UNIVERSITY OF PISA

DEPARTMENT OF INFORMATICS

Master Degree in Data Science and Business Informatics

Data Mining 1

Davide Piccoli – Lorenzo Parra – Matteo Di Giorgio

Academic Year 2022/2023

INDEX

1. Introduction

The aim of this report is to provide some analysis related to the RAVDESS dataset, to discuss it and try to give an explanation of the results. This dataset contains recordings by 24 professional actors, in which two different statements are claimed by a varying number of people. The way these statements are vocalized can be either by a speech, i.e. talking with a normal tone of voice, or by a song, i.e. singing the statement.
For each of these recordings, many attributes were selected, measured and inserted to the dataset.

2. Data understanding and preparation

The first step was importing the data and all necessary modules for its observation. This allowed some dataframe exploration: starting from some basic functions, it was possible to know the exact shape of the dataframe, as well as the name of its variables and their type.
The original dataset contains 2452 recordings, for each of which 38 attributes were observed. These attributes are explained in the following list:

1 . Modality: it explains the record modality and indeed it only has one possible value, which is "audio-only".

2 . Vocal channel: this variable is about the way the recording is vocalized. As already stated before, the two possible values here are "speech" and "song".

3 . Emotion: this describes the emotion felt by the actor while recording the audio. Possible values here are "neutral", "calm", "happy", "sad", "angry", "fearful", "disgusted", "surprised".

4 . Emotional intensity: this describes the intensity of the previously defined feature. Its possible values are "normal" and "strong".

5 . Statement: this variable defines which one between the two possible statements is vocalized in the recording. These two statements are "Kids are talking by the door" and "Dogs are sitting by the door".

6 . Repetition: since every statement is vocalized twice by every group of actor, this variable defines whether the recording includes the first or the second repetition of it. Therefore, possible values are "1st repetition" and "2nd repetition".

7 . Actor: recordings can be vocalized either in group or just by one person. This variable specifies how many actors are involved in the recording. Possible values are all numbers between 1 and 24.

8 . Sex: it specifies the gender of the actor/actors involved. Possible values here are "M" and "F".

9 . Channels: this variable involves the kind of signal of each recording. Possible values are "1" for mono audios and "2" for stereo audios.

10 . Sample width: each audio file can be considered as a sequence of frames, where each frame is consisting of samples[1]. There is one sample per channel, therefore sample width measures the number of bytes per sample. Typically this is either 1 or 2 bytes. Indeed, possible values here are "1" and "2" and they are related to the one given for the attribute "Channels" explained above.

11 . Frame rate: it indicates the frequency of sample used, measured in Hertz.

12 . Frame width: this variable specifies the number of bytes for each frame.

13 . Length ms: this attribute is about the length of each audio, measured in milliseconds.

14 . Frame count: it indicates the number of frames from the sample.

15 . Intensity: this variable is about the difference between the loudness of each recording and the maximum possible loudness, measured in dBFS[2].

[1] Work with WAV files. https://doc.sagemath.org/html/en/reference/misc/sage/media/wav.html

[2] dBFS means decibels relative to full scale. 0 dBFS is assigned to the maximum level of loudness. An audio recording whose signal reaches 50 percent of the maximum level of loudness would have a value of -6 dB FS. https://www.digitizationguidelines.gov/term.php?term=decibelsrelativetofullscale

16 . Zero crossings sum: this indicates the sum of the zero-crossing rate for each audio recording. The zero-crossing rate is the number of times the signal changes value, from positive to negative and vice-versa, divided by the length of the frame[3]

17 . Statistics of the original audio signal, for which we have mean, standard deviation, minimum, maximum, kurtosis and skewness.

18 . Statistics of the Mel-Frequency Cepstral Coefficients, consisting in logarithmic transformation of the signal's frequencies[4]. For this variable, we have mean, standard deviation, minimum and maximum.

19 . Statistics of the spectral centroid, which basically refers to the center of mass of the spectrum related to each audio signal[5]. In particular, the dataset provided mean, std, minimum, maximum, kurtosis and skewness for this attribute.

20 . Statistics of the stft chromagram, used to infer properties about the distribution of a signal's energy[6]. Here the attributes involved are the same already listed for the previous point.

The function named "info" lets us notice some missing values for the attributes vocal_channel, actor and intensity. In order to deal with these missing values, we wanted to avoid any changes on the original distribution of the data, so we decided not to use a simple imputation based on a particular statistical measure, such as mean or median. As for actor and vocal channel, we first analysed the distribution of their values through histograms. The function "random.choice" was then used so that the percentage of each occurrence for both attributes (i.e. speech or song for vocal_channel) could be computed and then recalled to fill the missing values. The following images show a comparison between original histograms and the new ones obtained after this operation: the distribution looks exactly the same as before.

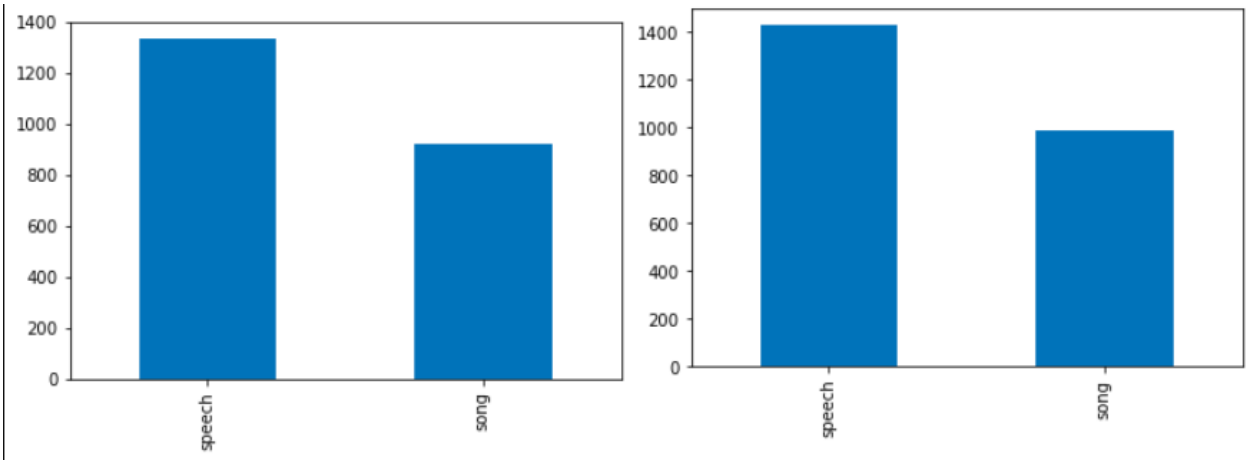

Figure 1: Vocal channel distribution before and after filling missing values.

[3]Zero Crossing Rate. https://www.sciencedirect.com/topics/engineering/zero-crossing-rate

[4]Learning from Audio: The Mel Scale, Mel Spectrograms, and Mel Frequency Cepstral Coefficients. https://towardsdatascience.com/learning-from-audio-the-mel-scale-mel-spectrograms-and-mel-frequency-cepstral-coefficients-f5752b6324a8

[5]A Tutorial on Spectral Feature Extraction for Audio Analytics. https://analyticsindiamag.com/a-tutorial-on-spectral-feature-extraction-for-audio-analytics/

[6]Chromagram visualization of the singing voice. https://www.isca-speech.org

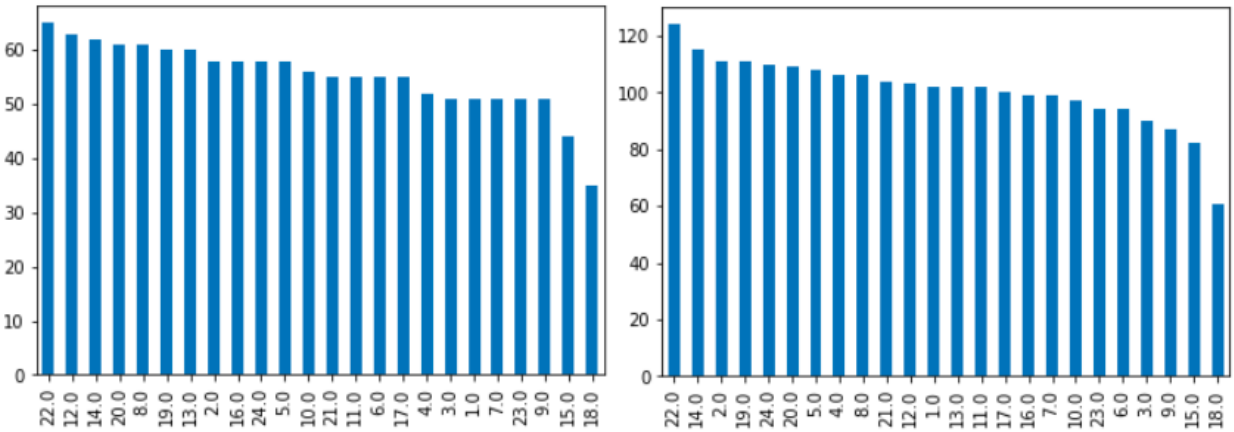

Figure 2: Actor distribution before and after filling missing values.

