Report

Report

Data mining project

Student: Mollica Paolo Junior & Vichi Emanuele

Student code: 564222, 544291

Course: Data Mining

A.Y. 2022-2023

Abstract

In this report we are going to analyze the ravdess_features.csv dataset, for this purpose we'll use the tools learned during the first part of the course of Data Mining. The main tasks we will tackle are:

- **Data Understanding** & **Preparation**
- **Clustering**
- **Classification**
- **Pattern Mining**

We'll face this tasks using Python and the necessary packages, among which the most important are: pandas, numpy, scipy, sklearn.

Contents

List of Figures

1. Data Understanding & Preparation

The *ravdess_features.csv* is a dataset with 2452 rows and 38 columns. The dataset is in the form of a **Data matrix** and each data object represents a point in a 38 dimensional space. The types of the attributes values are the following:

- **float64**: 25 **object**: 7 **int64**: 6

1.1 Data Semantics

1.1.1 Object

For the object type we have 7 columns, which are: *modality, vocal_channel, emotion, emotional_intensity, statement, repetition, sex*. Looking at the info for this attributes we obtained:

Attribute	Values	Count
modality	Audio-only	2452
vocal_channel	speech	1335
	song	921
	NaN	196
emotion	fearful	376
	angry	376
	happy	376
	calm	376
	sad	376
	surprised	192
	disgust	192
	neutral	188
emotional_intensity	normal	1320
	strong	1132
statement	Dogs are sitting by the door	1226
	Kids are talking by the door	1226
repetition	2nd	1226
	1st	1226
sex	M	1248
	F	1204

Table 1: Object type attributes info.

These are all categorical attributes: all of them are nominal ones except for repetition and emotional_intensity which are ordinal. From Table:1 we saw that the *modality* feature has only one value for each record, so we decided to drop it from the dataset.
We also saw that the only attribute with missing values was *vocal_channel* so we'll come back later to dealing with them. Furthermore, we can observe that *emotional_intensity*, *statement*, *repetition* and *sex* are a special case of discrete attributes, they are **binary attributes**.

1.1.2 Float64

For the Float64 type we have 25 columns, belonging to two different classes:

- **Categorical**: actor, stft_max
- **Numerical**: frame_count, intensity, mfcc_mean, mfcc_std, mfcc_min, mfcc_max, sc_mean, sc_std, sc_min, sc_max, sc_kur, sc_skew, stft_mean, stft_std, stft_min, stft_kur, stft_skew, mean, std, min, max, kur, skew

The categorical attributes both have discrete values. The *actor* one goes from 0 to 23 and it has 1126 missing values. Because it provide only enough information to distinguish one object from another, we decided to drop it. Same for $stft\_max$, which has only one value and do not convey information at all.

All the numerical attributes have just continuous values. We observed that the only one that had missing values (816) was *intensity*. For further analysis of the other continuous attributes we remind to the section:1.2.

1.1.3 Int64

For the Int64 type we have 6 columns. In this case they all have discrete values, but we can distinguish categorical and numerical attributes:

- **Categorical**: channels, sample_width, frame_rate, frame_width
- **Numerical**: length_ms, zero_crossing_sum.

In this case none of the attributes had missing values. Anyway, we decided to drop the $frame\_rate$ attribute because it had just one value, and also we dropped the other categorical attributes because they had just 6 values that were different from the others (maybe linked to the different channel).

1.2 Distribution of the variables and statistics

1.2.1 Categorical attributes

For the Objects categorical attributes we have already given a first look to their distribution in Table:1.
For the Float64 categorical values the distribution is irrelevant because the *actor* attribute can be treated as an ID, so its distribution is meaningless, while the $stft\_max$ has only one value. In the int64 case we have $frame\_rate$ and $sample\_width$ that have just one value, while for the other two we obtain table:2.

Attribute	Values	Count
channels	1	2446
	2	6
frame_width	2	2446
	4	6

Table 2: Channels and frame_width distribution.

1.2.2 Numerical attributes

For the numerical attributes we have computed the *mean*, *standard deviation*, *min* and *max* that are shown in Table:3. Then we have plotted the histogram of these attributes plotting over them the probability distribution that better fit the data (fig:1,2).
The majority of the attributes have a normal distribution, while others like *min*, *std*, *max* ecc. seem to have an exponential distribution. For some others we have noticed that is difficult to check their distribution using histograms, due to wrong values or outlier like in the sc_min attribute. Finally, we have noticed that *mean* follows a Laplace distribution.

Attribute	Mean	Std dev	Min	Max
length_ms	4092	598	2936	6373
zero_crossing_sum	12885	3665	4721	30153
frame_count	193587	36825	-1	305906
intensity	-37.6	8.4	-63.8	-16.4
mfcc_mean	-28.8	4.5	-43.8	-15.5
mfcc_std	136.8	20.4	83.6	195.9
mfcc_min	-758.9	99.9	-1085	-461.5
mfcc_max	199.2	26.0	126.3	280.2
sc_mean	5170	875	2361	7655
sc_std	3365	580	1491	4819
sc_min	551.8	508.0	0.0	2121.4
sc_max	11830	1005	7657	17477
sc_kur	-1.143	0.573	-1.796	3.658
sc_skew	0.348	0.353	-0.510	1.825
stft_mean	0.476	0.083	0.214	0.724
stft_std	0.331	0.024	0.210	0.392
stft_min	0.002	0.005	0.0	0.039
stft_kur	-1.248	0.212	-1.670	0.795
stft_skew	0.113	0.331	-0.994	1.466
mean	1.39e-08	4.26e-05	-0.00094	0.00122
std	0.021	0.021	0.001	0.152
min	-0.165	0.175	-0.999	-0.006
max	0.180	0.196	0.005	0.999
kur	11.203	6.615	1.758	59.086
skew	-0.048	0.455	-2.357	1.800

Table 3: Numerical attributes stats.

