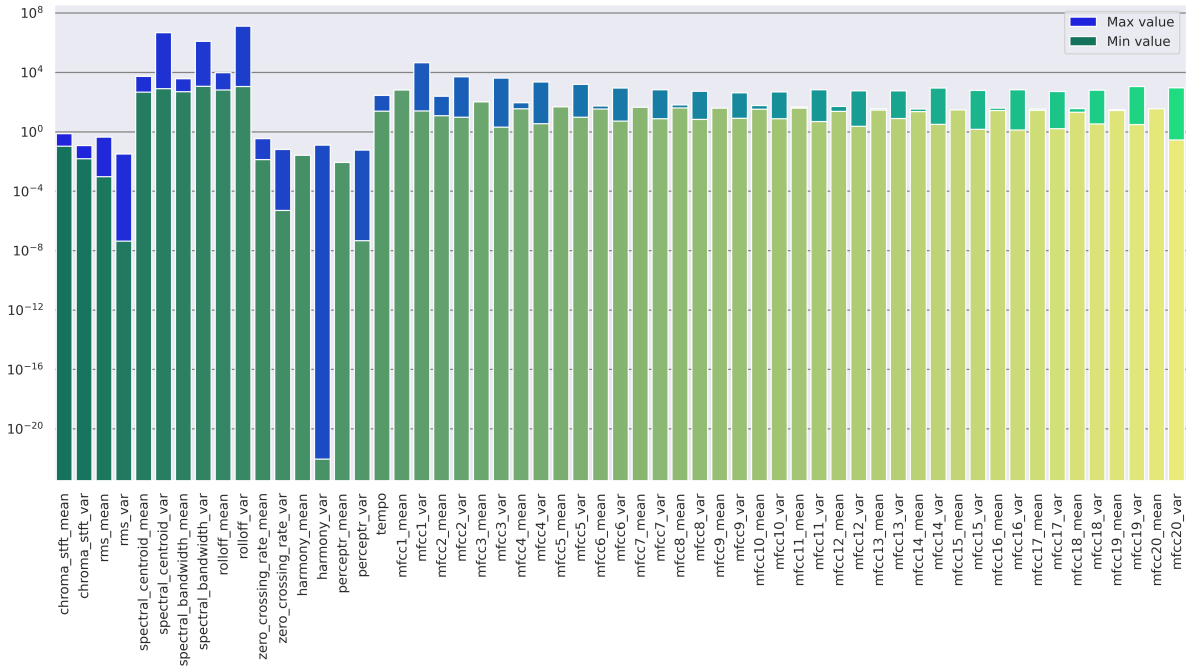

Figure 1.1: Minimum and maximum values of the features.

It is evident that no variable presents an absolute value that is larger than 10^8. Therefore, there is no need to use the *float64* data type assigned by *pandas*, we can safely use the *float32* data type without any loss of information. Furthermore, we assigned the data type *category* to the only remaining categorical variable, i.e. the *label*, leading to additional memory saving.

At the end of this phase, the size in memory of the data set was reduced by a factor of almost 2.

1.2. Correlation analysis

The goal of this phase was to find out how the features in the GTZAN data set are related to the target variable. Given the lack of expert knowledge in this field, this analysis is the only way for us to identify which are the most promising features. The correlation is computed through the use of the Predictive Power Score (PPS) [6]. The PPS is a more powerful metric w.r.t. the linear correlation coefficient since it can also detect non-linear relationships between features. The score ranges from 0 (no predictive power) to 1 (perfect predictive power).

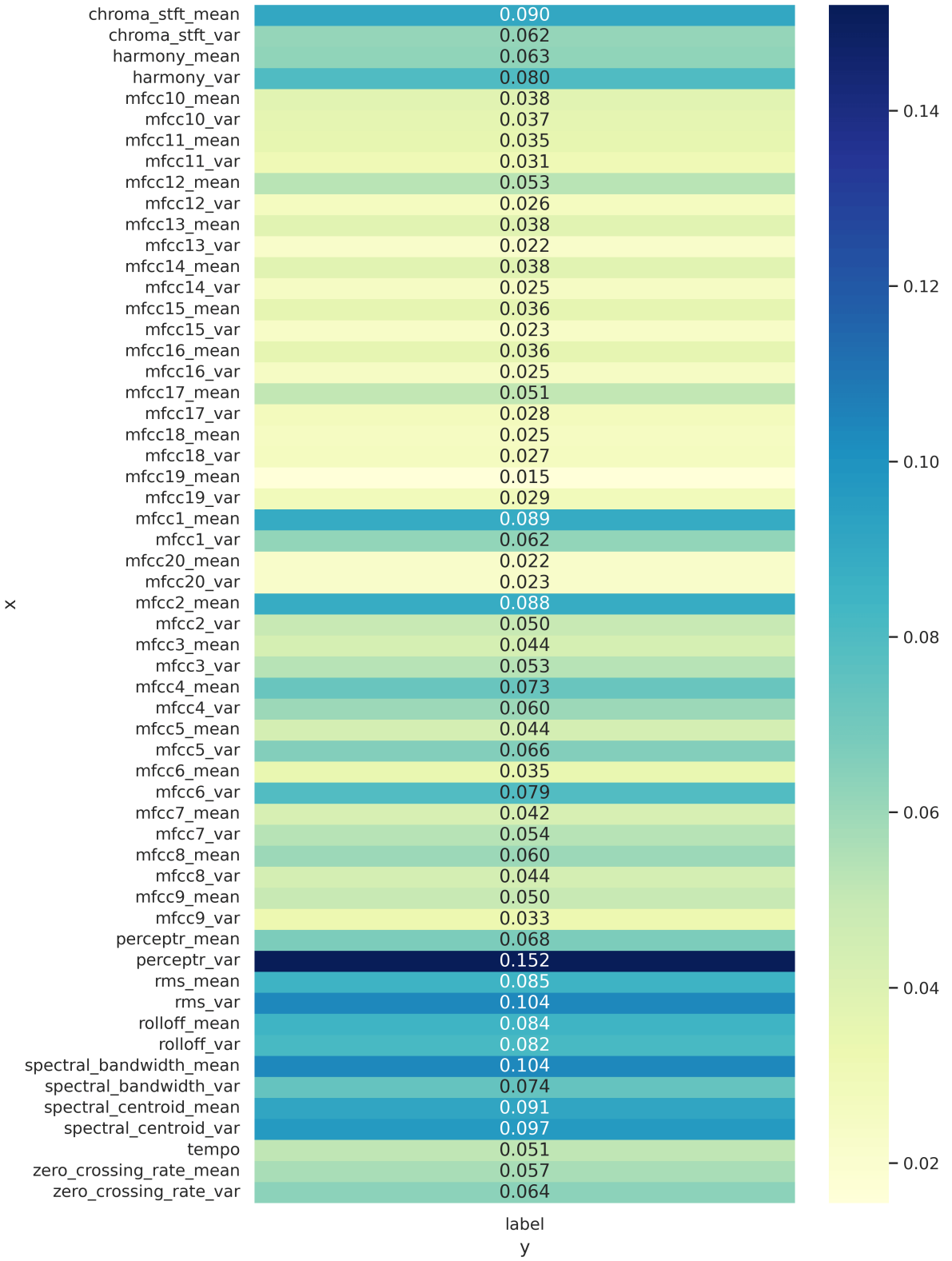

Figure 1.2: Predictive Power Score w.r.t. the label of all the features.

From the above chart, we can see that many features have a very weak correlation.
We plot below the features with a reasonable level of correlation:

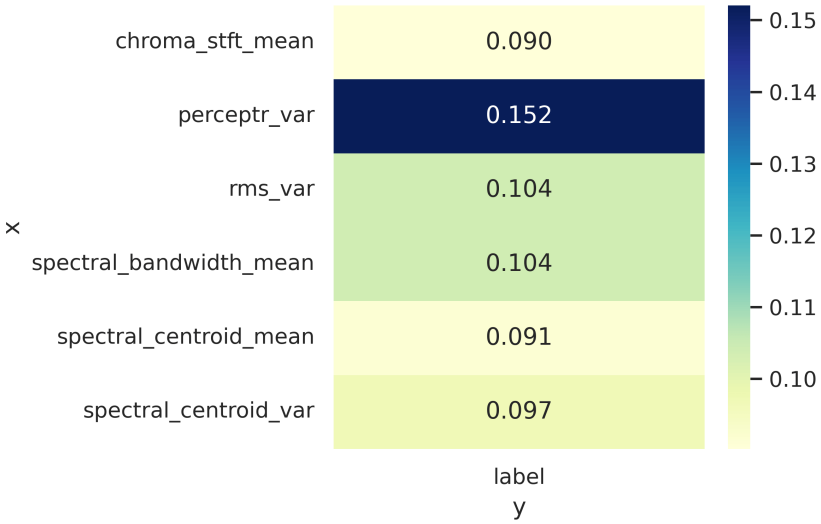

Figure 1.3: Features with the highest PPS w.r.t. the label.