For what concerns missing values in the “intensity” attribute, we chose a different way. At the beginning we tried with the same method used for ”vocal_channel” and ”actor”, but then some relevant changes on the correlation between ”intensity” and attributes related to statistical aspects, such as the standard deviation of the original audio signal, were noticed. We also tried both linear and polynomial interpolation, but nothing relevant was changed with respect to correlation with statistical variables and in particular ”std”. Indeed, our final decision was an interpolation exactly based on the correlation between ”intensity” and ”std” itself. Unfortunately, this method still causes some correlation loss between intensity and other statistical attributes, such as “min”, “max”. However, the loss is just by 0.25, so we left this method as valid, since the alternative option would have been dropping all rows with missing values in this column.

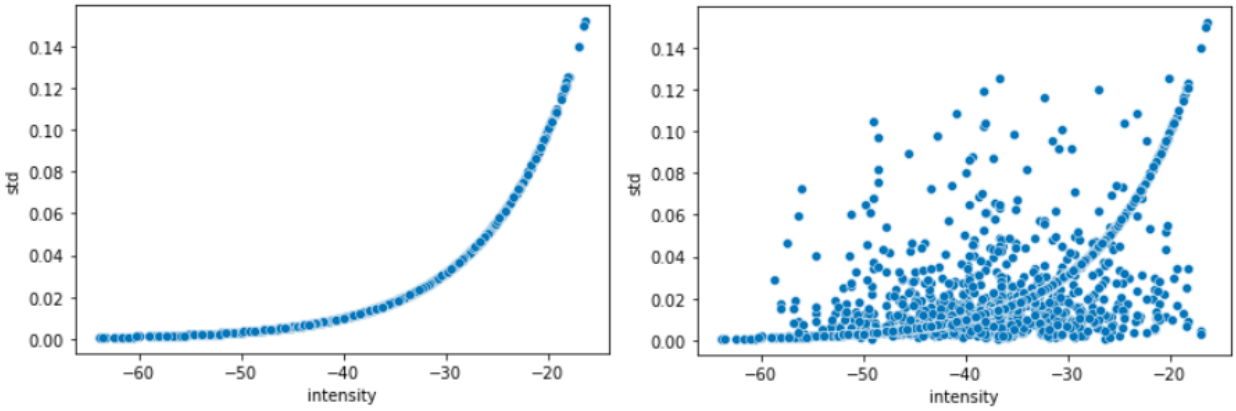

Figure 3: Correlation between ”intensity” and ”std”, next to the same correlation after filling missing values with linear interpolation. Since many noise points were added after this operation, we switched to a filling method based on the scatterplot on the left.

After dealing with missing values, we looked for errors. For those attributes whose possible values were finite, we used .unique command, which returns an array with all the possible entries which can be read along its column, but nothing interesting was found. Other numerical variables were explored by creating some scatterplots between couples. This method allowed us to notice some unexpected values along the column related to frame count: 6 recordings had a value of -1. We decided to delete these rows, because an audio with frame count = -1 has no information in it, and in addition to this 6 rows are just a very small slice of the whole dataset.
Then we called a loop to iterate over those columns whose variables were continuous only: in this way we could get a boxplot for each of them. We thought this could be useful to detect other possible errors as well as some outliers. Boxplots also helped us understand which columns had just the same value for every row (duplicates): we decided

to drop all of them.
Specifically, first of all we noticed that only 6 rows had the attribute frame width =4, meaning that only 6 recordings had channels=2. We decided to drop these rows and therefore we dropped both columns as well, since they then had the same value for each instance.
Then, we focused on modality, sample width and frame rate, which had the same value for every record, so these columns were dropped too.
The scatterplot where "frame count" and "length_ms" were compared highlighted a perfect correlation between these two attributes. In order to avoid multicollinearity issues, we decided to drop "frame count". Indeed, the plot below detects some errors on the first attribute, but also highlights a perfect correlation between the two variables.

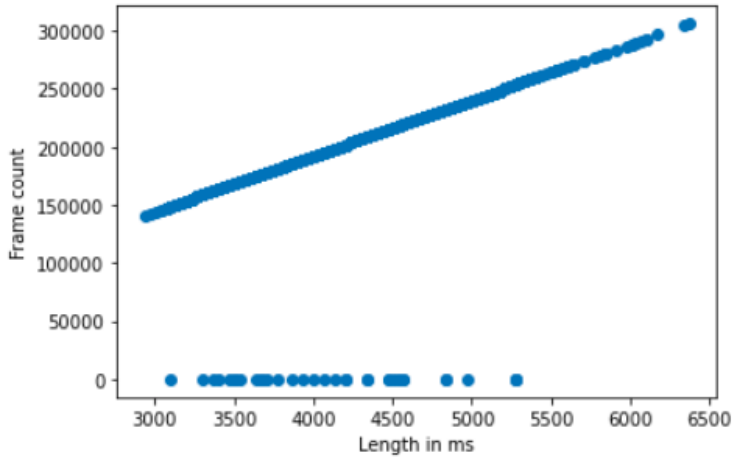

Figure 4: Original correlation between "frame_count" and "length_ms." A perfect correlation can be noticed, as well as some outliers where frame_count was equal to -1, as already said.

We then looked for other possible multicollinearity issues. The correlation matrix below was indeed useful to observe correlation values between each couple of those variables that were left after the cleaning process. The highest correlation can be noticed between statistical variables, as well as between the "intensity" variable and some statistical variables again. This probably confirms that filling missing values in the "intensity" column with an interpolation based on their correlation with "std" provides a reliable result.

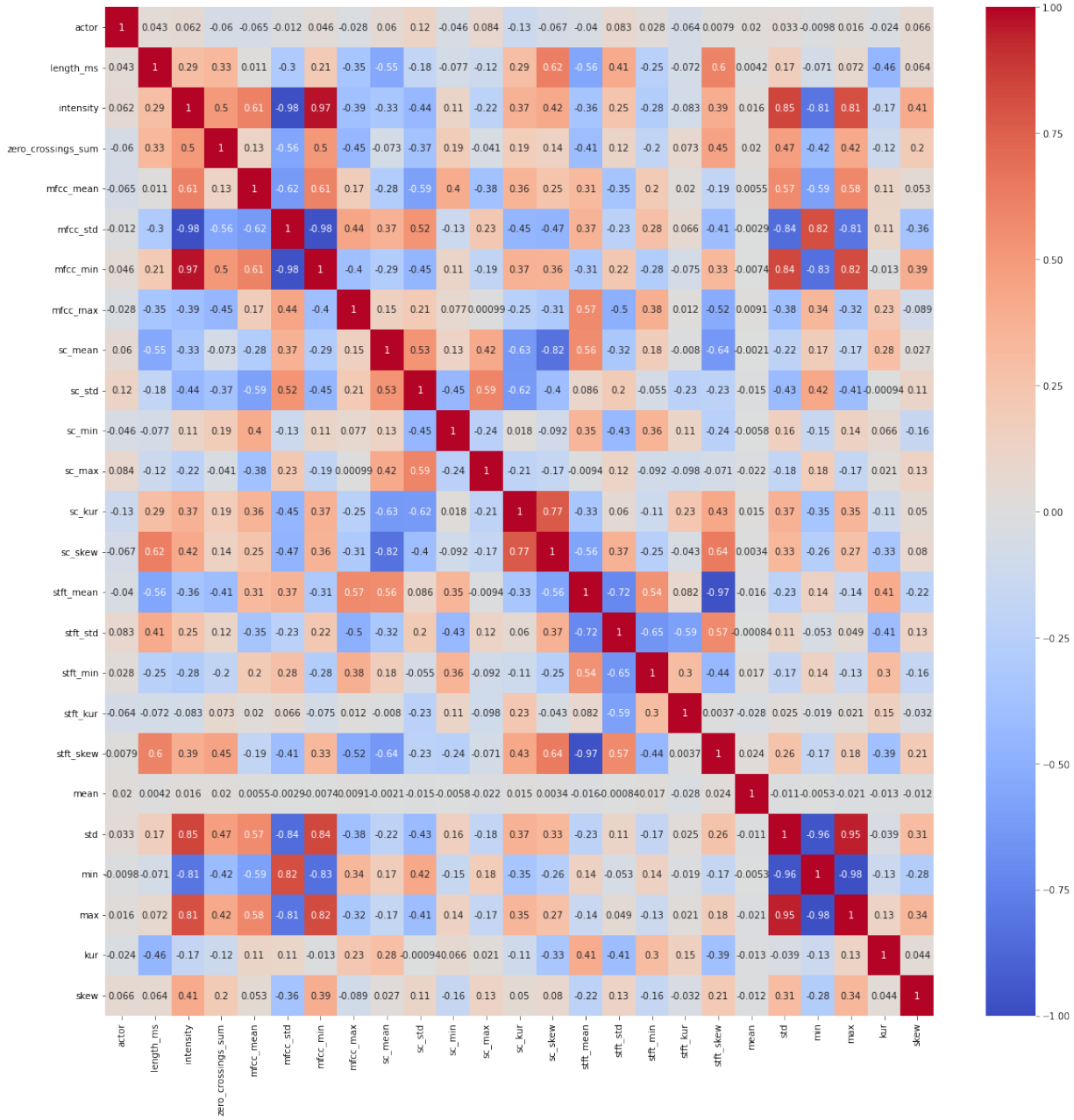

Figure 5: Correlation matrix.