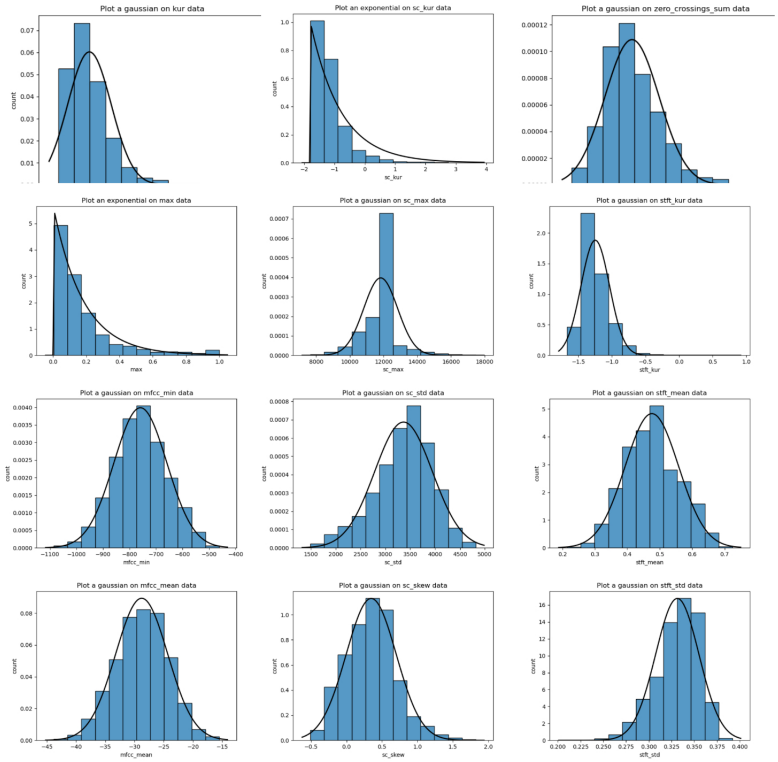

Figure 1: Hist with pdf

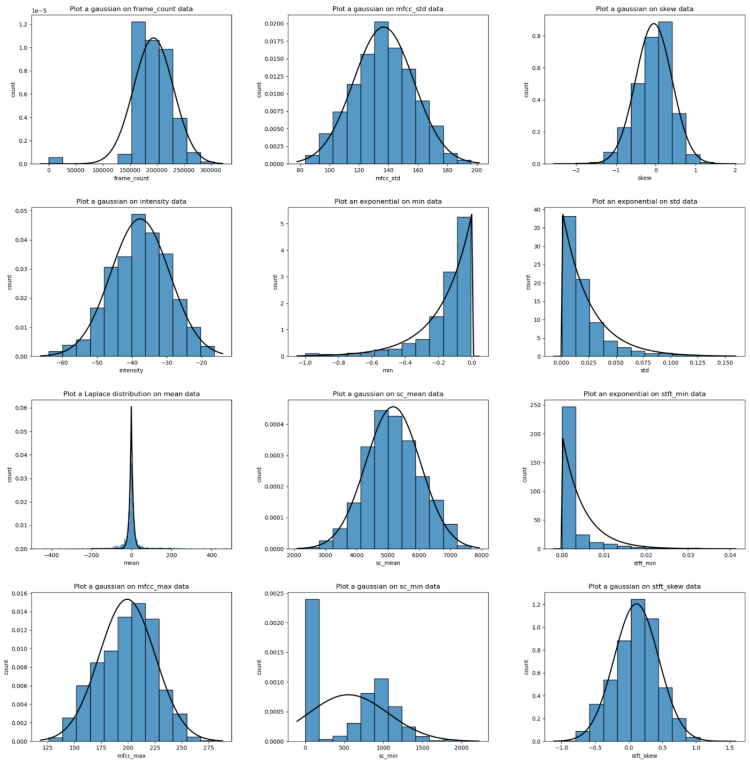

Figure 2: Hist with pdf

1.3 Assessing data quality

1.3.1 Missing Values

For a data matrix is important to have a complete vector of attributes for all the objects, so we need to check if there are missing values in our dataset.
In the previous sections we had seen that there are 3 attributes with missing values: *vocal_channel*, *actor* and *intensity*. For the *actor* one we already decided to discard it, while for the *intensity* missing values we'll face them using regression in section:5. So in this section we will look at the *vocal_channel* (categorical) attribute, which had 196 missing values.
Firstly, we assigned to *speech* those missing values that belonged to objects having *surpised* or *disgust* emotions because for this two, as stated in the brief description of the dataset and as we can see in Fig:3, there was no *song* values. Then, we started to see how the other attributes were distributed with respect to *song* and *speech*, which are the two possible values of *vocal_channel*. So we selected those attributes which had the most different distribution with respect to the two values, as shown in Fig:3.
The next step was, for each of these attributes, selecting the 0.01th quantile and the 0.99th quantile of the two distributions, respectively:

- $So_{0.01}$: 0.01th quantile of the distribution respect to *song*
- $Sp_{0.01}$: 0.01th quantile of the distribution respect to *speech*
- $So_{0.99}$: 0.99th quantile of the distribution respect to *song*
- $Sp_{0.99}$: 0.99th quantile of the distribution respect to *speech*

So we applied the algorithm:1 and we repeated this process for each attribute selected before, but it wasn't enough to fill all the missing values, so we applied it again to the *length_ms* attribute, which was the attribute with more different distribution respect to the *vocal_channel*'s values, but this time filtering it by emotion (using just the six emotions in common between song and speech). At the end of the process were still remaining 5 missing values, and we choose to assign them to *speech* because it was the mode of the attribute.

Algorithm 1 Replace Nan Algorithm

Input: Attributes distribution respect to *vocal_channel*
Output: Assigned *song* or *speech* to Nan values of *vocal_channel*

Take the 0.01th and 0.99th quantile for each distribution:$So_{0.01}$, $So_{0.99}$, $Sp_{0.01}$ and $Sp_{0.99}$
if $mean_{song} > mean_{speech}$ **then**
  $target.attr\_value < So_{0.01}$: $target.missing\_value = speech$
  $target.attr\_value > Sp_{0.99}$: $target.missing\_value = song$
end if
if $mean_{song} < mean_{speech}$ **then**
  $target.attr\_value > So_{0.99}$: $target.missing\_value = speech$
  $target.attr\_value < Sp_{0.01}$: $target.missing\_value = song$
end if

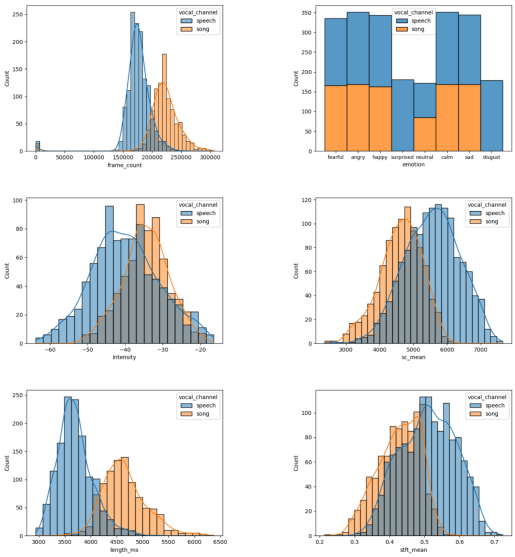

Figure 3: Histogram for missing values

1.3.2 Zero Values

Observing the distribution in fig:1 we noticed that *sc_min* had a peak near zero, which is distant from the rest of the distribution, and also it had 1021 null values. After further investigation we noticed that this 1021 values correspond to 1021 null values in the $stft\_min$ attribute too. Since they are continuous values it is difficult to proceed like in the *vocal_channel* case, so we decided, like we did for *intensity*, to deal this problem in section:5.

1.3.3 Outliers

For the outliers detection, at the beginning, we observed the distributions in fig:1,2. At first glance we can distinguish attributes that have a **non-well defined** distribution, attributes with a **Gaussian** distribution and attributes with **Exponential** distribution, respectively:

Non-well defined : $sc\_max, kur, frame_count, sc\_min, sc\_max, stft\_kur$

Gaussian : $intensity, zero\_crossings_sum, mfcc\_mean, mfcc\_std,$
$mfcc\_min, mfcc\_max, sc\_mean, sc\_std, sc\_skew, stft\_mean, stft\_skew, stft\_std, skew$

Exponential : std, min, max, sc_kur

Laplacian : $mean$

In order to obtain a better detection, we ensured that there was no zero in the exponential distributed attributes, we changed the sign for the *min* values, and we applied a log-transformation to them. So, we added a new column for each log-transformed attribute and we saw that their distribution now followed a **Gaussian** one.
After setting up the exponential distribution, we decide to use different strategies, one for the Gaussian distributed attributes and one for the remaining ones.
In the Gaussian case we decided to adopt the $3-\sigma$ **rule**, while for the others we decided to use the 0.0015th and 0.9985th quantile (that roughly correspond to the 99.73% of the $3-\sigma$). So, we added a new column corresponding to the count of the outliers in each data object, detected applying the rules above. Then, we decided to eliminate raws with 2 or more outliers: in this way we discarded 31 rows from the dataset, thus getting a dataset with 2421 objects.