If we were to perform feature selection, we will certainly keep these features.
However, in this project, we are going to use all the features anyway since the data set is not that large, and since we trust the fact that all the features in the original data set were selected because they were meaningful.

1.3. Data visualization

In order to visualize the data and understand its structure, we will use Principal Component Analysis (PCA), computed through Singular Value Decomposition (SVD). In the following chart, we can see the singular values of the matrix that represents the data set after the PCA was applied.

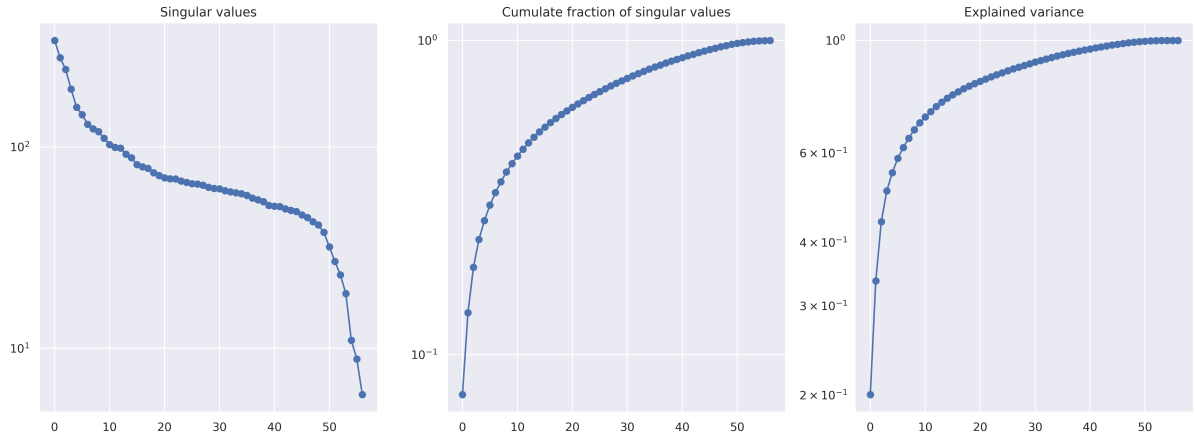

Figure 1.4: Singular values from PCA and their explained variance.

We could remove some singular values in order to reduce the size of the data leading to a negligible loss of information. However, size is not a concern with this data set and we are not interested in size reduction. Instead, we can understand from the explained variance of the singular values that the data we are working with has a complex structure. In fact, we can see that we need to keep a lot of dimensions in order to explain 90% or more of the variance. The following table contains some significative examples of the amount of variance that we can explain using only a subset of the singular values:

Number of singular values	Explained variance
1	20.05%
2	33.60%
3	43.94%
4	50.54%
31	90.67%
39	95.31%
57	100.00%

This suggests that the problem at hand is not easy and that low-complexity models will likely struggle at finding a good decision boundary, at least in the multi-class case.

We will visualize the data after projecting it onto the principal directions, which we know are the directions that contain the highest variance.

We can see a scatter plot of the two first principal components of each sample:

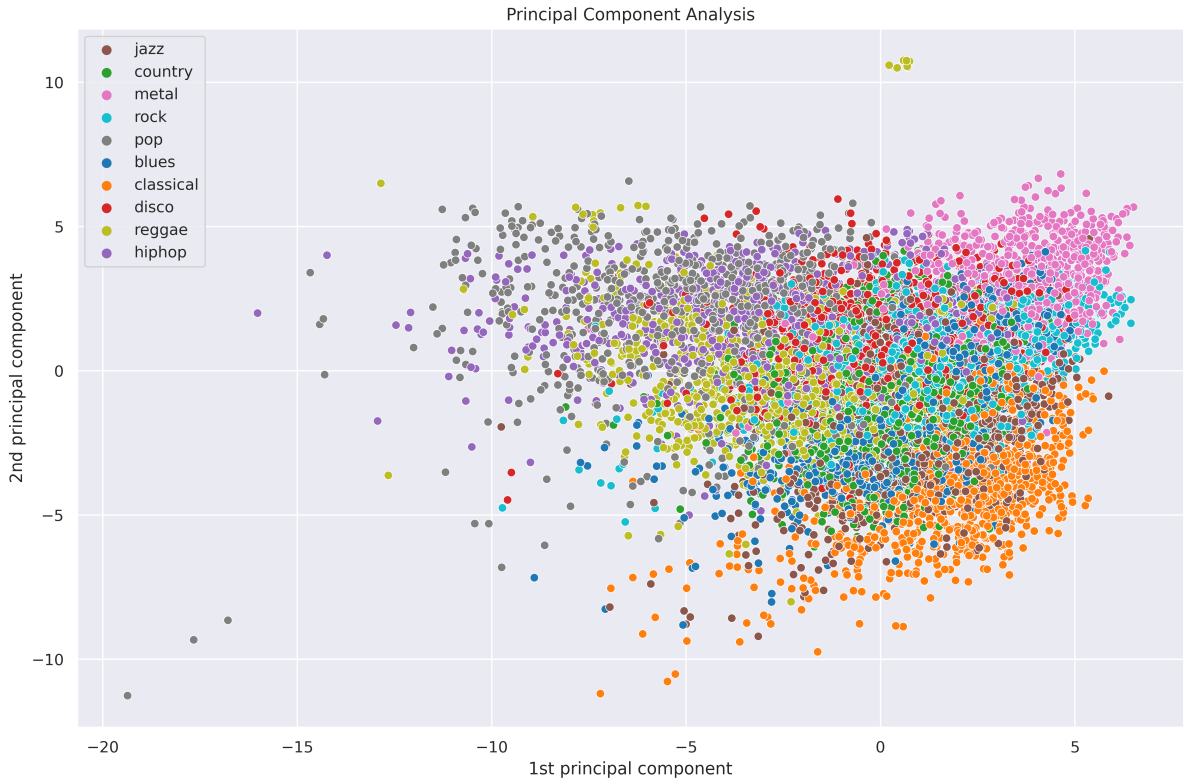

Figure 1.5: Scatter plot of the data projected on the two first principal directions.

And a scatter plot of the three first principal components of each sample:

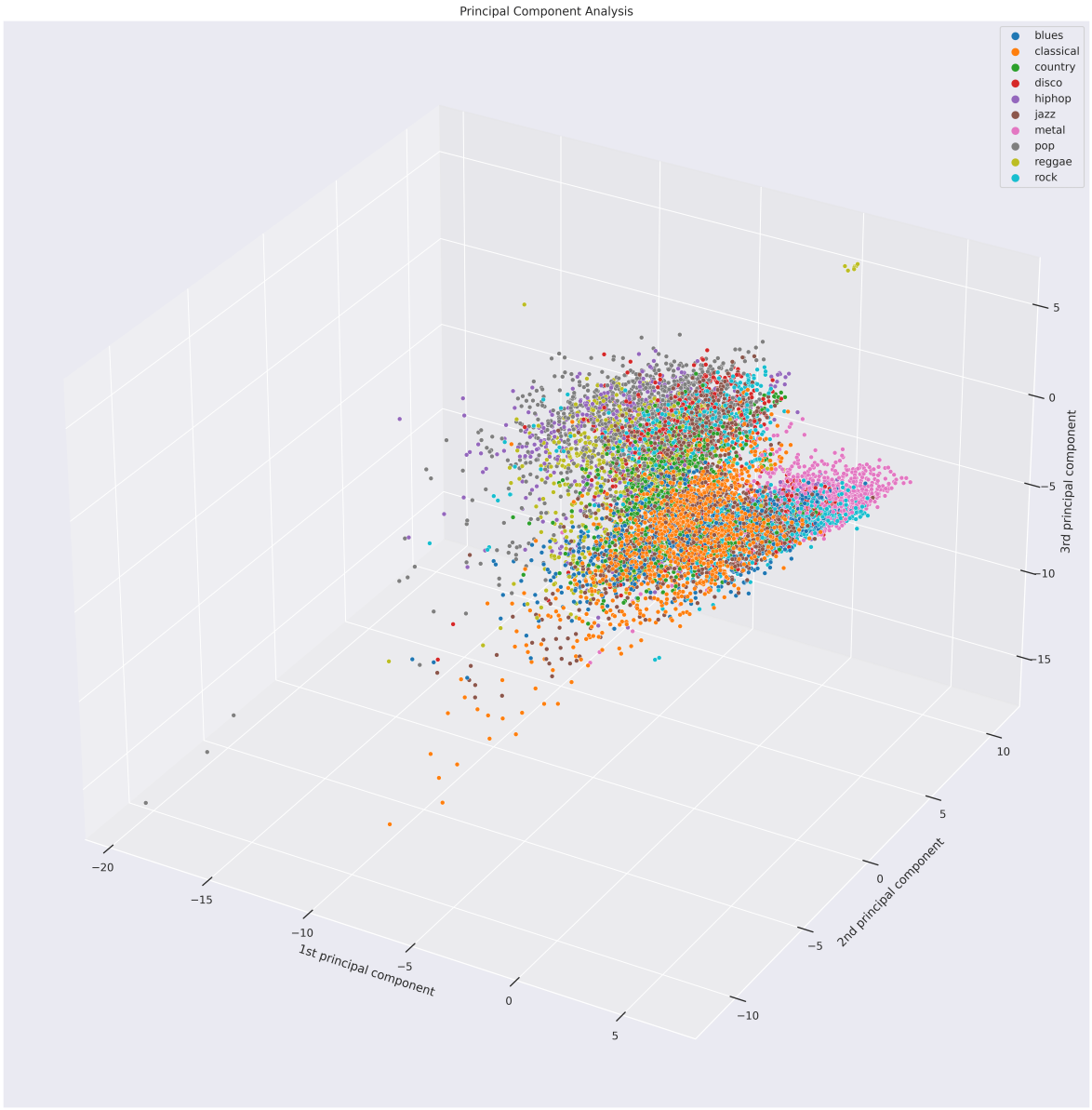

Figure 1.6: Scatter plot of the data projected on the three first principal directions.