An outlier was noticed on the boxplot over the zero crossings sum variable: we dropped it. The same happens to a few outliers on the boxplot related to "std".
Finally, we dropped the whole column related to the attribute "sc_min", because almost half of the instances had a value equal to 0 here.

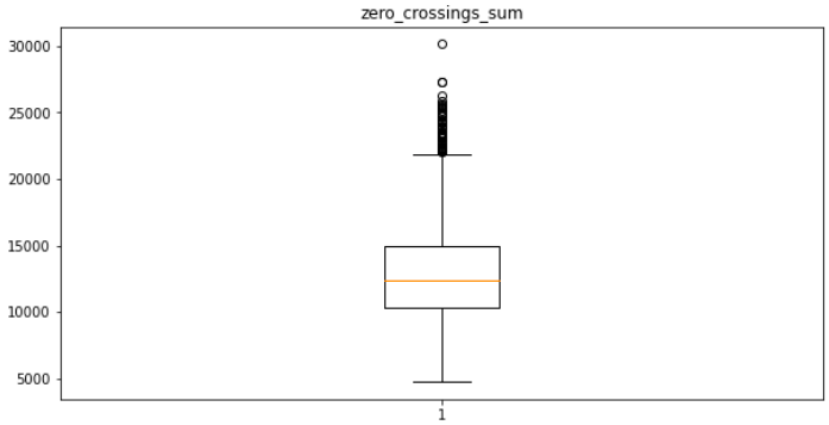

Figure 6: Boxplot on "zero_crossings_sum", highlighting an outlier equal to 30153.

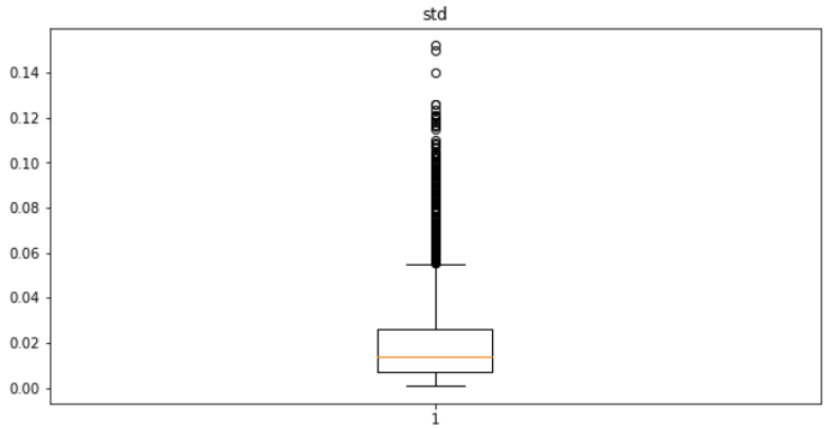

Figure 7: Boxplot on "std", highlighting some outliers around and above 0,14.

3. Clustering: introduction

Each clustering method was implemented both on the whole dataset and on some partitions of it. These partitions were obtained setting all values for a particular attribute as the same for all instances. Slices of the original dataset based on different vocal channels or different genders of the actors did not lead to anything relevant, while partitions based on different emotions provided the most interesting results.
Before proceeding with the implementation of the clustering algorithms, all numerical attributes were isolated from the original dataset, so that they could be normalized by the MinMaxScaler method.

3.1. K-means

Our initial attempts with this method involved the whole dataset. A for-loop was implemented in order to plot the SSE: this should have told us the number of clusters to look for. The range of values for K in which the SSE has been plotted is between 2 and 50. However, it was not possible to find any particular value of k where the derivative changed significantly and got close to 0.

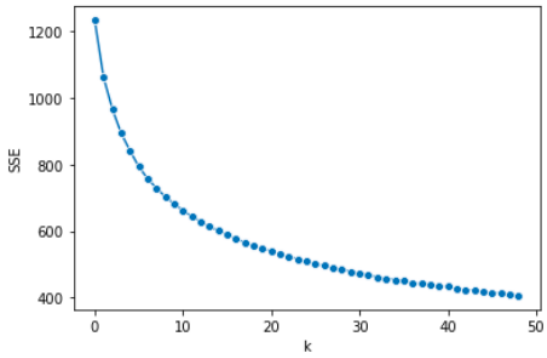

Figure 8: SSE curve implemented on the whole dataset. Sub-datasets with respect to vocal channel returned a very similar result.

Indeed, results obtained by implementing K-means on the whole dataset are not that useful. For example, the first image below represents the scatterplot between "length_ms" and "zero_crossings_sum" with respect to "vocal_channel", taking the whole dataset as input. The second image shows the scatterplot between the same attributes with kmeans labels applied on, with k=2. The same kind of distinction is still notable, but not that informative. We did not obtain any better result than this one when applying the algorithm to the whole dataset: this is the reason why we decided to focus on partitions of it.

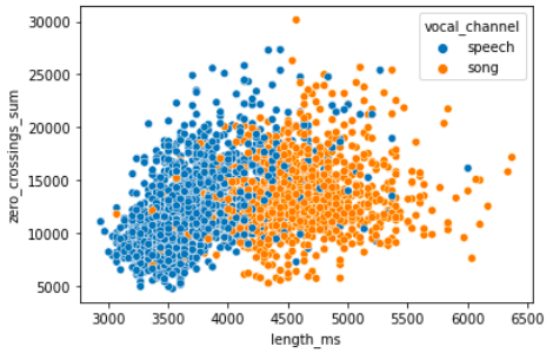

Figure 9: Scatterplot between this couple of attributes, denoting some kind of distinction with respect to the class.

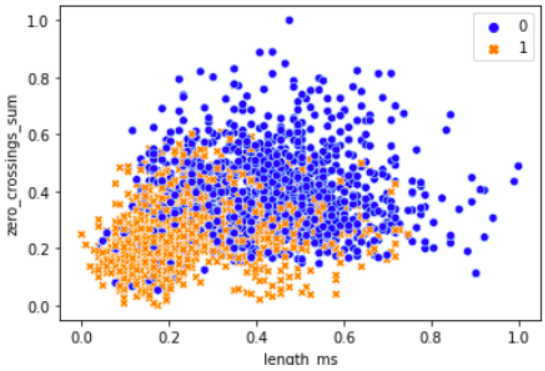

Figure 10: K-means method result shown on the same couple of attributes.

Considering a dataset where the only two attributes are "length_ms" and "mfcc_max" and all instances have just a "neutral" emotion, the SSE curve is much more informative than the one previously observed. As the image below shows, this plot provides a more evident elbow, which implies a value around 2 as the optimal amount of clusters

to look for. The plots obtained for every other sub-dataset based on a different emotion, with respect to the same variables are also very similar to this one.

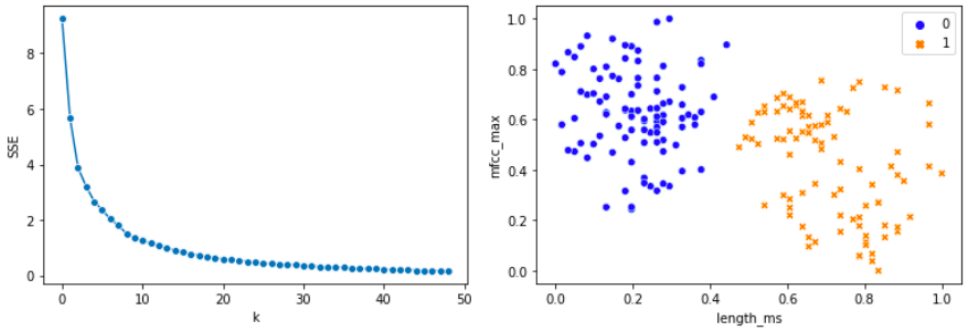

Figure 11: On the left: SSE curve implemented on a dataset with only "length_ms" and "mfcc_max" as attributes, isolating instances with only "neutral" emotion. On the right: the scatterplot obtained implementing K-means on the same sub-dataset can be observed.

Also the following results were found by implementing the algorithm on two-attribute sub-datasets were only one emotion was selected.