1.4 Variable transformations

In the previous section we already used variable transformations like log-transform, but at the end we decided to keep the original values (exponentially distributed) because such a transformation would have changed the nature of the data and we didn't know how we were going to use them yet. Eventually we will reason for each case if it's better using the log-transformed attributes or not.
For the categorical attributes, instead, we decided to map them to (discrete) numerical values. We transformed *vocal_channel*, *emotional_intensity*, *sex*, *repetition* and *statement* in binary attributes with values 0 and 1, while for the *emotion* feature we mapped his values simply from 0 to 7.

1.5 Pairwise correlations and eventual elimination of variables

In order to deal with the correlation, we performed a cycle to calculate the pairwise correlation for the numerical attributes. In that cycle, we took the absolute value of the correlation and compared it to a threshold value of 0.95. Doing so, we obtained the following lists of correlated attributes pairs:

- $[min, std], [max, std], [max, min]$
- $[intensity, mfcc\_std], [intensity, mfcc\_min], [mfcc\_min, mfcc\_std]$
- $[stft\_mean, stft\_skew]$

Then for this list of high correlated attributes we decide to maintain only one attribute for each pair. In order to do a reasonable choice, we plotted a box-plot for each attribute and we counted its outliers. Then, for each list, we choose to leave the attribute with the lower number of outliers.
At the end, the attributes we decided to keep are *std*, $mfcc\_std$ and $stft\_mean$. For the *intensity* attribute, as said previously, we decided to not discard it in order to perform a further analysis for the missing values in the regression section, like for *sc_min* and $stft\_min$, so this attributes will not be used in the following analysis. At the end of the preprocessing phase we obtained a dataset with 2421 rows and 27 columns.

2. Clustering

In this section we are going to analyze three different types of clustering: **centroid-based clustering**, **density-based clustering** and **hierarchical clustering**. The main difference between the three methods is that the first two produce **unnested** clusters, while the hierarchical one produces **nested** clusters. An unnested cluster is simply a division of the data objects into **non-overlapping** clusters, while the hierarchical method generate nested clusters organized as a tree, in a way that each cluster (node) in the tree contains its sub-clusters (children).

2.1 Analysis by centroid-based methods

For centroid-based methods we will consider two types of clustering: K-means and Bisecting K-means. For our analysis we will try to understand how much this type of clustering will help us in the detection of the categorical values of *emotion* and *sex*.
Since K-means is based on prototypes in terms of **centroid** and it is typically applied to objects in a

continuous **n-dimensional** space, where in this case n is the number of attributes taken into account, we used polar graphs to select the most meaningful attributes for the values detection, reported in fig:4.
We created these graphs by taking the numerical attributes, applying them a MinMax scaler because of the different scale of the attributes, and then taking the mean for each attributes. So, looking at them, we choose to maintain only these attributes:

for *emotion* : $mfcc\_std$, sc_kur, $stft\_kur$, $mean$, std, kur

for *sex* : $mfcc\_max$, $stft\_std$, $stft\_mean$, $mfcc\_mean$

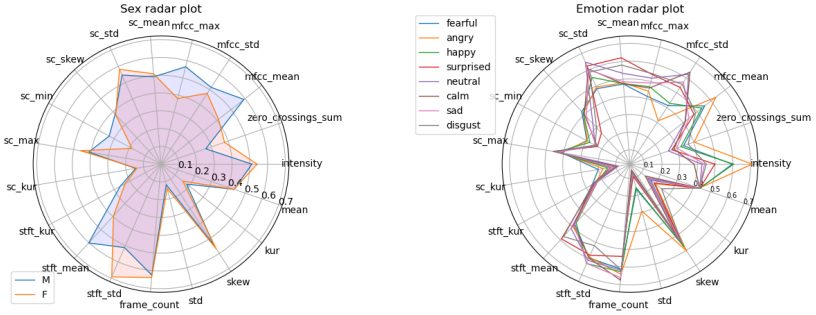

Figure 4: Radar plot for *emotion* and *sex* attributes

2.1.1 K-means

K-means is by definition a prototype-base, partitional clustering technique that attempts to find a user-specified number of clusters (**k**). So, in this case, we wanted to use clustering to see if categorical attributes, such as *emotion* and *sex*, spontaneously cluster together with respect to their values. Considering so, although we did an analysis with the aim of finding the best k to use, at the end we used predefined k give by the number of different values that the two attributes can assume, respectively $k = 8$ and $k = 2$.
For the clustering evaluation, considering the nature of the dataset, we used the **SSE** and the **Silhouette coefficient**. The results are reported in table:4.

	SSE	**Silhouette**
emotion	71	0.24
sex	163	0.33

Table 4: K-means evaluation

2.1.2 Bisecting k-means

We decided to perform the Bisecting k-means algorithm too, because it has less trouble with initialization as it performs several trial bisection and takes the one with lowest SSE. However we must be careful because the final set of clusters doesn't represent a local minimum for the total SSE, due to local nature of the algorithm.
Using the same setting as K-means we obtained the results reported in table:5.

	SSE	**Silhouette**
emotion	79	0.19
sex	163	0.34

Table 5: Bisecting K-means evaluation

2.1.3 Discussion

From Table:4,5 we can see that the results of the different algorithms are analogous for both the attributes. The SSE give us information about the distance from objects to the cluster centroid, instead Silhouette tell us if an objects is closer to other objects in its cluster or to objects in other clusters.
In fig:6 we observe that the nature of SSE give us a globular clusters, while if we look at the scatter plot of the *emotion* attribute we can notice that there is no clustering among its values. For the *sex* attribute, looking at the silhouette value we can say that both the algorithms produced purer clusters. We can also notice that they perform better on the *sex* attribute observing at fig:5, in which we can see a lower tendency to form globular clusters. Indeed, it seems to be similar to the scatter plot of the true values.

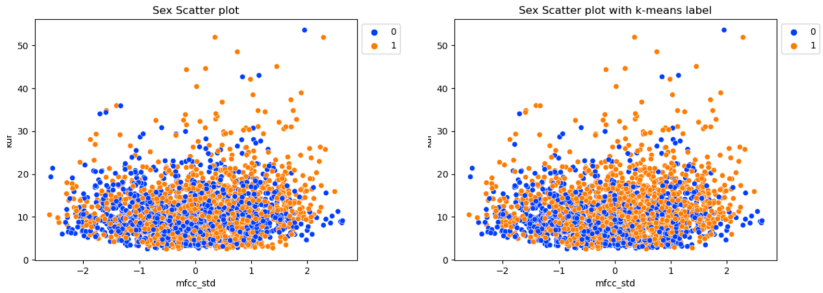

Figure 5: Scatter plot for *sex* .