As expected, it is not easy to visually cluster the data in either of the two plots, even if the situation gets slightly better in the second one. This is expected since as explained earlier we need to use many singular values in order to explain a significant amount of the variance.

2 | Machine Learning models

In the first part of our analysis, the two classification problems we are considering were tackled using multiple Machine Learning models. The models Naive Bayes (NB) and Voting Feature Intervals (VFI) are the same ones used in the original paper [1]. On the other hand, the CART Decision Trees (DT) and K-Nearest Neighbors (K-NN) models are more modern variants of the ones considered in the paper.

Every model was considered in a separate notebook. Given that those notebooks ended up sharing a large amount of code, many methods were refactored into a separate library file, called *lib.py*, which was then imported into the other notebooks just like a regular library. This file contains many functions useful to perform pre-processing, normalization, label encoding, class balancing and plotting.

2.1. Naive Bayes

Naive Bayes methods are a set of supervised learning algorithms based on Bayes' theorem, with the additional "naive" assumption of conditional independence between every pair of features.

In spite of this over-simplified assumption, Naive Bayes classifiers behave quite well in real-world situations. They require a small amount of training data to estimate the necessary parameters and are known to be extremely fast compared to more sophisticated methods.

On the other hand, although Naive Bayes is known to be a decent classifier, it is also known to be a bad estimator, meaning that the probability outputs that it returns should not be taken too seriously.

In particular, in our analysis, we used the Gaussian Naive Bayes algorithm, which assumes the likelihood of the features to be Gaussian. The code is provided by the library *scikit-learn* [3]. Every step of this analysis is reported in the notebook *Naive_Bayes.ipynb*.

Naive Bayes can work with a categorical target variable, therefore it was enough to normalize the data through standardization in order to fit the model.

In the following charts, we can see the accuracy of the multi-class classifier and of the binary classifiers with different sizes of the train-validation split and with a different number of folds for the K-fold cross-validation.

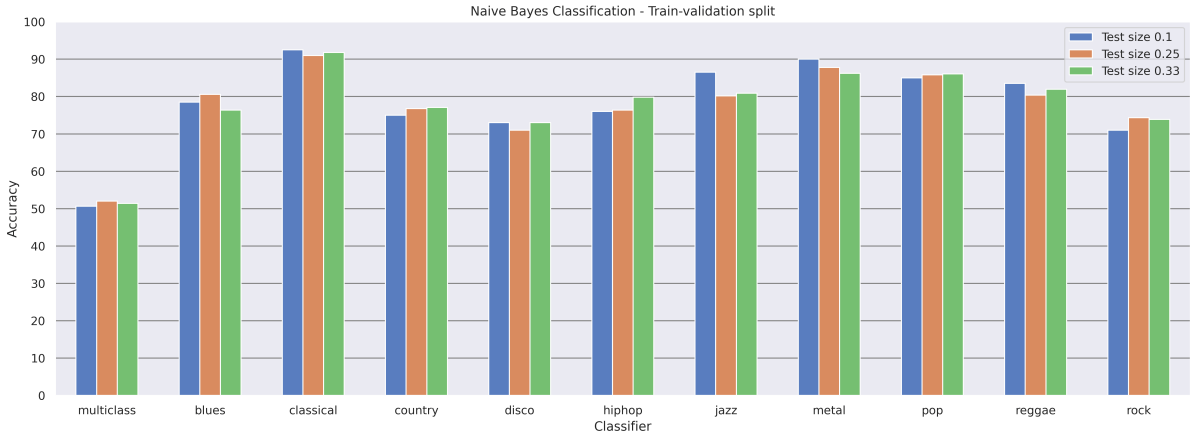

Figure 2.1: Accuracy of the NB classifier with the train-validation split.

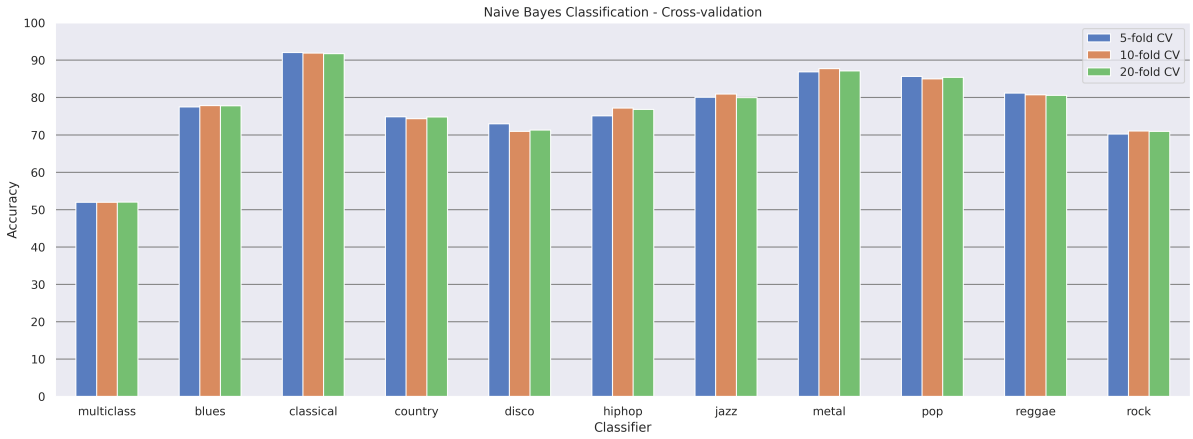

Figure 2.2: Accuracy of the NB classifier with K-fold cross-validation.

We can see that the accuracy, in particular in the multi-class case, is far from good. This is expected since, as stated in the EDA, the multi-class classification problem is not easy, and a very simple model like Naive Bayes is not able to handle well this complexity.

The following is the confusion matrix of the multi-class Naive Bayes classifier.

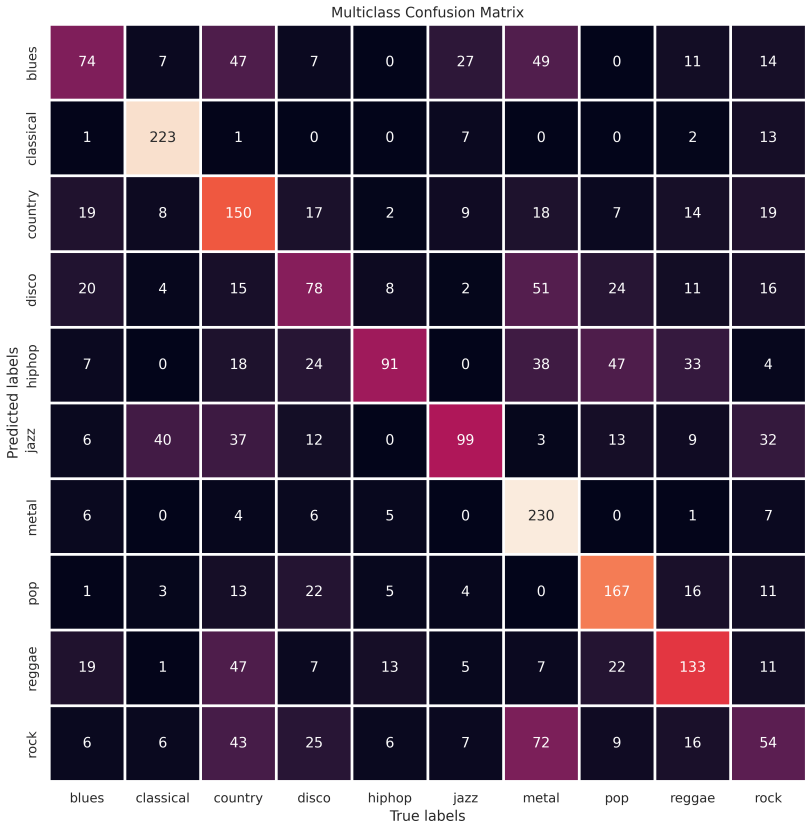

Figure 2.3: Confusion matrix of the multi-class NB classifier.

2.2. Voting Feature Intervals

Voting Feature Intervals (VFI) is a deterministic supervised classification model similar to Naive Bayes. VFI constructs intervals around each class for each feature. Class counts are recorded for each interval on each feature and the classification is performed using a voting scheme.