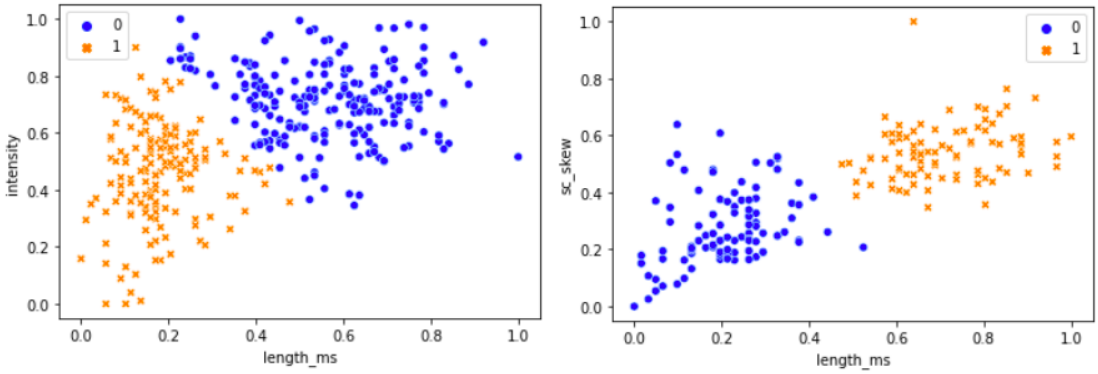

Figure 12: On the left: scatterplot from K-means on a sub-dataset with the couple of attributes indicated on the axis and only "sad" instances. On the right: scatterplot from K-means on a sub-dataset with the couple of attributes indicated on the axis and only "neutral" instances.

3.2. Xmeans and bisecting K-means

These methods did not return anything more useful than what had already been obtained with K-means implementations.
Since implementations on sub-datasets with only two attributes gave the same result provided by K-means, Xmeans was implemented also on sub-datasets with more than two attributes, still isolating same kinds of emotions, in order to observe if at least in this case something different could be obtained. One of the outputs can be observed below: in this case we let the algorithm find the best number of clusters by itself.

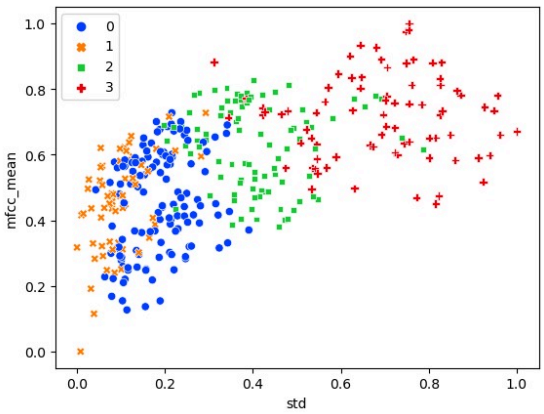

Figure 13: Scatterplot from Xmeans implementation on a dataset with all the original attributes and only "angry" instances. The attributes "mfcc_mean" and "std" were those giving this result when plotted.

Furthermore, the image below shows the implementation of bisecting K-means on the same sub-dataset on the left of figure 11. The only notable difference from K-means is just the way some points in the middle are classified.

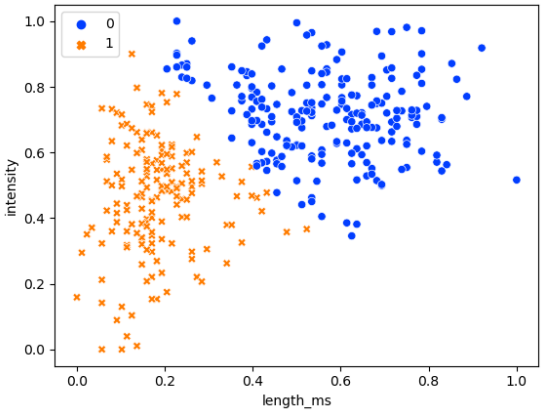

Figure 14: Scatterplot from bisecting K-means on a sub-dataset with the couple of attributes indicated on the axis and only "sad" instances.

3.3. Hierarchical clustering

Hierarchical clustering was mostly implemented with Ward's method and complete linkage method, since the majority of the clusters obtained with K-means method had too many noise points for the single linkage method to be efficient. The dendrograms below are the result of Ward's method applied to, again, different sub-datasets based on different emotions. Two clear partitions can be noticed.

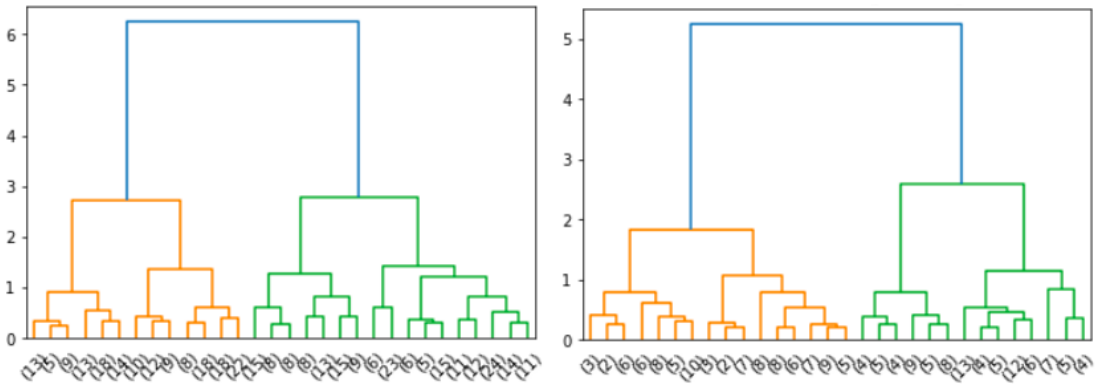

Figure 15: On the left: dendrogram from a sub-dataset with only "sad" instances and "length_ms", "stft_mean" as attributes. On the right: dendrogram from a sub-dataset with only "neutral" instances and "length_ms", "mfcc_max" as attributes.

The dendrogram below also provided a good partition between two clusters and its results were represented on a scatterplot as well.

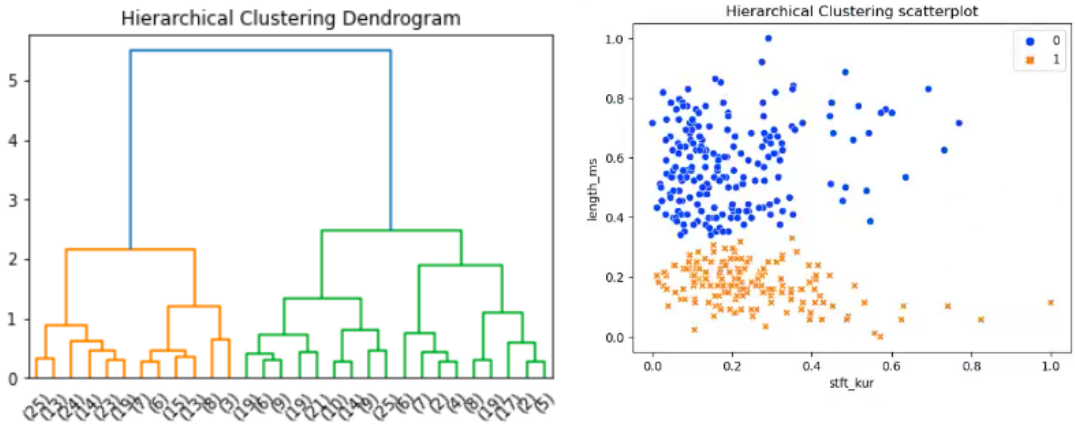

Figure 16: On the left: dendrogram from a sub-dataset with only "sad" instances and "length_ms", "stft_kur" as attributes. On the right: scatterplot based on the results.

3.4. DBScan

At the beginning, the choice regarding which value should have been assigned to epsilon and minpoints was rather casual. Then, these parameters were tuned based on both the results obtained and the related plots of the distance from the kth neighbor.
Something interesting was found on sub-datasets isolating "song" instances. The image below on the left represents the trend of the distance from the 60th neighbor for a dataset with "intensity" and "mfcc_mean" as attributes: this provided us with the right value to set Eps at. The image on the right shows the related result.

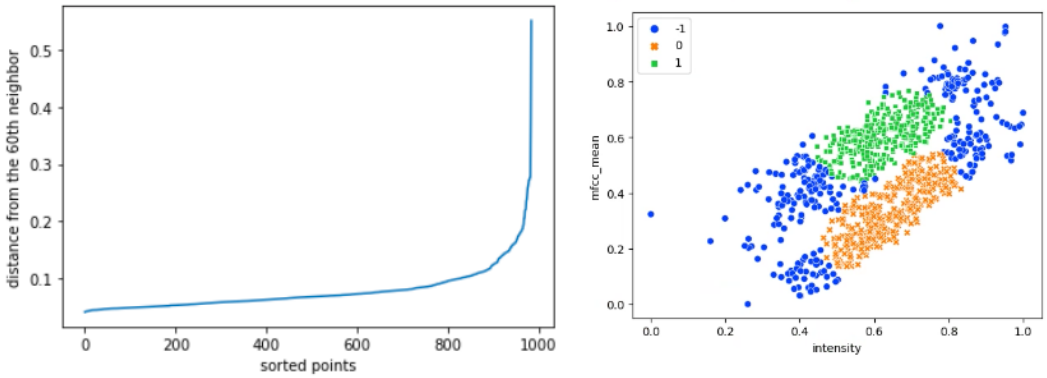

Figure 17

A similar implementation with a different couple of variables follows.