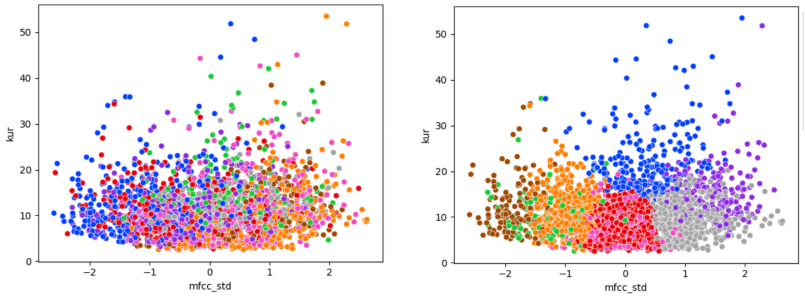

Figure 6: Scatter plot for *emotion* with actual and k-means labels.

2.2 Analysis by density-based clustering

Density-based clustering detects regions of high density that are separated from one another by regions of low density. We decided to perform the same analysis on the *emotion* and *sex* attributes as in section:2.1, so we used the same feature selection.

2.2.1 DBSCAN

The setting of DBSCAN requires a minimum threshold of the number of points within a given neighborhood of a point (**MinPts**), and a user-specified distance to define the neighborhood (**Eps**).
In order to find the best values for these hyperparameters, we performed an analysis with respect to the silhouette coefficient for different MinPts, and we obtained an optimal results for **MinPts** = 22. Instead, for Eps we plotted a k-dist graph (fig:7), where k-dist is the distance from a point to its k^{th} nearest neighbor. This led us to set **Eps** = 0.95. The results are reported in table:6 and the resulting clusters are reported in fig:8.

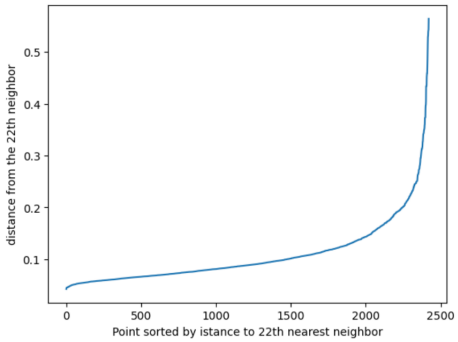

Figure 7: K-dist graph

	Silhouette
emotion	0.33
sex	0.13

Table 6: DBSCAN evaluation

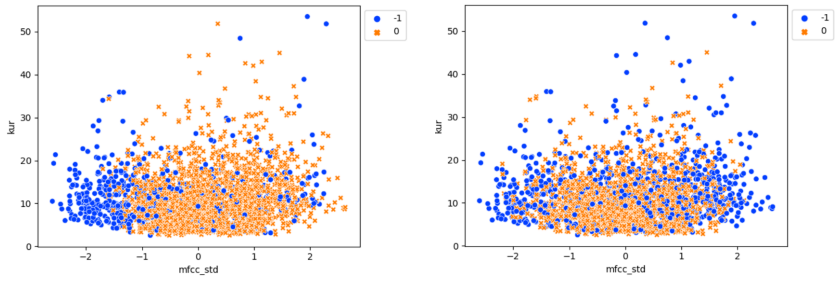

Figure 8: Scatter plot with and DBSCAN labels(*emotion*:left, *sex*:right)

2.2.2 Discussion

DBSCAN uses a density-based definition of a cluster, this led to a resistance to noise and ability to deal with clusters of arbitrary shapes and sizes. However, in this case, this algorithm can't help us in our analysis, because the distributions of *emotion* and *sex* respect to the selected attributes cause them to overlap each other as shown in fig:9 using the violin plots. In this plots, we take the distributions of *emotion* and *sex* with respect to the attributes that seems to guarantee the major divergence between their values. So the DBSCAN, in both cases, detects a huge highly dense region and a sparser region around as shown in fig:8. This suggested us to not perform further analysis on this type of clustering algorithms, like **OPTICS**.

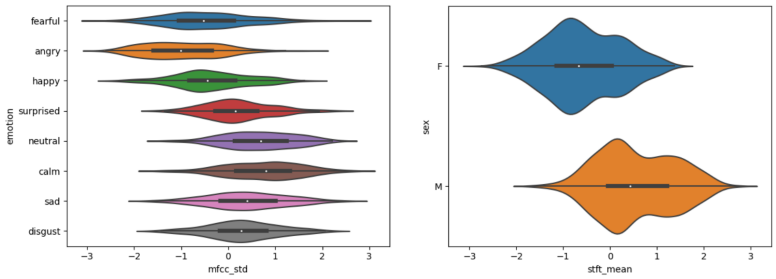

Figure 9: Violin plot for *emotion* and *sex*

2.3 Analysis by hierarchical clustering

In this section we'll perform an analysis by hierarchical clustering, in particular we'll use an **agglomerative** algorithm, so starting with each data objects as individual clusters and, at each step, merging the closest pair of clusters. Again, we performed the analysis on the *emotion* and *sex* attributes as in section:2.1, and so we used the same feature selection as before.
Since the computation of proximity is central in hierarchical algorithm, we performed the analysis with different distance functions. In particular, we used **complete** and **ward** linkage with the euclidean metric. The hierarchical can be compared in this to the Bisecting k-means, because they both have a lack of a global objective function and instead they use various criteria to decide their local behaviour.

2.3.1 Complete linkage

Here we used the complete linkage, which takes the maximum distances between all the observations into the two sets. The results are reported in table:7. The dendrograms and the scatter plots with the hierarchical label for the *emotion* and *sex* attributes are shown, respectively, in fig:10 and fig:11.

	Silhouette
emotion	0.43
sex	0.28

Table 7: Hierarchical with complete linkage evaluation

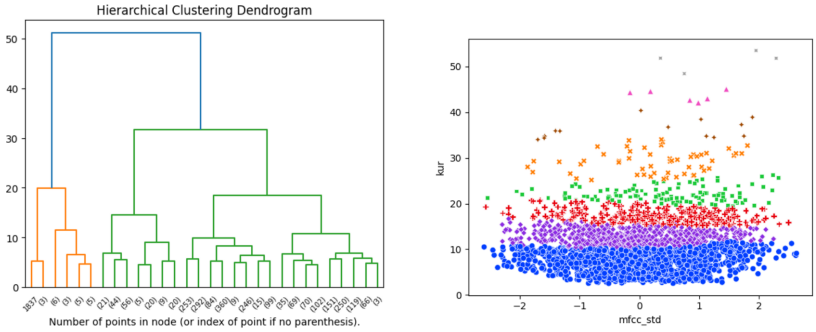

Figure 10: Dendrogram and scatter plot for *emotion* after agglomerative clustering

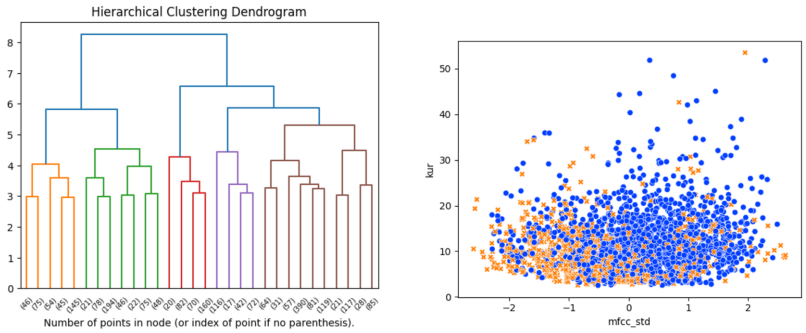

Figure 11: Dendrogram and scatter plot for *sex* after agglomerative clustering

2.3.2 Ward linkage

Here we used the ward linkage, which minimizes the variance of the clusters being merged. In the ward's method the proximity between two clusters is defined as the increase in the squared error that results when they are merged, so it is very similar to the objective function of the K-means.