The code is provided by the *vfi* Python library, which is maintained in a GitHub repository [7]. Every step of this analysis is reported in the notebook *VFI.ipynb*.

VFI can work with a categorical target variable, therefore it was enough to normalize the data through standardization in order to fit the model.

In the following charts, we can see the accuracy of the multi-class classifier and of the binary classifiers with different sizes of the train-validation split and with a different number of folds for the K-fold cross-validation.

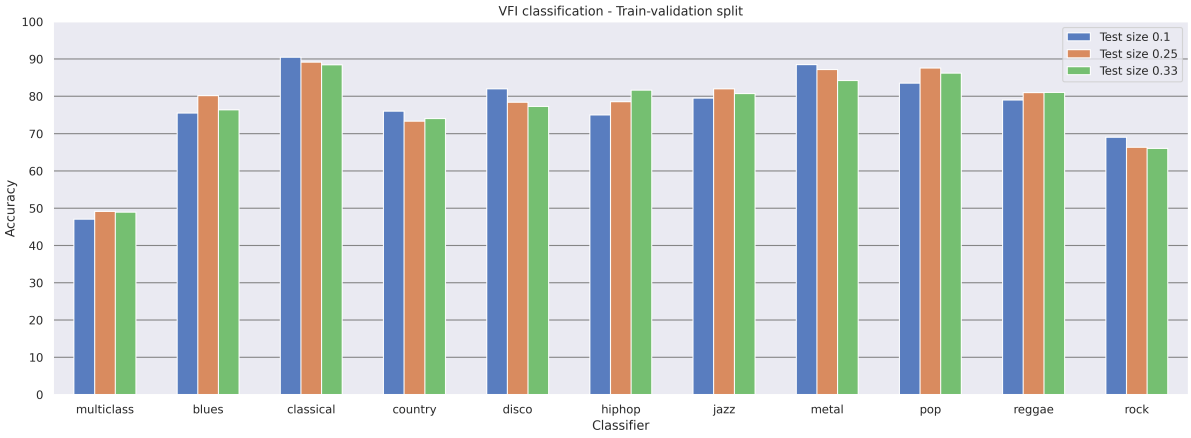

Figure 2.4: Accuracy of the VFI classifier with the train-validation split.

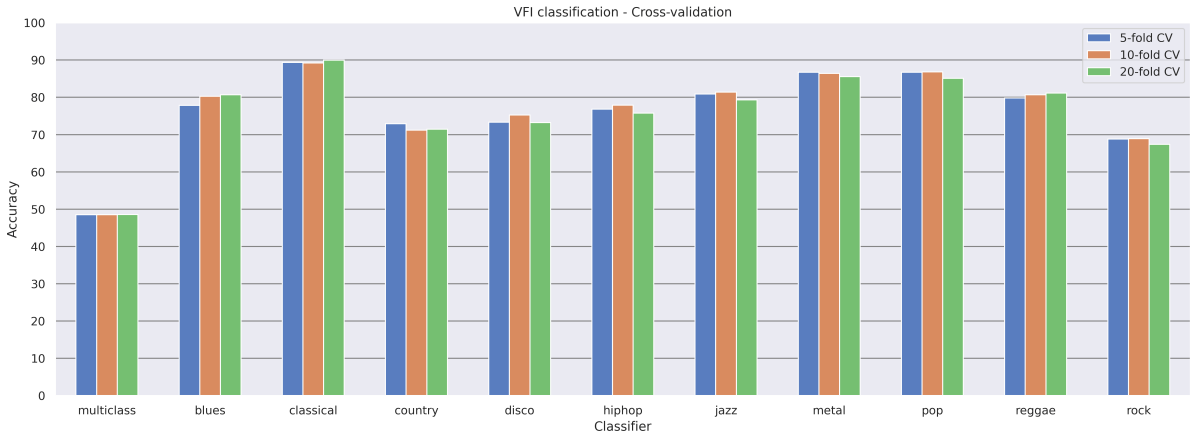

Figure 2.5: Accuracy of the VFI classifier with K-fold cross-validation.

VFI is a very similar model w.r.t. Naive Bayes, therefore we expected a similar performance. This is confirmed by our tests since we can easily see that the performance of VFI is very bad in the case of the multi-class classification problem and it is very good in the case of a simpler problem like classical music classification.

The following is the confusion matrix of the multi-class VFI classifier.

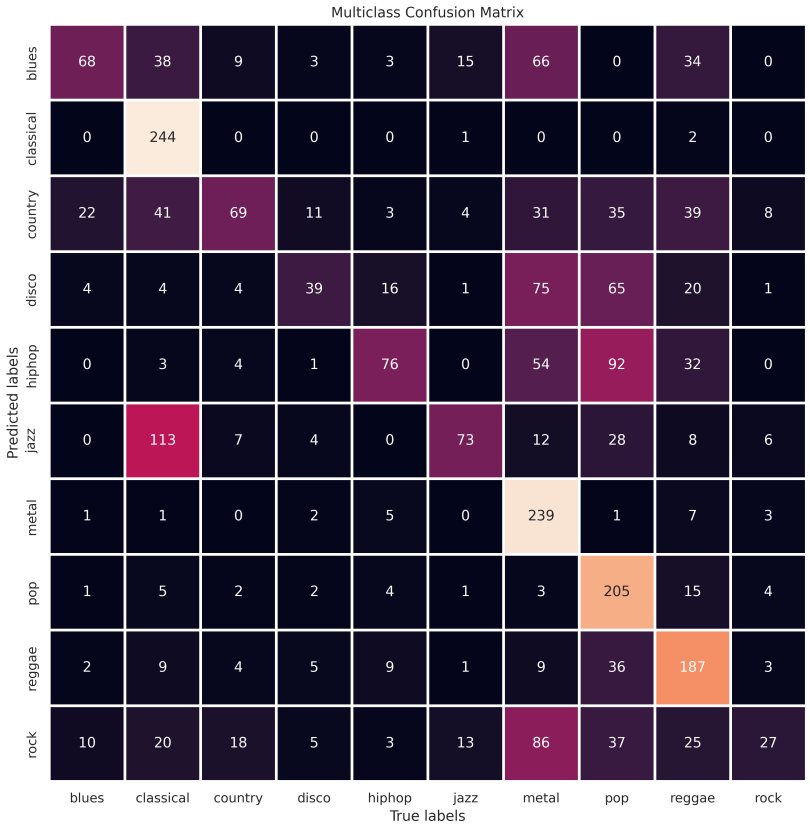

Figure 2.6: Confusion matrix of the multi-class VFI classifier.

2.3. CART Decision Trees

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. DTs are able to predict the value of a target variable by learning simple decision rules inferred from the data features. DTs are also simple to understand and interpret.

In our analysis, we used an implementation of decision trees known as CART (Classification and Regression Trees). CART constructs binary trees using the feature and the threshold that yield the largest information gain at each node. The code is provided by the library *scikit-learn* [3]. Every step of this analysis is reported in the notebook *CART_DT.ipynb*.

Decision trees can work with a categorical target variable and do not require normalizing the data, therefore with this model, the pre-processing required is minimal.

In the following charts, we can see the accuracy of the multi-class classifier and of the binary classifiers with different sizes of the train-validation split and with a different number of folds for the K-fold cross-validation.

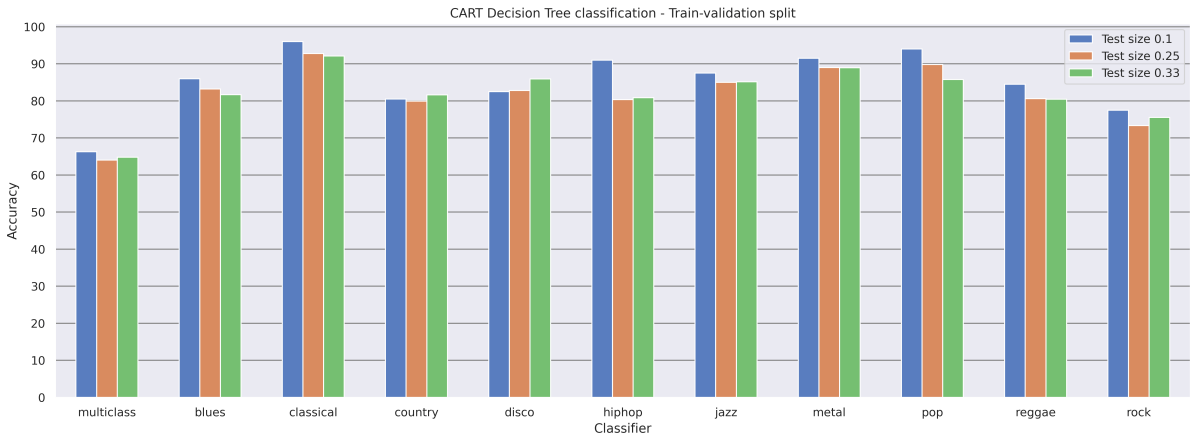

Figure 2.7: Accuracy of the CART DT classifier with the train-validation split.