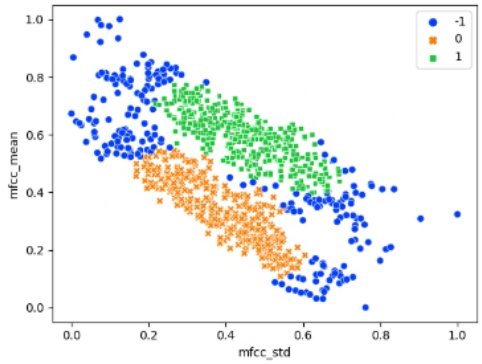

Figure 18: DBScan implementation on a two-couple dataset, with "song" as unique vocal channel, setting eps=0,065 and minpoints=50.

As the image below shows, most of the implementations on a dataset where "speech" instances were isolated returned just one single cluster with some noise points around, unfortunately conveying very little information about the variables explored.

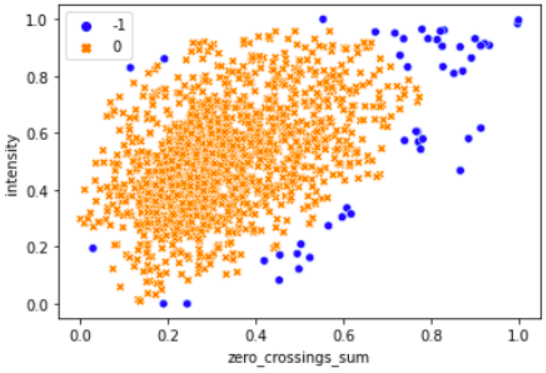

Figure 19: Example of one of the scatterplots obtained with the DBScan method where only one cluster was clearly visible.

Later, as already done when dealing with K-means and hierarchical clustering, we implemented the algorithm on different sub-datasets, each one of them belonging to the same emotion and involving two variables only.
For example, the sub-dataset with only "neutral" instances and with "length_ms" and "zero_crossings_sum" as attributes returned what follows.

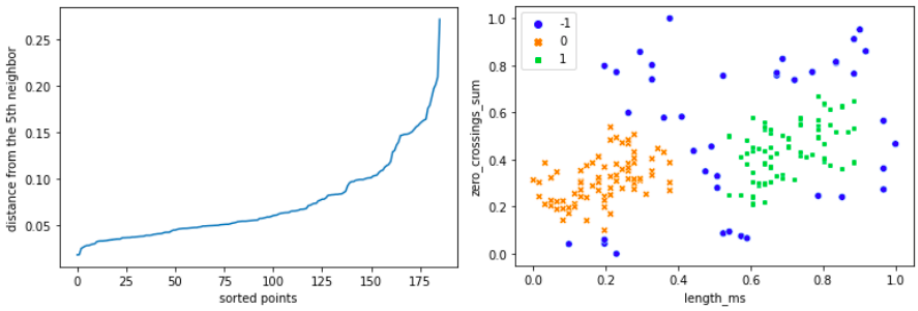

Figure 20: On the left: distance plot highlighting an ideal value of Eps around 0,07. On the right: scatterplot obtained applying DBScan with parameters based on the previous distance plot.

4. Decision trees

As already done with the implementation of clustering algorithms, we thought about partitioning the dataset before implementing the Decision Trees algorithm. Basically, this would have meant to "artificially" build the first split by putting in the root node of each resulting tree one of the values observed for the particular attribute splitting the original dataset. This method did not seem meaningful with an algorithm whose purpose is already exactly to split the records on attributes that optimize certain criterion. Therefore Decision trees were implemented on the whole dataset and its goal was to correctly predict emotions based on the other attributes.
The algorithm was implemented by tuning some parameters in order to get the most readable and useful output. As for measures of impurity, we noticed that entropy was not that good, because it mainly returned too large values for the accuracy score, meaning that the result was overfitting the training set. Therefore, we preferred using the Gini index only to measure the quality of a split.
Unfortunately, the outputs at this point were quite unreadable, because the decision trees were extremely complex and thus very wide. This convinced us to tune the max_depth and ccp_alpha parameters, but the second one returned better outputs. For this reason, only ccp_alpha was used and pruning could be performed, resulting in a loss of complexity (and then accuracy), but at least conveying useful results. We looped over this parameter, starting from ccp_alpha=0 to ccp_alpha=0.003, with step size=0.0002. Of course, we found ccp_alpha and the resulting accuracy score to be inversely proportional. This can indeed be observed in the image that follows, whose purpose is just to provide insights about how the pruning parameter can affect the final outcome of the Decision Tree algorithm.

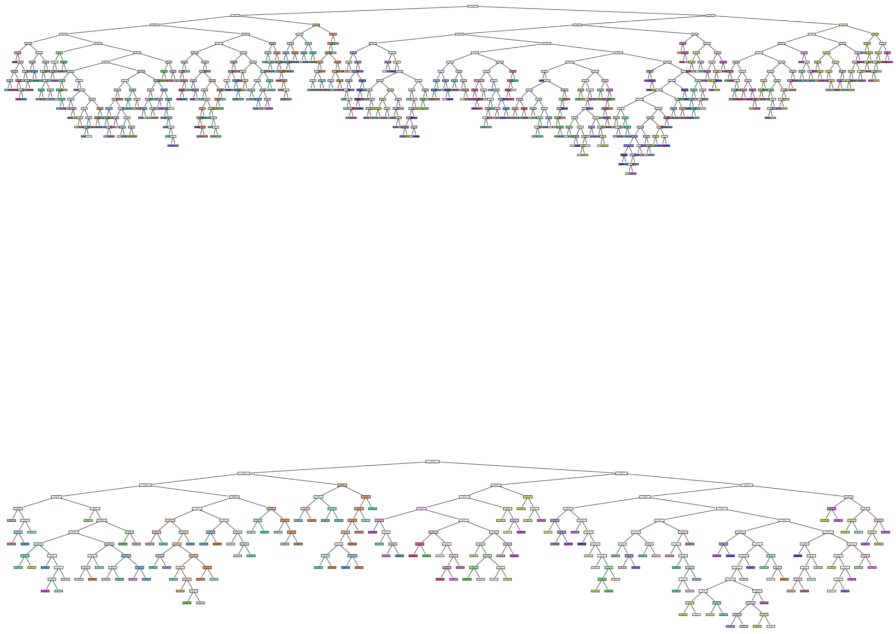

Figure 21: Above: decision tree whose parameter ccp_alpha was set to 0,001, with a resulting accuracy score on training set equal to 0,91. Below: decision tree whose parameter ccp_alpha was set to 0,002, with a resulting accuracy score on training set equal to 0,667.

Since decision trees were of course too wide to be readable, the relationship between the pruning parameter and the related accuracy score can be better explained by confusion matrices. We computed them for every value assigned to ccp_alpha. The images below show how the confusion matrix changes with respect to increasing values assigned to the ccp_alpha parameter. Again, the purpose of these images is to show how the accuracy decreases as the pruning parameter ccp_alpha increases, thus with the green along the diagonal of the matrix getting lighter and lighter.

Figure 22: Outcome on the training set. Values given to ccp_alpha are 0.0008, 0.001, 0.0012, 0.0014, 0.0016, 0.0018, 0.002. Confusion matrices with ccp_alpha smaller than 0.0008 are not included, because they are basically the same as the first one in this image. The accuracy scores are 0.949, 0.910, 0.855, 0.807, 0.737, 0.690 and 0.667 respectively.

Moving to the test set, when the ccp_alpha value is equal to 0.001, the accuracy score is 0.374, thus much lower than the one obtained in the training set. The diagonal of the matrix below, indeed, is not that "chromatically" evident as those in the previous matrices.

Figure 23: Outcome on the test set, with ccp_alpha value equal to 0.001 and accuracy score equal to 0.374. This value to the ccp_alpha parameter was given because it seemed the best to return a good accuracy without overfitting too much.

The relationship between the accuracy score and both ccp_alpha first and max_depth then is summed up by the following plots.

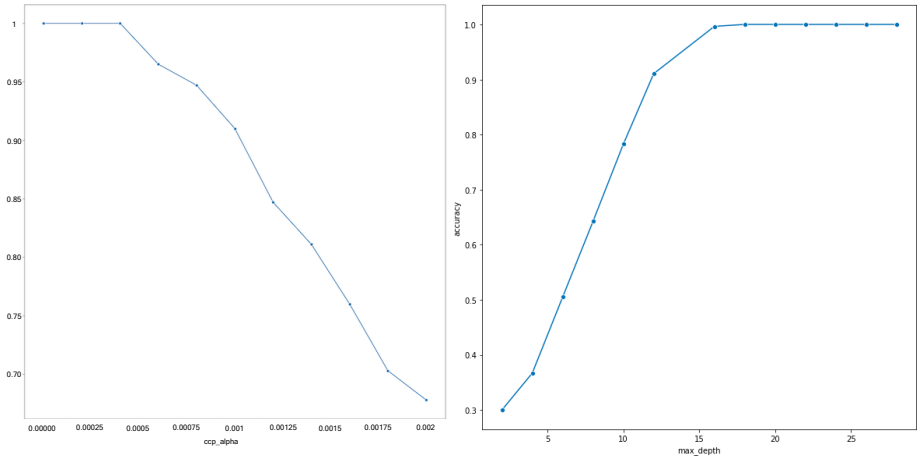

Figure 24: On the left: accuracy decreases when ccp_alpha increases in the training set. On the right: when the value assigned to max_depth increases, accuracy increases too.