While the results are comparable (table:8) with the complete linkage ones, we can notice a difference in the resulting dendrograms (fig:12).

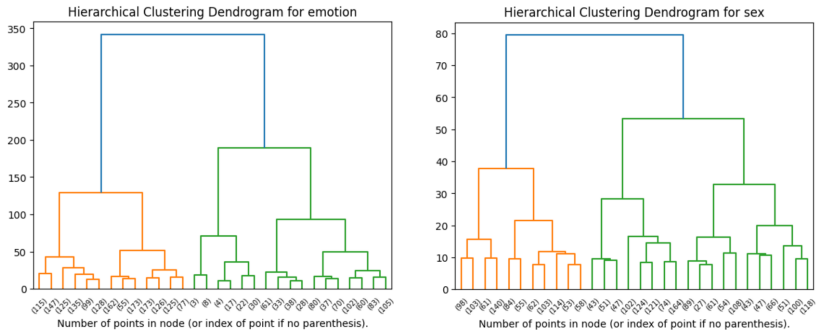

Figure 12: Dendrogram with ward linkage

	Silhouette
emotion	0.33
sex	0.28

Table 8: Hierarchical with ward linkage evaluation

2.4 Final discussion

We have a dataset represented by a data matrix, where each data objects represent a point in a 24 dimensional space (*sc_min*, *intensity* and *stft_min* not used). Our analysis aimed at detection of *emotion* and *sex* values exploiting the clusters created by the different methods and using selected attributes (sec:2.1), so the algorithms acted on a 6-dimensional space for *emotion* and a 4-dimensional space for *sex*.
We observed that for *emotion* none of the algorithms succeed to give a right interpretation to given dataset, maybe the problems are due to distribution of *emotion* respect to other attributes that doesn't seems to be clearly distinct. Indeed, the *emotion* values often overlap each other, as we can see in fig:9. Thus, this gets the algorithms in trouble because the real distribution doesn't correspond to a minimum of their objective function.
With the *sex* attribute all the algorithms seems to work better, surely also due to the fact that *sex* represent a binary attribute and this definitely simplifies the detection. In table:9 we reported the confusion matrix for the k-means (bisecting k-means has analogous results), DBSCAN and hierarchical clustering (ward linkage). From this table we can notice that k-means had the best results for the *sex* values detection.
This result does not surprise us because the density-based methods present the same troubles as in the *emotion* case. Indeed, if we look the confusion matrix we see that the values are equally distributed both in the denser region and in the sparse region detected from the algorithm. This reflects the distribution of *sex* with respect to the selected attributes, as we see in fig:9. Hierarchical partly has the same problem as DBSCAN, because it performs locally optimization, and so it's just the k-means that performs a global optimization minimizing the global SSE.

3. Classification

In this section we will perform classification using three methods: Decision Tree, K-NN and Naive Bayes.
We decided to apply the classification methods to both *emotion* and *vocal_channel* features. Before looking at these models in detail, it is worth highlighting the different encoding of the *emotion* attribute needed in the two cases: in the case where *vocal_channel* was the target, we applied the one-hot

K-means label

Actual value	0	1	total
M	1132	101	1233
F	249	939	1188
total	1381	1040	

DBSCAN label

Actual value	0	-1	total
M	831	402	1233
F	836	352	1188
total	1667	754	

hierarchical label

Actual value	0	-1	total
M	1123	110	1233
F	449	739	1188
total	1672	849	

Table 9: *Sex* confusion matrix for K-means, DBSCAN and Hierarchical

encoding to the *emotion* feature, because emotions haven't a natural relationship to each other, such as a natural ordering; in the other case, we encoded the *emotion* values just with integers, because the only relevant thing was to have some identifier for the different emotions and, even more importantly, to have all the values contained in just one column.
Another important aspect is that we decided not to perform feature selection because of the nature of the algorithms to automatically identify the best attributes to be tested.

3.1 Decision trees

The tendency of the decision trees to overfitting the training data is visible in the trees obtained without selecting the hyperparameters (fig:13), with the corresponding results reported in table:10.

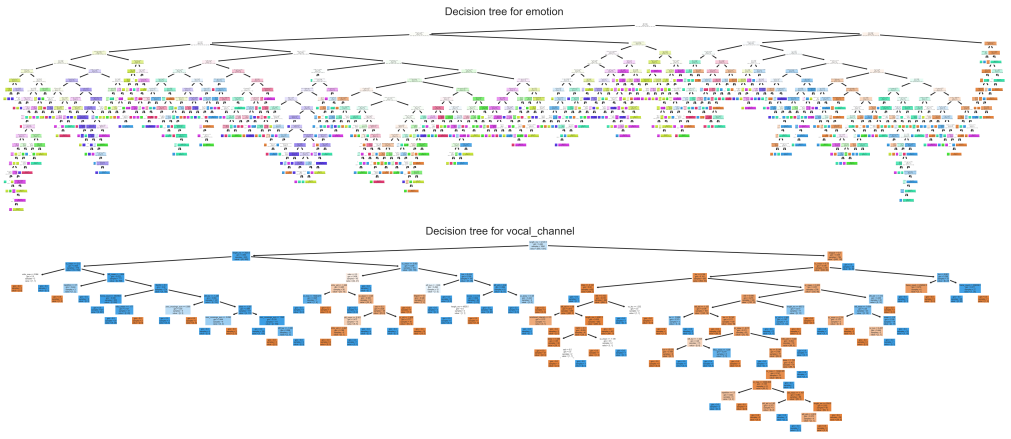

Figure 13: Decision Tree without selecting hyperparameters

emotion accuracy:	0.41
emotion F1-score:	0.40
vocal_channel accuracy:	0.93
vocal_channel F1-score:	0.92

Table 10: Decision Tree evaluation

Because of that, we used a grid search to find the best values for `criterion`, `max_depth`, `min_samples_leaf` and `min_samples_split`. Given the dimension of the emotion tree, we decided to limit `max_depth` range of values to a maximum of 20. The results are reported in table:11.

best emotion `criterion`	entropy
best emotion `max_depth`	16
best emotion `min_samples_leaf`	1
best emotion `min_samples_split`	3
emotion accuracy	0.36
emotion F1-score	0.33
best vocal_channel `criterion`	gini
best vocal_channel `max_depth`	6
best vocal_channel `min_samples_leaf`	3
best vocal_channel `min_samples_split`	10
vocal_channel accuracy	0.93
vocal_channel F1-score	0.94