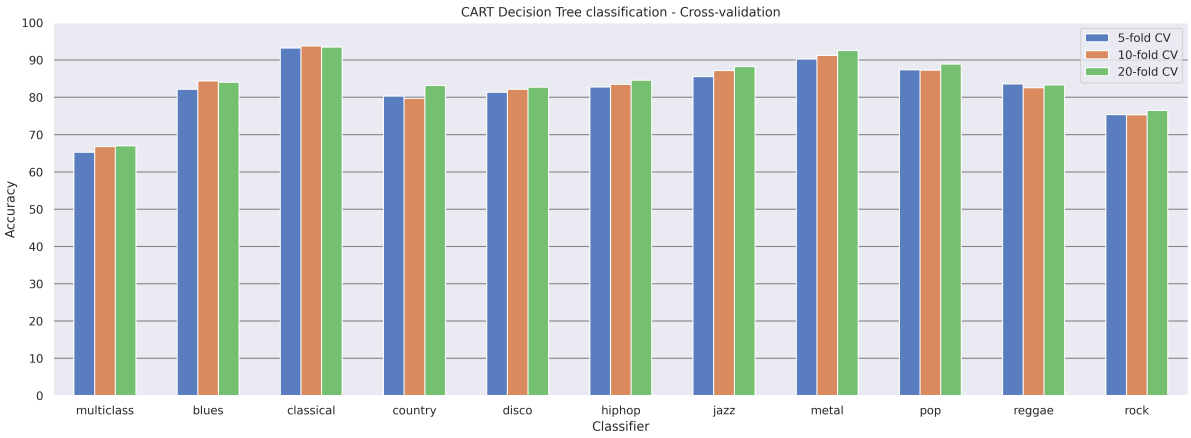

Figure 2.8: Accuracy of the CART DT classifier with K-fold cross-validation.

We are starting to see better overall accuracy. In fact, decision trees are known to offer good performance despite their short training phase. Furthermore, DTs are non-parametric models, meaning that they do not make any assumption about the distribution of the data and therefore are able to capture complex relationships between the variables.

The following is the confusion matrix of the multi-class CART DT classifier.

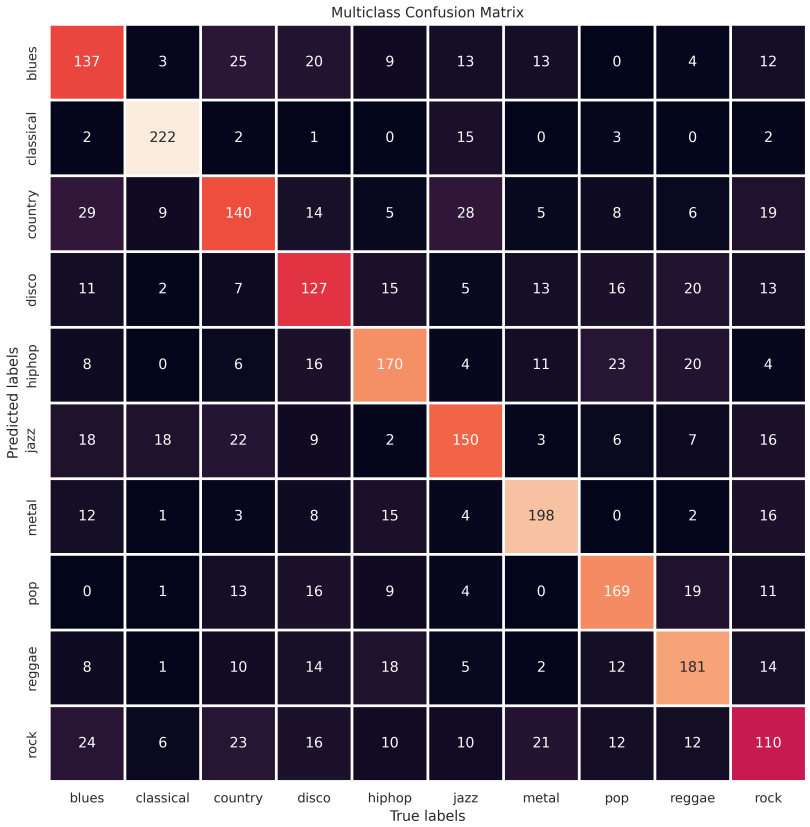

Figure 2.9: Confusion matrix of the multi-class CART DT classifier.

2.4. K-Nearest Neighbors

Neighbors-based classification is a type of instance-based learning, meaning that it does not attempt to construct a general internal model, but simply stores instances of the training data. Classification is computed through a simple majority vote of the nearest neighbors of each point.

In our analysis, we used the K-Nearest Neighbors (K-NN) classifier, which implements learning based on the nearest neighbors of each query point. The code is provided by the library *scikit-learn* [3]. Every step of this analysis is reported in the notebook *KNN.ipynb*.

It is known that the optimal choice of the value of k is highly data-dependent: in general, a larger k suppresses the effects of noise but makes the classification boundaries less distinct.

On the other hand, there are two possible choices for the weights:

- *uniform*: assigns uniform weights to each neighbor.
- *distance*: assigns weights proportional to the inverse of the distance from the query point.

Then, the value assigned to a query point is computed from a simple majority vote of the nearest neighbors.

This means that this time the model we are working with has some hyper-parameters. In order to find the best-performing model in our scenario, we will try all the combinations of the following two hyper-parameters. Note that we can afford this brute force approach since there are just two hyper-parameters that can only take a small set of values and since the time required to train each model is low.

K-NN can work with a categorical target variable but performs better when the data is normalized, therefore also in this case we will normalize the data set through standardization.

We have trained a different model for each combination of two hyper-parameters and compared them with one another using 20-fold cross-validation. The best model is the one with the highest average accuracy.

In the following chart, we can see the performance of all the models trained to solve the multi-class classification problem. We did not attempt to train models with a number of neighbors higher than 20 since we noticed that in this specific case adding more neighbors was making the predictions worse.

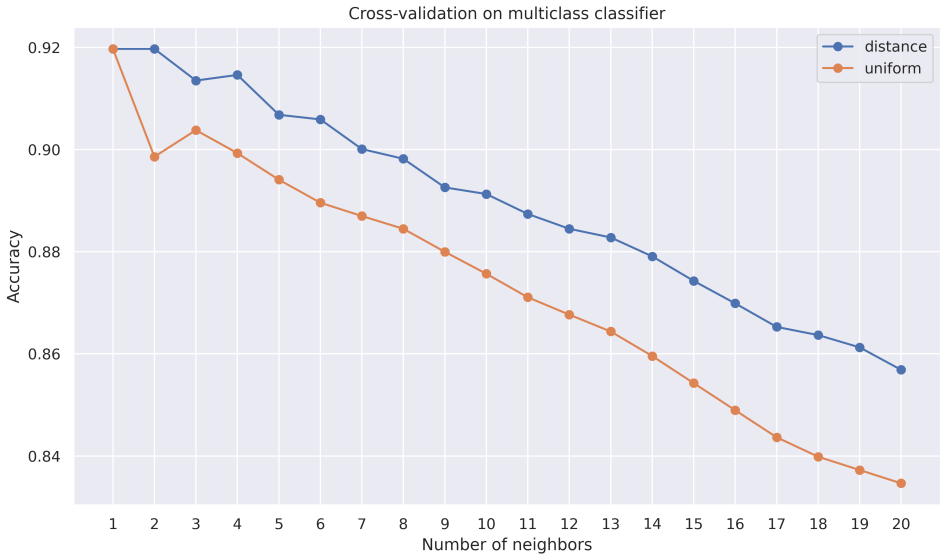

Figure 2.10: Model selection results of K-NN multi-class.

In the following chart, we can see the performance of all the models trained to solve the binary classification problems. In this case, we limited the number of neighbors to 15 since adding more neighbors was not improving the performance.