Then cross-validation was implemented with the same range of ccp_alpha values as before and the mean of the resulting accuracy scores on the test set for each of the values given to the parameter can be observed in the table below.

Accuracy score	0.3901	0.3889	0.3878	0.3818	0.3884	0.3813	0.3931	0.3978	0.3984	0.3949	0.3984	0.3931	0.3789	0.3706	0.3658
ccp_alpha	0	0,0002	0,0004	0,0006	0,0008	0,001	0,0012	0,0014	0,0016	0,0018	0,002	0,0022	0,0024	0,0026	0,0028

Figure 25: Comparison between ccp_alpha parameter and the resulting mean of the accuracy score obtained with cross-validation (cv=5).

5. Naive Bayes

As for Naive Bayes classifiers, the Gaussian alternative was implemented, so that it could deal with continuous variables as well.
The table below summarizes how this model performed on our data, based on the difference between its predictions and the real values in the testing set.

	Precision	Recall	F1-score	Support
ANGRY	0.60	0.60	0.60	109
CALM	0.66	0.19	0.30	110
DISGUST	0.25	0.37	0.30	57
FEARFUL	0.64	0.19	0.29	112
HAPPY	0.31	0.28	0.30	111
NEUTRAL	0.22	0.71	0.33	56
SAD	0.26	0.10	0.14	112
SURPRISED	0.30	0.74	0.42	57
ACCURACY			0.35	724
MACRO AVG	0.40	0.40	0.33	724
WEIGHTED AVG	0.44	0.35	0.33	724

Figure 26: Classification report assessing quality of the Gaussian Naive Bayes model previously implemented.

As the table shows, the model did not perform that well overall, except for same cases. For example, the "angry" emotion was predicted correctly for 60% of its instances. "Calm" and "fearful" got quite good precision scores, but a much lower recall score, meaning that out of all the instances presenting these emotions, the model predicted them correctly only for 19% of the times. "Neutral" and "surprised" instances, instead, were predicted correctly 71% and 74% of the times, respectively. However, their small precision scores underlie the fact that the model also predicted these two emotions for instances which had nothing to do with them. The overall accuracy score of 0.35 is a little bit lower than those obtained when implementing the Decision Tree algorithm.
The Receiver Operator Characteristic curves resulting from this model can be observed below.

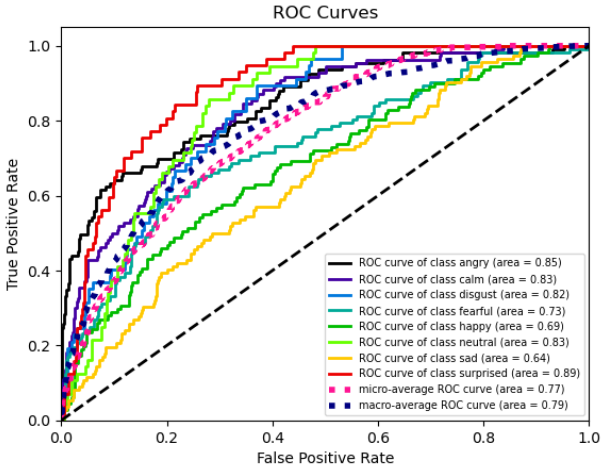

Figure 27: ROC curves from Guassian Naive Bayes implementation.

The ROC curves in the image above confirm what already seen in the previous table: some emotions (such as "angry" and "calm") were predicted much better than others ("sad" is the emotion with the lowest score) , with relevant differences in the resulting trade-offs between true positive rate and false positive rate and then in the area under the corresponding curve.

6. KNN

Before implementing this algorithm, it was necessary to standardize numerical data, as well as applying OneHotEncode to categorical data. In this way, it was possible to observe the results of KNN considering the whole dataset.
At first, the parameter tuning was based on the results from an iteration where the regular "fit" method was used. Thus, each combination for the parameters "metric" (cityblock or Euclidean) and "weights" (uniform or distance) was observed, while the number of neighbors was set equal to each value in the interval between 1 and half the rows of the dataset.
In order to better validate the result, we implemented the grid search method, that basically does the same thing as the procedure described above but uses cross-validation to get the most precise accuracy score. Cross-validation implies more iterations for each combination of the KNN's parameters so the computation of the grid search took longer. The results obtained were quite similar to those previously observed: they follow the same curve but are slightly lower, that is because the grid search gives a more accurate estimate on how the choice of the parameters would perform in classification.
The performance of KNN classifier is better when the chosen metric is cityblock, k is around 75 and the weights are uniform, meaning that all points in each neighborhood are weighted equally. In this case, indeed the accuracy score on the testing set is around 0.37.

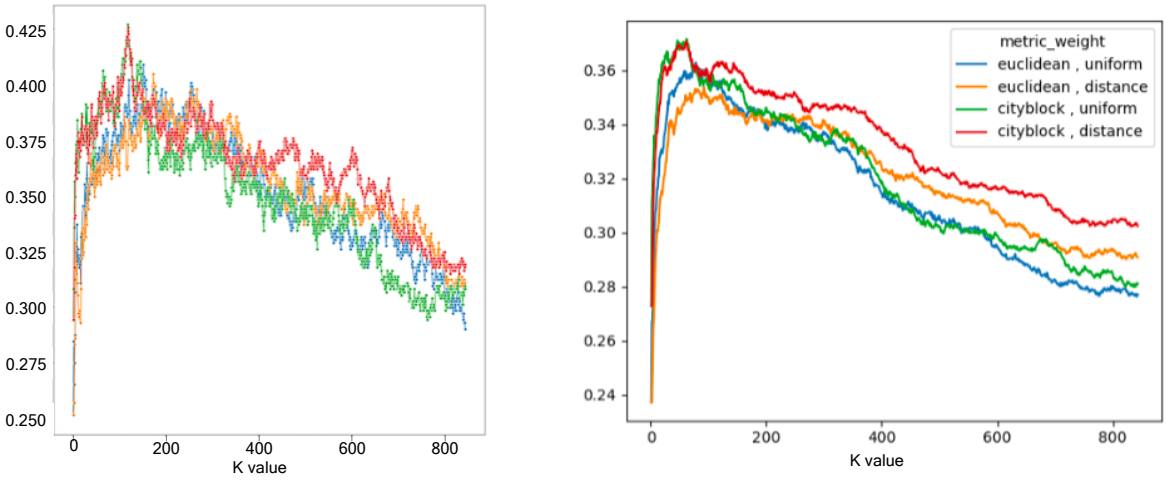

Figure 28: KNN implementation on the whole dataset. On the left: parameters are tuned via iteration. On the right: parameters are tuned via grid search.

7. Pattern mining

Since the pattern mining algorithm only works with categorical data, the first step here was to discretize every numerical variable. The function "qcut" was useful for this purpose: the parameter regarding the number of bins in which the data would be divided up was tuned.
When applying the apriori algorithm, its parameters were tuned as well. For example, the following images show how much the number of all frequent itemsets varies with respect to different support thresholds, with different numbers of bins created when discretizing numerical data (represented by different lines in the plots) and for different values given to the parameter "zmin". This parameter indicates the minimum number of items that an itemset should have in order to be considered.