Table 11: Results after grid search

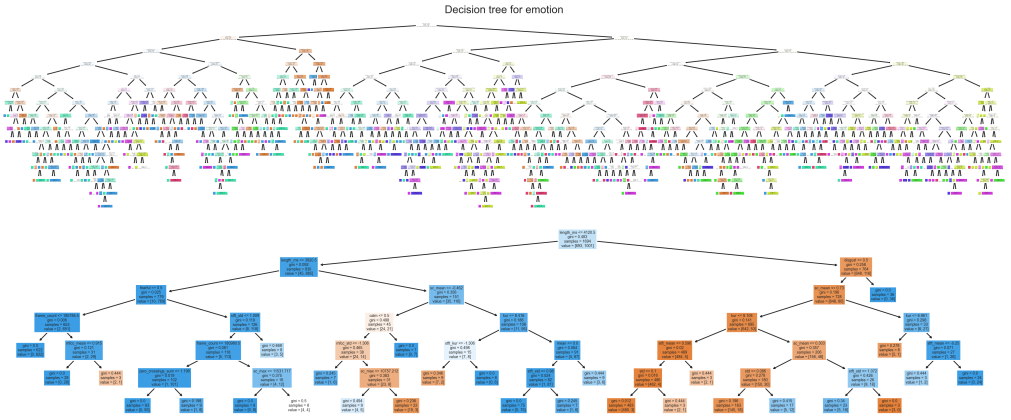

Figure 14: Decision three after grid search.

Because of the behaviour of the emotion tree, whose accuracy and F1-score decreased because of the limit to the `max_depth`, and given that also the initial scores were not ideal, we decided to try to post-prune just the decision tree for the *vocal_channel* feature.
To post-prune the tree, we reserved a validation set in order to find the best `ccp_alpha` parameter. We trained the model, we used the grid search to find the best hyperparameters for the pre-pruning of the tree and then we used a minimal cost-complexity pruning:

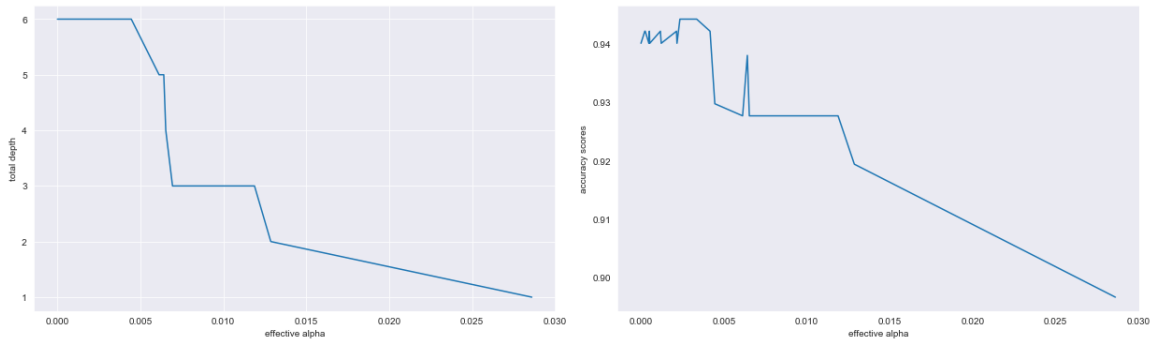

Figure 15: Post-pruning optimization

The best value of `ccp_alpha` calculated on the validation set was 0.00235. In fig:16 there's the resulting tree, in fig:17 the ROC curves and in table:12 the evaluation results.

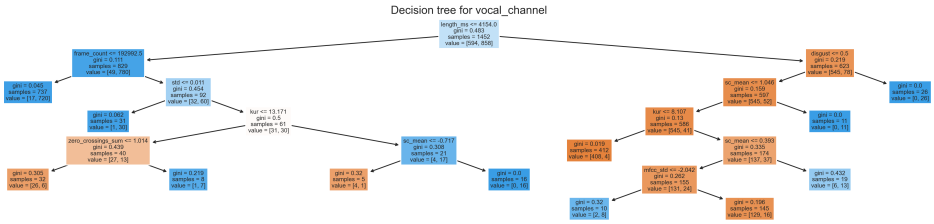

Figure 16: Decision Tree after post-pruning

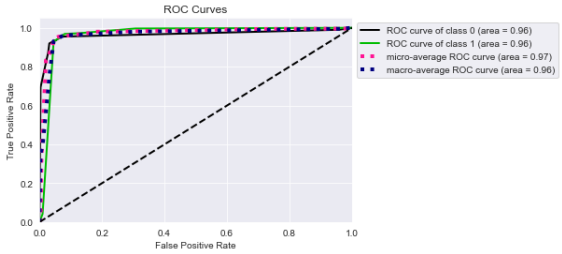

Figure 17: ROC curves

accuracy:	0.948
F1-score:	0.946
ROC AUC	0.965

Table 12: Evaluation after post-pruning

3.2 K-NN

The Nearest-Neighbor Classifier is an example of lazy learner, it delay the process of modelling the training data until it is needed to classify the test example, in fact it tries to find all the training examples that are relatively similar to the attributes of the test example. So, in order to best apply the algorithm, there are three hyperparameters to be set: metric, n_neighbors and weights.
In order to do so, we decided to apply a grid-search and find the best hyperparameters' values:

emotion : **metric**: cityblock **n_neighbors**: 23 **weights**: distances **best_score**:0.41

vocal_channel : **metric**: cityblock **n_neighbors**: 24 **weights**: distances **best_score**:0.95

We can also see the behaviours for different values of the `metric` and `weights` parameters and the ROC curve in fig:18. The ROC AUC values calculated on the test sets are in table:13.

	ROC AUC
emotion	0.82
vocal_channel	0.99

Table 13: ROC AUC for K-NN

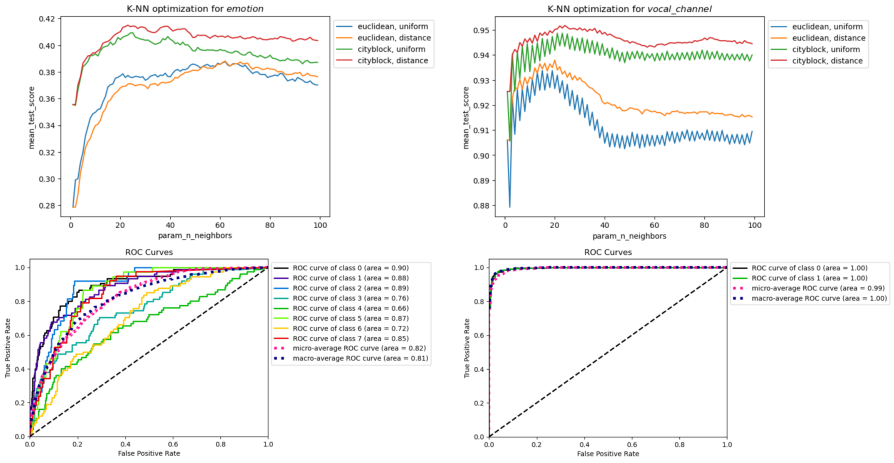

Figure 18: Settings optimization and ROC curve

3.3 Naive Bayes

Bayesian Classifier are used when the relationship between the attribute set and the target class is non-deterministic, that is the class label of a test example cannot be predicted with certainty even though its attributes are identical to some of training examples.
The Naive Bayes in particular assume that the attributes are conditionally independent, given the class label. Assumption that we will take as true, although we cannot verify it. Moreover the Naive Bayes assumes Gaussian form of probability distribution for the continuous variable, so to perform the analysis we take only the Gaussian distributed attributes as shown in 1.2. We reported the ROC curve in fig:19, and the results of ROC AUC for the test sets in table:14.