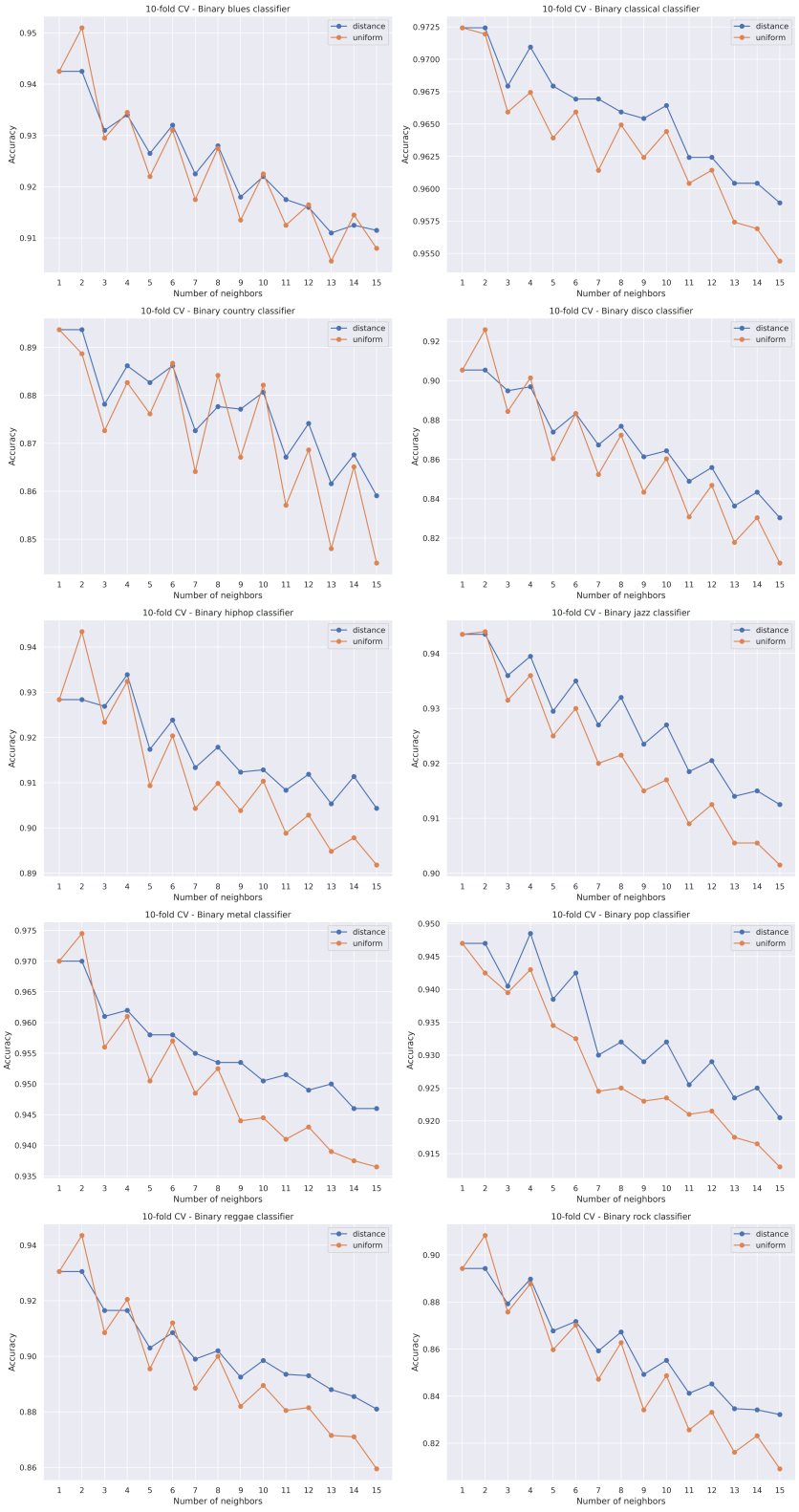

Figure 2.11: Model selection results of K-NN binary.

The following table reports the best hyper-parameters found for each one of the multi-class

problem and the 10 binary problems:

Classification problem	Number of neighbors	Weights
multi-class	1	distance
blues	2	uniform
classical	1	distance
country	1	uniform
disco	2	uniform
hip-hop	2	uniform
jazz	2	uniform
metal	2	uniform
pop	4	distance
reggae	2	uniform
rock	2	uniform

In the following charts, we can see the accuracy of the multi-class classifier and of the binary classifiers with different sizes of the train-validation split and with a different number of folds for the K-fold cross-validation. The models we are using are the ones with the hyper-parameters selected in the previous step.

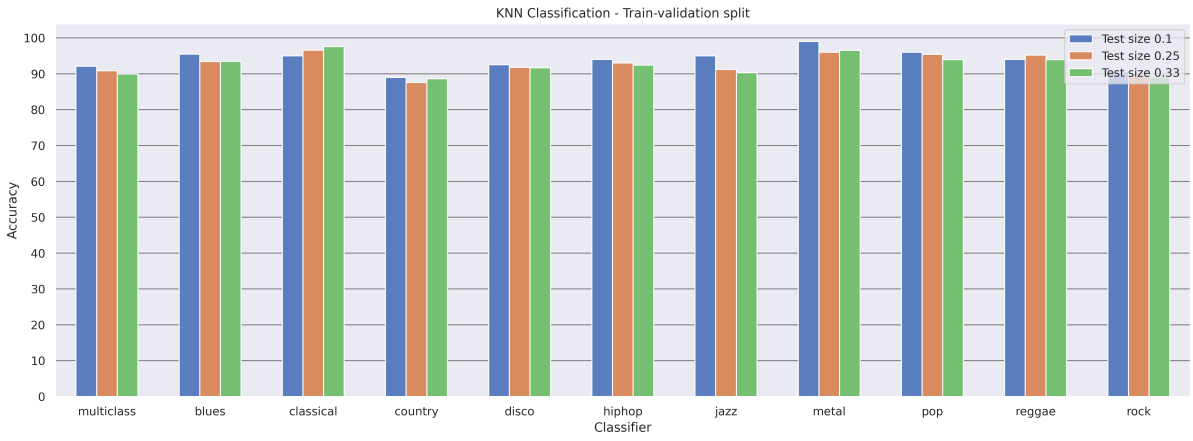

Figure 2.12: Accuracy of the K-NN classifier with the train-validation split.

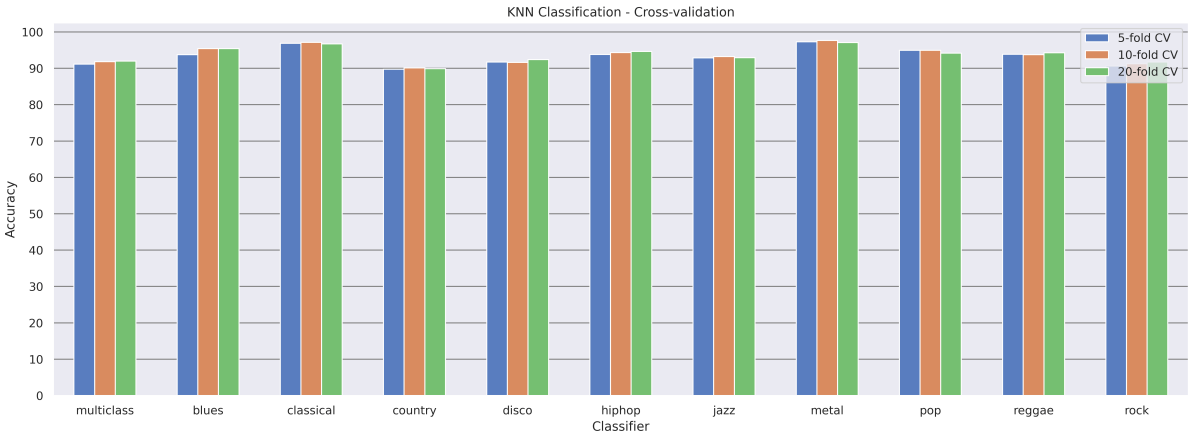

Figure 2.13: Accuracy of the K-NN classifier with K-fold cross-validation.

K-NN was able to reach a very high accuracy. In fact, K-NN is known to be a very powerful model, given that it is also non-parametric and that it makes no assumptions about the data distribution, meaning that is able to capture even very complex relations inside of the data set. It is also known to perform very well when the number of samples is large w.r.t. the number of classes. Its main drawback is that it requires keeping in memory the whole data set in order to compute predictions but in our case, since the data set we are working with is of very small size, this is not an issue.

The following is the confusion matrix of the multi-class K-NN classifier.

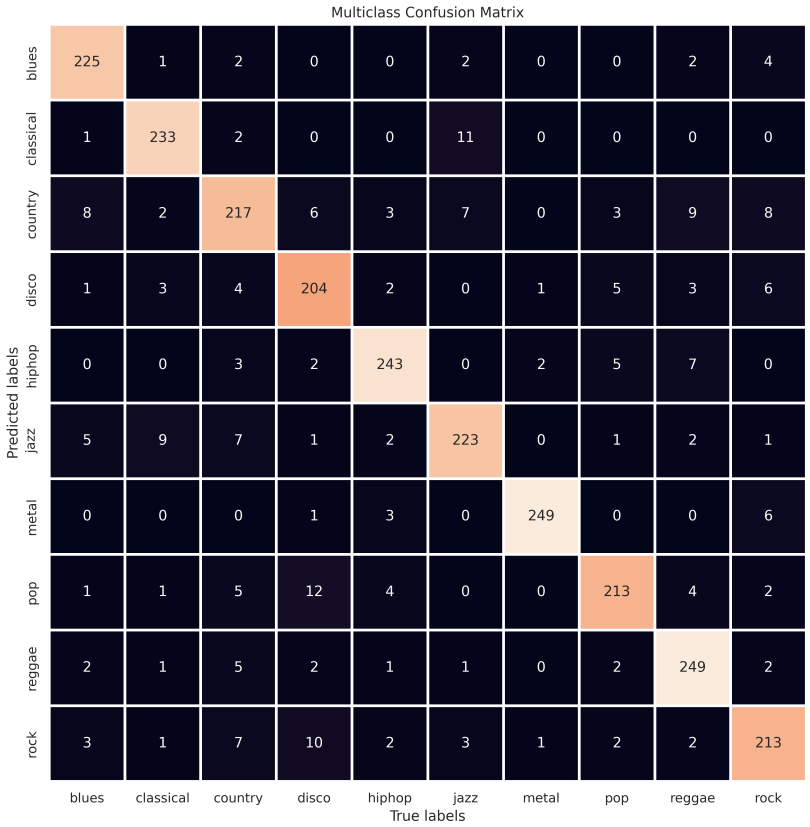

Figure 2.14: Confusion matrix of the multi-class K-NN classifier.