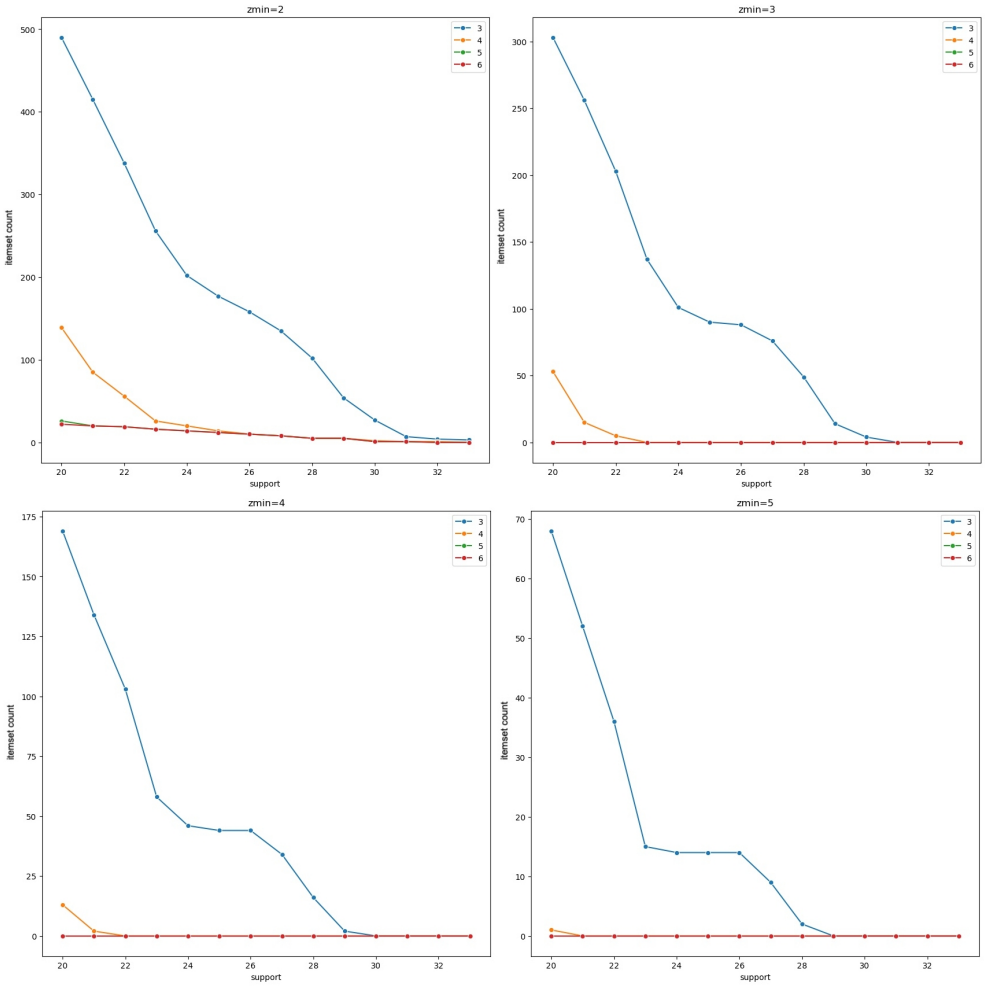

Figure 29: Comparison between the amount of all frequent itemsets and support thresholds, with respect to different values for the parameter zmin (2, 3, 4, 5) and considering 3, 4, 5 or 6 bins for discretization.

These four plots highlight that when the support threshold increases, it is harder to find itemsets, especially when these should include at least four items. Furthermore, as you can see, the difference between 5 and 6 bins can only be noticed when zmin=2 and the support threshold is around 20.
The same analysis was pursued considering closed frequent itemsets and maximal frequent itemsets as well. The outcome can be observed below.

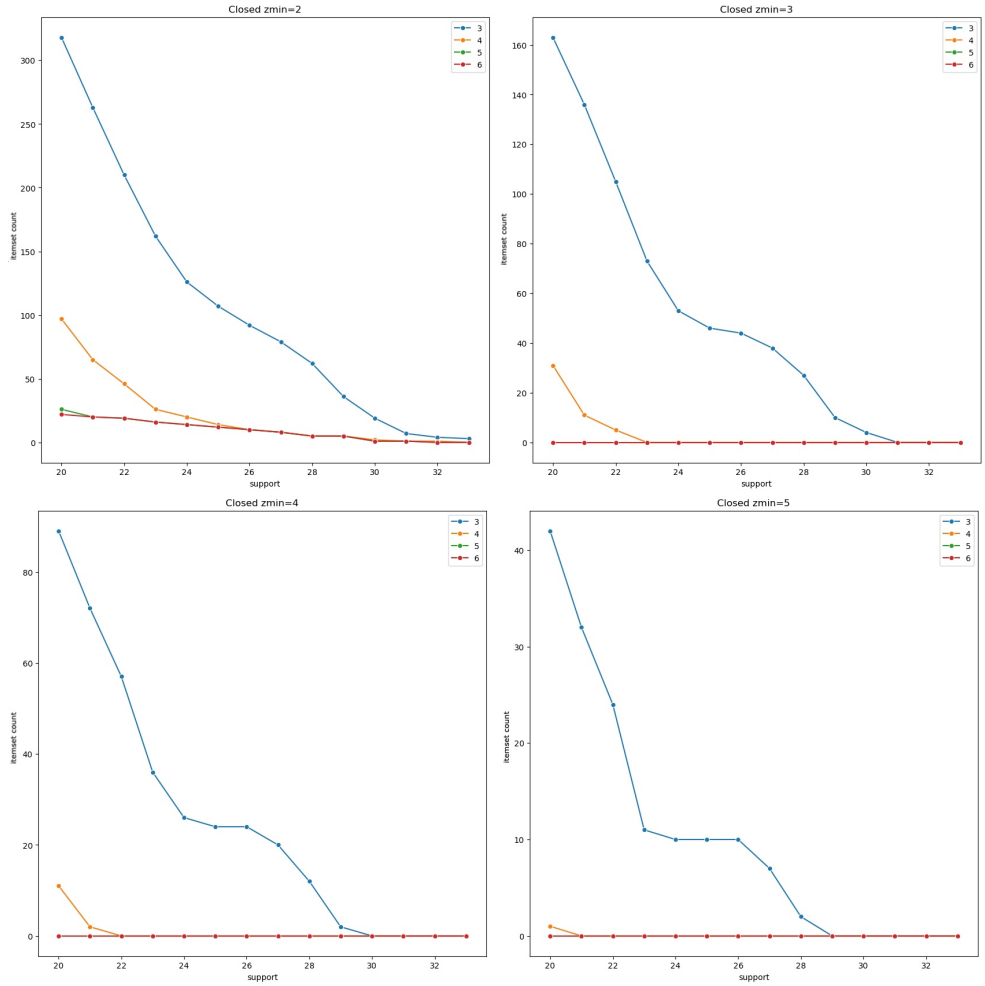

Figure 30: Comparison between the amount of all closed frequent itemsets and support thresholds, with respect to different values for the parameter zmin (again, 2, 3, 4, 5) and considering 3, 4, 5 or 6 bins for discretization.

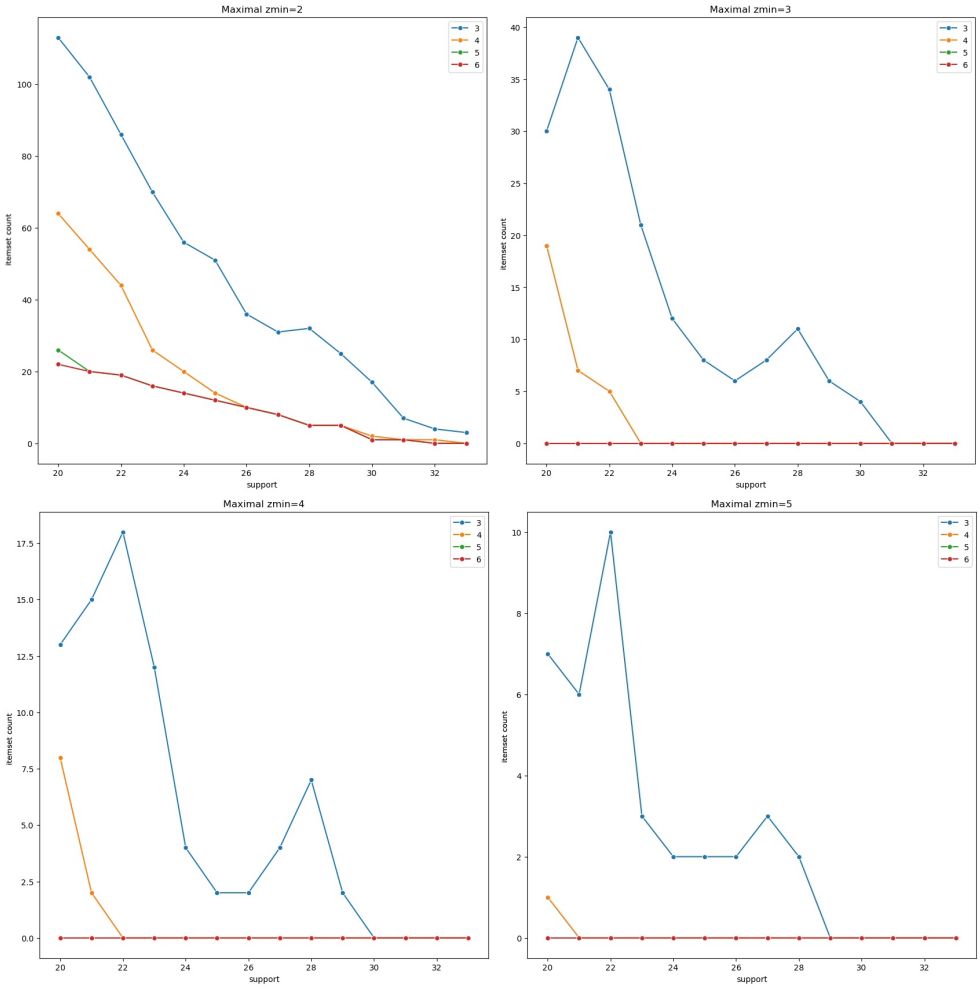

Figure 31: Comparison between the amount of all maximal frequent itemsets and support thresholds, with respect to different values for the parameter zmin (2, 3, 4, 5) and considering 3, 4, 5 or 6 bins for discretization.