	ROC AUC
emotion	0.73
vocal_channel	0.98

Table 14: ROC AUC for Naive Bayes

3.4 Final discussion

In this final discussion, we'll talk about the classification models applied to the *vocal_channel* attribute. In order to compare the previous models, we'll look at the confusion matrix heatmaps (fig:20). Even though all the three models are more than valid, looking at the heatmaps we can say that the naive bayes classificator works slightly worse than the other two.

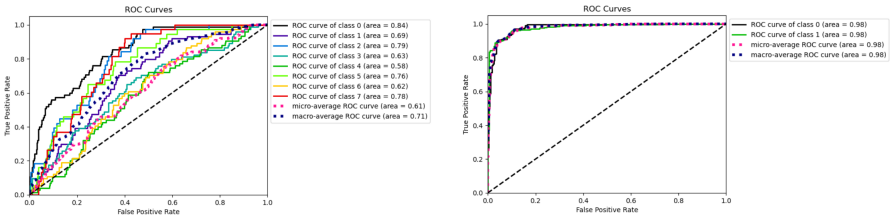

Figure 19: ROC curve for naive bayes

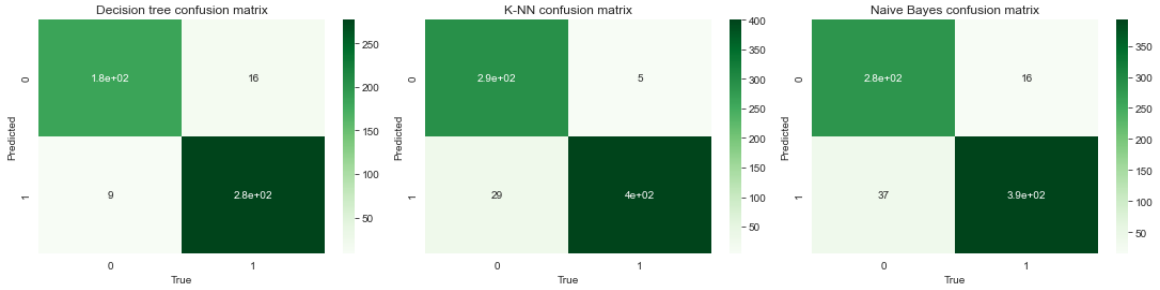

Figure 20: Confusion matrix heatmaps

As we expected, we can see from Figure 20 that K-NN and Naive classificators work better for the *song* values, because we have less *speech* values in the dataset, and consequently in the training set.

What is interesting here is the fact that the decision tree shows an opposite behaviour. Actually, the reason is probably that the decision tree is able to understand that the *speech* values have two more emotions, *disgust* and *surprised*, so if an object have one of them it is classified as *speech*.

4. Pattern Mining

In order to apply pattern mining algorithms, we needed to have only categorical features into our dataset, so attributes like *vocal_channel* or *emotion* have been mapped to their old values. For each numeric attribute, instead, values we grouped its values into four buckets based on its quartiles, and we concatenated each of these groups (represented as strings) to its feature name. For example, a categorical value for the attribute *std* can be `(0.0139, 0.0262]_std`.

4.1 Frequent itemsets

Firstly, we generated the frequent itemsets using two different values of support, 10 and 20. We will not report the list of rules because they are not easy to read at first sight.
The result of using different thresholds was to obtain 73 frequent itemset in case of `support=20` and 1492 in case of `support=10`. The same exponential growth with fewer values of the support threshold can be observed with closed and maximal itemsets. fig:21 is useful to summarize this behaviour.Obviously, the problem of choosing a lower support threshold can be resulting into select non-significant itemsets.

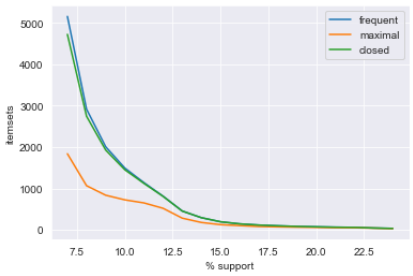

Figure 21: Number of itemsets according to support threshold

4.2 Association rules

The association rules generated by Apriori and FP-Growth algorithms are the same, and changing the confidence threshold results into a significant increment of the generated rules. For example, fixing the support value to 20, we obtained 101 rules using `confidence=70` and 977 rules using `confidence=50`. In fig:22 are the plots of confidence and lift for the generated rules.

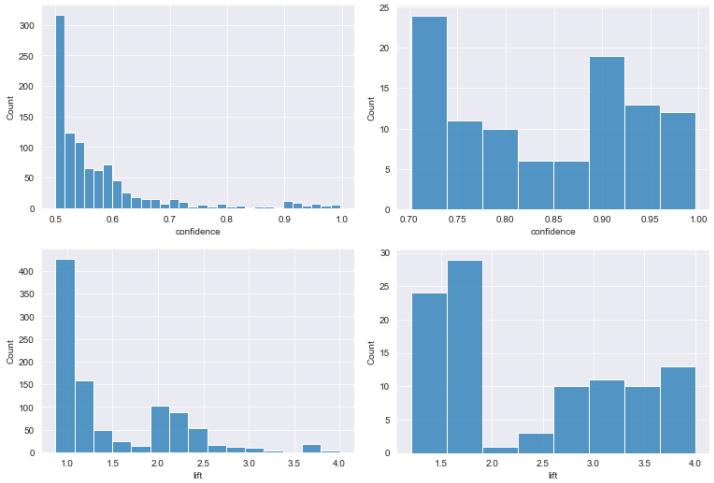

Figure 22: Confidence and lift of generated rules

What we can observe from these plots is that rules generated by `confidence=50` (plots on the right) tend to have a lot of values with a confidence between 0.5 and 0.6. Given that the confidence of a rule is the number of cases in which the rule is correct relative to the number of cases in which it is applicable, we can say these rules are probably not really useful.
In addition, these new rules tends to have a value of lift near to 1, which means that, given a rule $X \rightarrow Y$, $P(X|Y)$ is similar to $P(Y)$. Again, we can conclude that these rules are probably not significant.

4.3 Classification using association rules

Using `support=20` and `confidence=70`, we obtained a total of 30 rules useful to classificate the *vocal_channel* attribute, in table:15 we reported the first three rules for both values of *vocal_channel*.

consequent	antecedent	confidence	lift
speech	((3604.0, 4004.0]_length_ms, (172972.0, 190591....	0.960360	1.625897
speech	((3604.0, 4004.0]_length_ms,)	0.954622	1.616181
speech	((14.055, 53.501]_kur,)	0.927273	1.569879
song	((-2.8409999999999997, -0.686]_sc_mean,)	0.727723	1.777817
song	((2.507, 6.53]_kur,)	0.902640	2.205138
song	((-2.806, -0.732]_mfcc_max, F)	0.703839	1.719470

Table 15: Examples association rules for *vocal_channel* detection

5. Regression

As anticipated in section 1, our goal, using regression, was to find the missing values of three attributes: *stft_min*, *sc_min* and *intensity*.
Each time we splitted the dataset into a training set and a test set. Then, we used the training set to select the most interesting features with `SelectKBest` through two score functions: `mutual_info_regression` and `f_regression`.