3 | Deep Learning models

In the last part of our analysis, we will switch from Machine Learning to Deep Learning, exploring a more modern model known as feed-forward fully-connected Neural Networks (FFNNs). Neural Networks are a very powerful instrument and the Universal Approximation Theorem provides a theoretical foundation for their applicability over any problem.

Fully-connected neural networks are networks in which all the neurons of all layers are connected to all the neurons of the following layer.

For simplicity, in our analysis, we will consider only networks in which all hidden layers have the same amount of neurons. This will let us reduce the number of hyper-parameters that we have to tune, without decreasing the approximation power of the model.

We will need to build and train a different neural network for each one of the classification problems we are facing.

3.1. Training

This section describes the content of the notebook *NN_selection.ipynb*. The library used to build, train and evaluate the neural networks is *keras* [4].

When training a neural network, there is a much larger number of hyper-parameters to be considered w.r.t. ML models, and the time required for the training phase is also much higher. In order to find the best model for each one of our problems, we cannot just blindly train a different network for every possible combination of the hyper-parameters: the extremely high number of combinations and the time required to train just one network make this unfeasible.

Therefore, we will focus on a very small subset of the hyper-parameter space, whose boundaries have been tuned empirically by training pseudo-randomly some hyper-parameter combinations and by checking the performance.

In the multi-class problem, a softmax layer was added after the last layer of the network in order to convert the outputs of the network into probabilities. This network outputs one

probability for each class, therefore it has 10 outputs. In the case of the binary problem, the last layer has only one output and uses the sigmoid function as an activation function, in order to map the output in the interval $[0, 1]$.

The following ranges were checked:

Hyper-parameter	Start (included)	Stop (included)	Step
Number of hidden layers	1	5	1
Number of neurons per layer	15	100	5
Mini-batch size	5	50	5

Again, given the large amount of time that the training phase requires, when comparing the performance of two different networks we will not resort to K-fold cross-validation but to a simple train-validation split of the data set, in particular with 80% of the data allocated for training and the remaining 20% for testing.

In order to train the networks, the Adam optimizer was used. In the multi-class problem, the loss function used is the categorical cross-entropy, whereas in the binary problem the binary cross-entropy. In both cases, the reference metric was the accuracy of the model on the validation set.

This time, other than normalizing the data through standardization, an additional pre-processing step is required: NNs require numerical variables, therefore we will need to convert the *label* from a categorical variable to a numerical variable. In the multi-class case, we need to created a one-hot encoding of such variable, that represents every single label as a vector of 0 and 1, where all elements are equal to 0, except for the one that represents the correct label. In the binary case, it was enough to encode as 1 the label of the positive class and as 0 the label of all the other classes.

Furthermore, we decided to store the results of the evaluation of each possible model in a persistent database, in order to avoid repeating the same training twice and in order to easily filter and aggregate the results with the goal of selecting the best model. The database of choice was MongoDB, given the extreme flexibility that it allows in storing the data. The connection with the instance of the database was handled with the Python library *pymongo*.

At the beginning of the training phase, we query the database in order to check if a network with the current combination of hyper-parameters had already been trained. If yes, the training phase is aborted and the loop restarts with a different combination

of hyper-parameters. Otherwise, the training phase is performed and at the end, the accuracy of the trained model, together with the combination of its hyper-parameters, is stored in the database.

The following table reports the hyper-parameters of the models selected at the end of this phase.

Classification problem	Number of hidden layers	Neurons per layer	Batch size
multi-class	2	95	5
blues	1	75	5
classical	1	60	15
country	1	80	20
disco	2	65	5
hip-hop	1	25	15
jazz	2	55	30
metal	2	70	15
pop	2	60	50
reggae	1	90	25
rock	1	75	20

With every network, the activation function used was the SELU (Scaled Exponential Linear Unit) which is an improved version of the very well-known ReLU (Rectified Linear Unit). Different activation functions were considered in this phase, mainly the hyperbolic tangent, the ReLU, the ELU (Exponential Linear Unit) and the SELU. The latter was found to be the best-performing one and therefore was kept.

In the last section of the *NN_selection.ipynb* notebook, we are retrieving the best combinations of hyper-parameters selected from the database through the id associated with their entry, in order to save those models using the *model.save* method of *keras*, which stores the all the information required to transfer the model from one notebook to another in just a single file in the *h5* format.

In this way, we will be able to evaluate the performance of those models in another notebook, without the need of connecting to the database at every single step. All the models can be found in the *models* folder of the GitHub repository [5].

3.2. Evaluation

As anticipated, the evaluation of the selected models is carried out in a different notebook, which is *NN_classification.ipynb*. In this notebook, we are loading the models saved in *NN_selection.ipynb* through the use of the *load_model* function from *keras*. We want to train those models once again, but with the same methodologies (i.e. train-validation split and K-fold cross-validation) used for ML models, in order to allow for a fair comparison between them.

In the following charts, we can see the accuracy of the multi-class classifier and of the binary classifiers with different sizes of the train-validation split and with a different number of folds for the K-fold cross-validation. The models we are using are the ones with the hyper-parameters selected in the previous step.

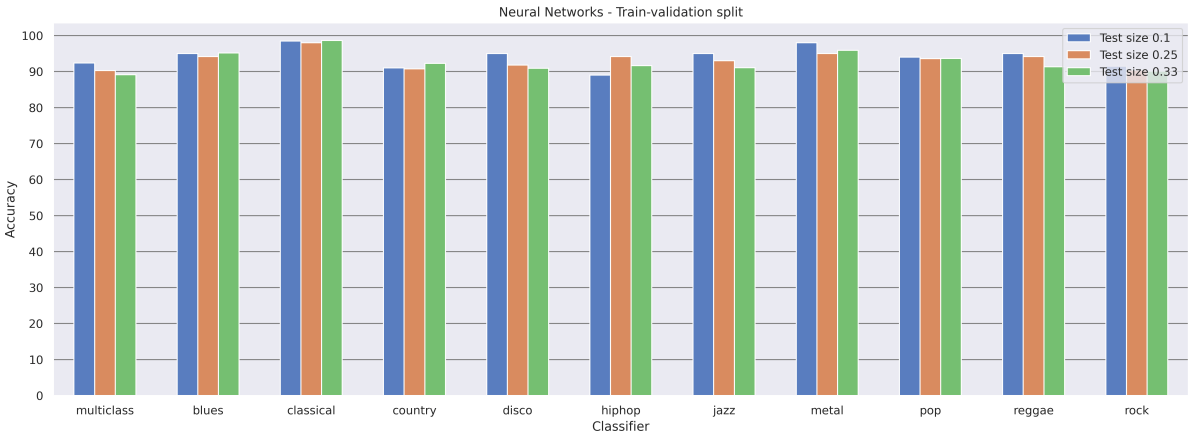

Figure 3.1: Accuracy of the NN classifier with the train-validation split.

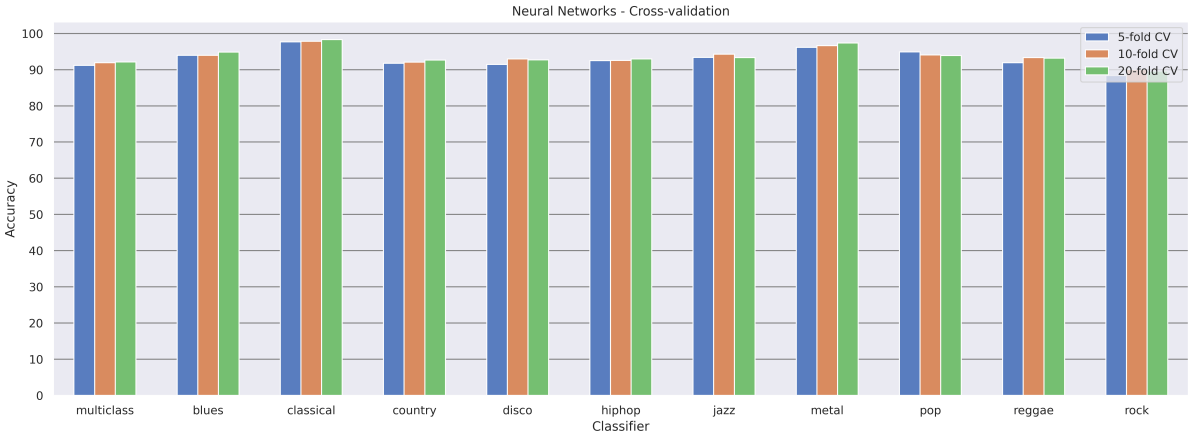

Figure 3.2: Accuracy of the NN classifier with K-fold cross-validation.