From the images above it is possible to notice that, while closed frequent itemsets roughly follow the same trend as frequent itemsets, the amount of maximal frequent itemsets is more complicated to predict. Indeed, the derivative of the lines representing maximal frequent itemsets is not negative in every point. This means that frequent itemsets for a certain level of support can have a number of frequent immediate supersets which is not related to the support threshold itself. Thus, there can be more maximal frequent itemsets at higher levels of support. For frequent and closed frequent itemsets, instead, the anti-monotone property of support still holds. Anyway, after a certain support threshold is reached, it is unlikely to find relevant itemsets.
The previous images also show that the number of bins created when discretizing numerical data is inversely proportional to the number of itemsets found. Indeed, when the bins are 6 and the itemsets should have at least three elements, no result has been found (the red line in the images above is flat).
Then we looked for rules. In order to get relevant results, we chose a support level of 24% and discretized numerical variables with 3 bins. The figures above helped us tune these paramaters, since it seemed to provide the most "middle ground" results.
The first part of our analysis was mainly focused on finding out the total number of rules for each combination of confidence, given 24 as the support level and zmin between 3 and 6, since we are not interested in rules generated by couples. The outcome can be observed below.

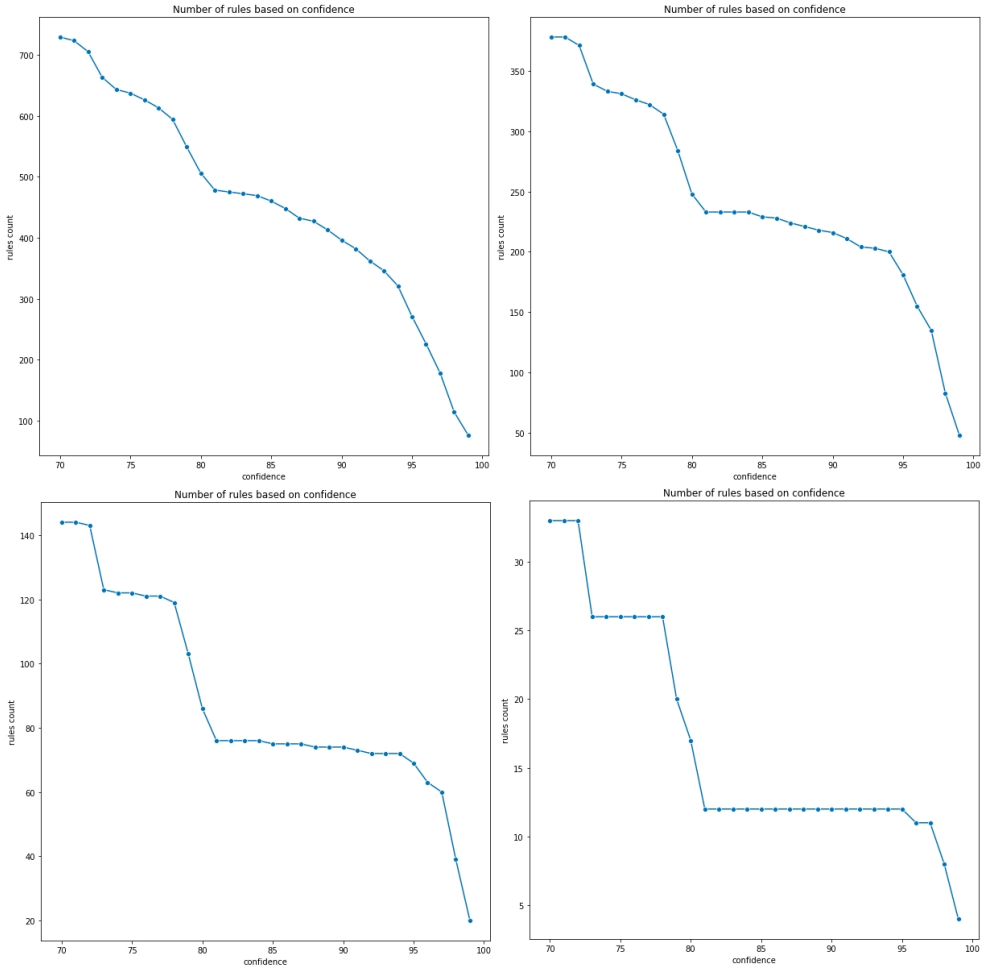

Figure 32: Number of rules found based on different levels of confidence, with respect to a minimum number of items per itemset equal to 3, 4, 5 and 6 respectively.

As the images show, when the minimum number of items per itemset increases, the rules found are fewer and different confidence levels lead to smaller variations on the number of total rules, even though there are two thresholds around 80 and 95 where the amount of rules drops dramatically.
Among all these rules, the maximum lift observed was around 3 for each confidence threshold. However, our specific goal was to find rules with an emotion as a consequent. Still holding our support level at a value of 24, the maximum level of confidence allowing us to find some rules of this kind is 37. The output in this case consists in three rules with a lift value around 2.4, meaning that they are quite interesting. All of these three rules have the emotion "angry" as a consequent. Further details can be observed in the table below.

Consequent	Antecedent	abs_support	%_support	confidence	lift
angry	((-717.307, -461.489]_mfcc_min, (0.171, 0.999]_max, (-33.922, -18.016]_intensity, (0.0201, 0.126]_std, (84.092, 128.391]_mfcc_std, (-1.0, -0.155]_min)	245	10.153336	0.371212	2.46081
angry	((-717.307, -461.489]_mfcc_min, (0.171, 0.999]_max, (-33.922, -18.016]_intensity, (84.092, 128.391]_mfcc_std, (-1.0, -0.155]_min)	245	10.153336	0.371212	2.46081
angry	((-717.307, -461.489]_mfcc_min, (0.171, 0.999]_max, (0.0201, 0.126]_std, (84.092, 128.391]_mfcc_std, (-1.0, -0.155]_min)	245	10.153336	0.371212	2.46081

Figure 33: Rules with an emotion as a consequent with the highest lift value.

It is necessary to decrease the confidence level up to 33 in order to find many other rules with another emotion as a

consequent: calm, with a lift value around 2,1.

8. Conclusion

To sum up, it might be useful to compare all the outcomes provided by the algorithms that this report has covered. First of all, clustering algorithms were implemented. The main issue in this case was the fact that it was necessary to isolate couples of variables from the initial dataset in order to get relevant results. This might help to understand possible dependencies between attributes, but not that much when the goal is to find different patterns for each of the eight emotions.
Talking about supervised learning, here the results were more interesting: it was possible to operate on the whole dataset and still get some useful insights based on different emotions. The graph below compares performances of the three classifiers that were used for this purpose.

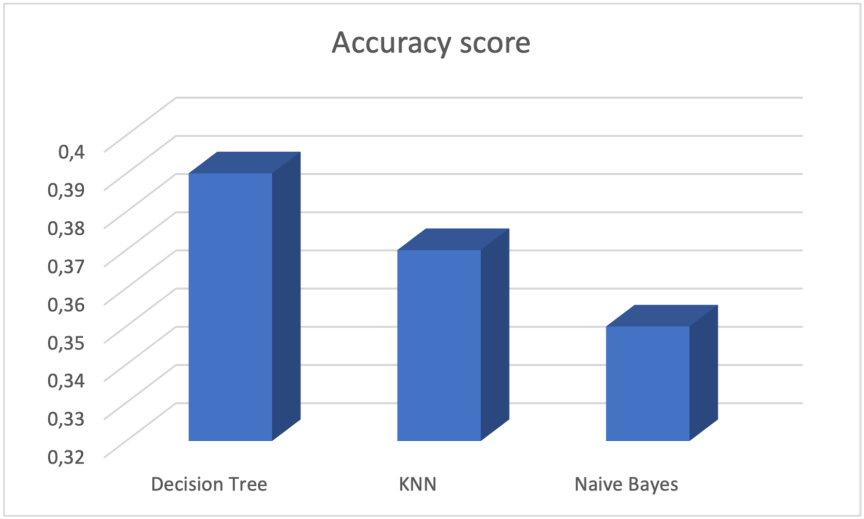

Figure 34: Performance comparison among Decision Tree, KNN and Naive Bayes algorithms.

Decision Tree algorithms were those returning the highest accuracy score, followed by KNN and Naive Bayes. This result is not that surprising. First of all, it was quite predictable that Naive Bayes could not be better than the others, since there was no parameter tuning and therefore grid search with stratified k-fold cross validation could not be implemented here. Furthermore, decision trees might have returned better results than KNN because of the curse of dimensionality that affects KNN outputs when dealing with high-dimensional data like in this case.
Finally, pattern mining was implemented. However, the most interesting results provided by this algorithm did not involve emotions as we would have liked.