5.1 Missing values of stft_min

In fig:23 there are the histograms obtained using `SelectKBest` with `mutual_info_regression` and `f_regression`:

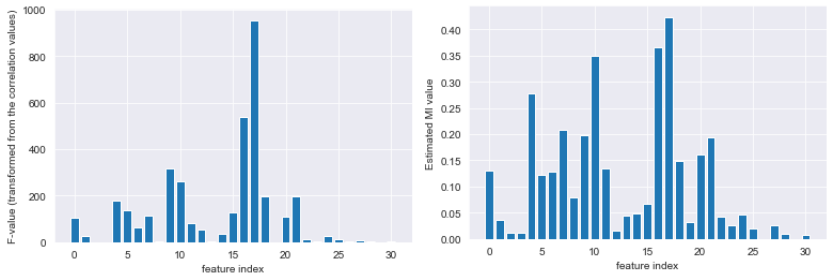

Figure 23: Feature scores for $stft\_min$

We selected the best 10 features according to `f_regression`, we scaled our training set through the `MinMaxScaler` and we used the K-NN model for regression:

```
neighbors.KNeighborsRegressor(n_neighbors=k, weights='distance')
```

The choice of the model, here and for the other two features to fill, has been made not only looking at the RMSE and R^2 values, but also to the resulting distribution of the attribute (if data follow any particular distribution and it's homogeneous, with no random peaks), considering anyway that if there are highly correlated attributes a linear regressor is generally suggested. In fig:24 are the RMSE and R^2 obtained by varying the value of K:

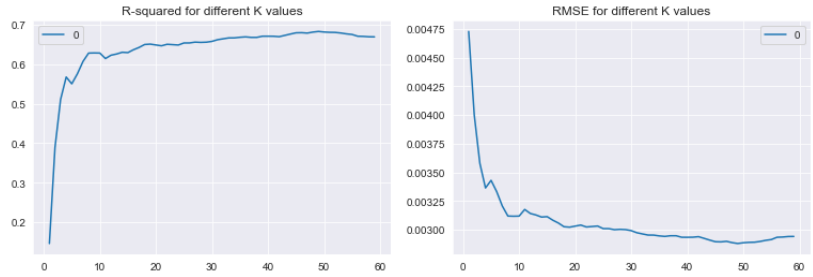

Figure 24: RMSE and R^2 for stft_min

Choosing K = 7, we obtained RMSE = 0.0032 and R^2 = 0.6070, and we used this model to predict the missing values. In fig:25 there is the distribution of *stft_min* after filling it with the new ones, which correspond to an exponential distribution.

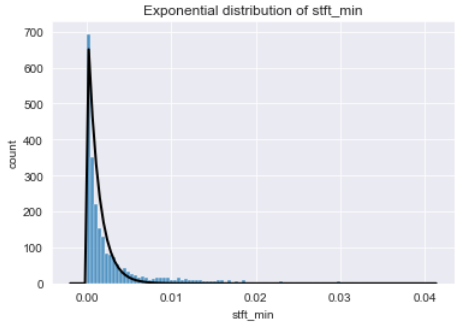

Figure 25: Distribution of $stft\_min$ updated

5.2 Missing values of sc_min

Differently from the previous section, a linear regressor turned out to be the best model to find the missing values of the *sc_min* feature.
Again, to evaluate the model, we considered not just the RMSE and the R^2 error (which were similar in every model we tried) but also the distribution obtained.
Firstly, we used the same approach used for *stft_min* to select the best features, but this time our model worked better using them all. This is true also for the *intensity* attribute, for which we used a linear regressor again. Anyway, these are the scores on each feature given by `mutual_info_regression` and `f_regression`:

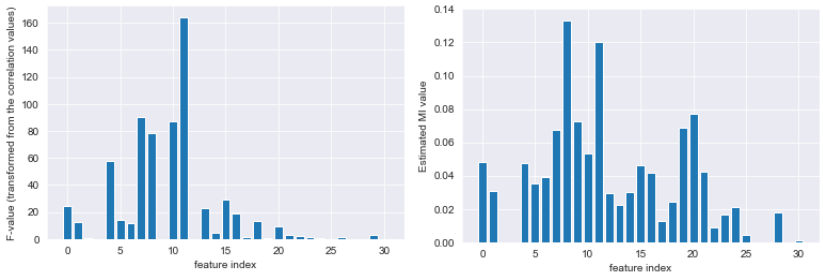

Figure 26: Feature scores for *sc_min*

After training the model, we obtained RMSE = 178.1847 and R^2 = 0.5258 on the test set, we used it to predict the missing values and we obtained a Gaussian distribution for *sc_min*:

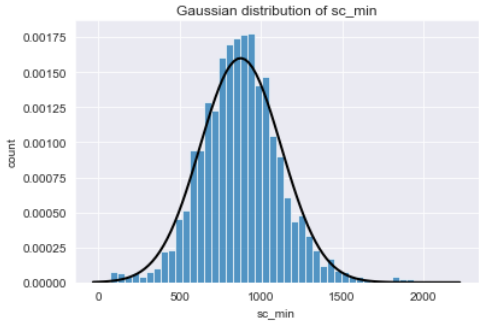

Figure 27: Distribution of *sc_min* updated

5.3 Missing values of intensity

We used a linear regressor to predict the missing values of *intensity*. What's different here is that the regression is mostly affected by two attributes as reported in fig:28.

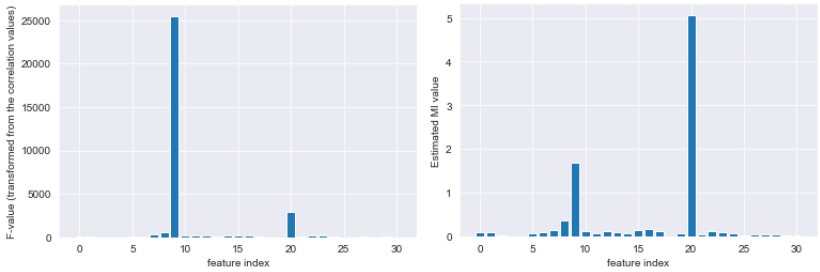

Figure 28: Feature scores for *intensity*

After training the model, we obtained RMSE = 0.1479 and R^2 = 0.9774 on the test set, we used it to predict the missing values and we obtained a gaussian distribution for *intensity* as we can see in fig:29.

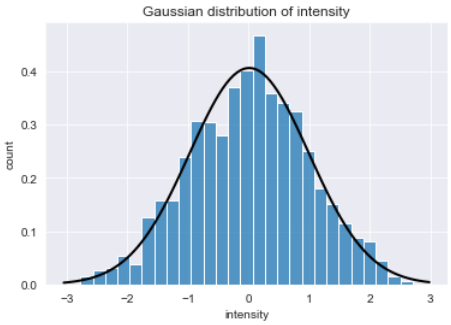

Figure 29: Distribution of *intensity* updated

Anyway, we found out it still was strongly correlated with *mfcc_std* (−0.98), so we decided to drop the *intensity* attribute from the dataset. The reason is that, between the two, we trusted more *mfcc_std* because values have been measured, while half of the *intensity* ones are just predicted.