As expected, the selected NNs achieved very high accuracy in all the problems. In fact, NNs are capable of learning any complex patterns and relationships in the data. However, even if they offer very high performance, they suffer from many drawbacks, such as their long training phase, the very large amount of data they require and their lack of interpretability, being a Black-Box model.

The following is the confusion matrix of the multi-class NN classifier.

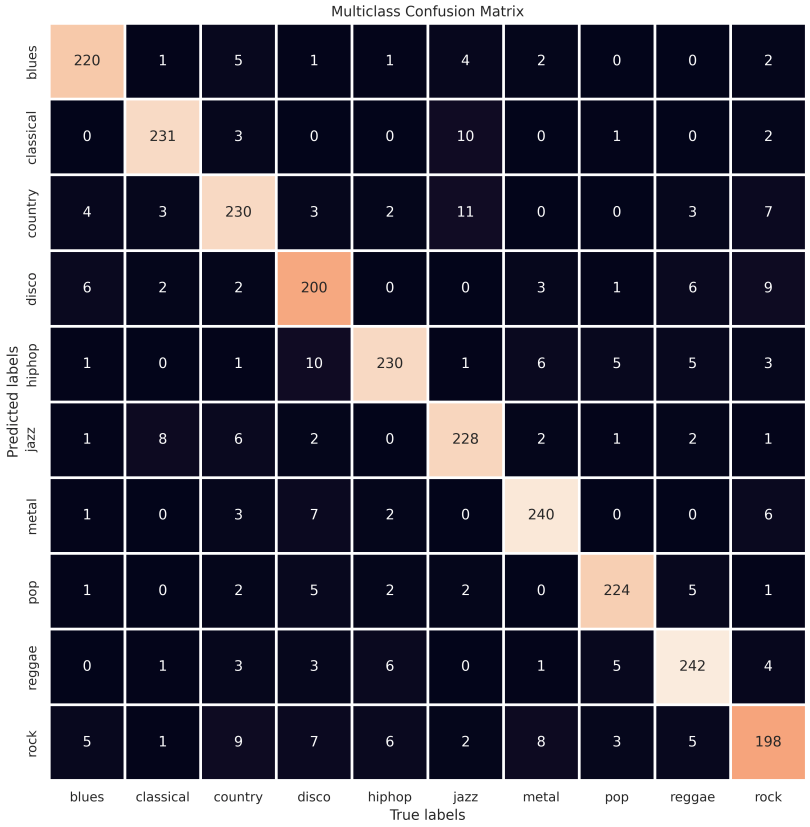

Figure 3.3: Confusion matrix of the multi-class NN classifier.

4 | Comparison

In the following chart, we can see a comparison of the accuracy of all the models considered in this project, both in the multi-class case and in the binary case. Those results were produced by evaluating all the models using 20-fold cross-validation.

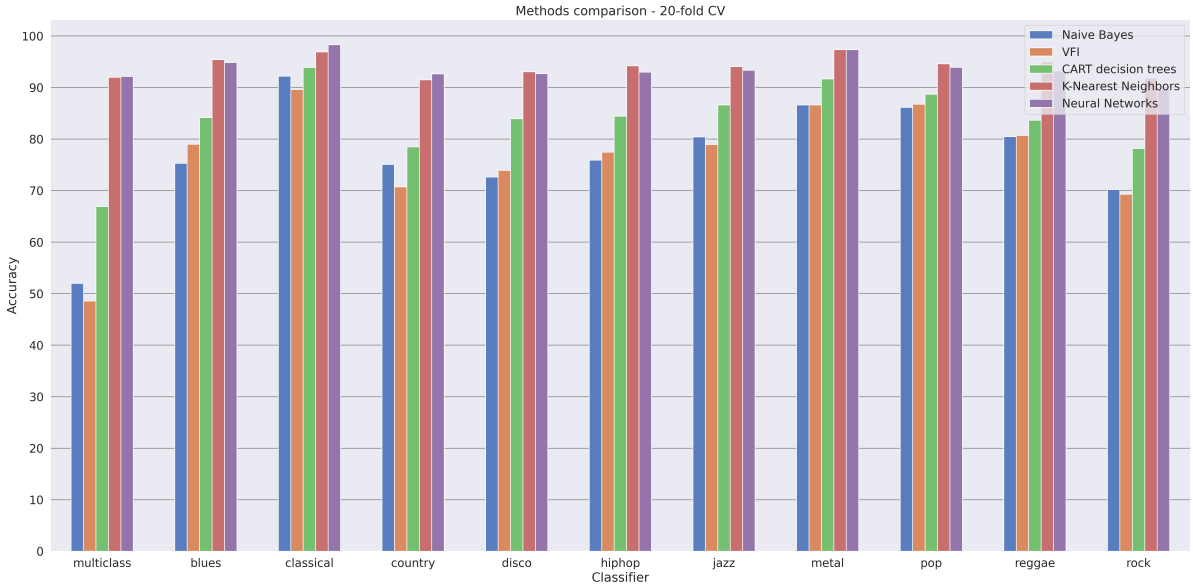

Figure 4.1: Comparison of the accuracy of the models considered in the project.

As expected, K-Nearest Neighbors (K-NN) and Neural Networks (NNs) are the most performing models. On the other hand, the performance of very simple models like Naive Bayes (NB) and Voting Feature Interval (VFI) is much worse, in particular in the more complex setting of the multi-class classification problem.

An interesting remark is that we expected more performance from NNs, given their theoretical properties. However, we have to cope with the fact that we are in a real-world setting, in which the data at our disposal is not unlimited. NNs and Deep Learning models, in general, are known to require much more data than Machine Learning models. Even if we were able to extend the number of samples in the data set from 1000 to 10000 by splitting each song into 10 tracks with a length of 3 seconds, we cannot say

that we have enough data to effectively train a Deep Learning model. It is possible that this is the reason why we are getting a high performance indeed but still not completely outperforming a simpler model like K-NN.

On the other hand, as stated in the original paper [1], binary classification models outperform multi-class classification models in terms of accuracy. The task of separating musical instances of a particular genre from all the others seems easier.

Furthermore, from an analysis of the performance in the binary problems, we can see that some musical genres are much harder to recognize w.r.t. other genres. For instance, we can see that telling whether a musical track resembles classical music or not is an easy task given that all models, even the least powerful ones, were able to achieve high accuracy. On the other hand, the rock genre seems to be hard to classify and we have to resort to more powerful models in order to get good results.

This is expected since some genres are characterized by an inherent mutual ambiguity, meaning that it is sometimes hard, even for human annotators, to tell the difference between them. This is the case of genres like blues and jazz. The paper *Browsing Music Spaces: Categories And The Musical Mind* [8] states that "Genres emerge as terms, nouns that define recurrences and similarities that members of a community make pertinent to identify musical events". Therefore, we can say that musical genres are in general hard to be systematically described and no complete agreement exists in their definition and assessment.

5 | Conclusions and future developments

The ambiguity inherent to every definition of musical genre, together with the high dynamics that undermine its persistence over time, characterizes the complexity of the automatic genre categorization task. Nonetheless, as stated in the original paper [1], the results we obtained are very encouraging and suggest that simple musical features can provide effective information for genre categorization. We can conclude that building systems for automatic genre categorization is certainly possible and can also bring a high level of accuracy.

The project can be developed further in two different directions:

- As suggested in the original paper, we can collect the binary classifiers into a hierarchical meta-learner, whose performance is expected to be better than the performance of the corresponding multi-class classifier.
- Starting from the audio tracks, we could extract their corresponding spectrogram, which is a very compact representation of the audio track in the form of an image that allows us to see how the frequencies vary over time. Given that the spectrogram is an image, we can process it using more modern technologies from Deep Learning, like Convolutional Neural Networks (CNNs). In that case, we would also be able to increase the samples at our disposal using a technique called data augmentation, which consists in fictitiously augmenting the data set by performing geometrical transformations on the images. Given those premises, we expect that CNNs will guarantee even more performance w.r.t. fully-connected NNs.

Bibliography

[1] Roberto Basili, Alfredo Serafini, and Armando Stellato. Classification of musical genre: a machine learning approach., 01 2004.

[2] Andrada Olteanu. Gtzan dataset. `https://www.kaggle.com/datasets/andradaolteanu/gtzan-dataset-music-genre-classification?resource=download-directory`, 2020.

[3] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[4] François Chollet et al. Keras. `https://keras.io`, 2015.

[5] Alberto Pirillo. Comparison of ml models for musical genre classification. `https://github.com/albertopirillo/naml-project-2023`, 2023.

[6] Florian Wetschoreck, Tobias Krabel, and Surya Krishnamurthy. 8080labs/ppscore: zenodo release. `https://doi.org/10.5281/zenodo.4091345`, October 2020.

[7] Christos Aridas. Voting feature intervals - python. `https://github.com/chkoar/vfi`, 2019.

[8] Franco Fabbri. Browsing music spaces: Categories and the musical mind. `http://www.mediamusicstudies.net/tagg/others//ffabbri9907.html`, 2007.