Data Science Lab: Process and methods
Politecnico di Torino

Project report
Student ID: s270005

1. Data exploration

The dataset provided consists in 30281 *Librivox* recordings divided into training (24449) and evaluation (5832). The tracks are collected from 10 different speakers (labeled from *a* to *j*) and each record is associated with a unique ID.
The audio records are all sampled at 24kHz (checked) and they are composed by an **average of 12000 sample**:

TRAINING											
Len	12000	11999	11879	11159	9671	8086	7724	6166	2759	2471	2039
N°	505	23935	1	1	1	1	1	1	1	1	1

EVALUATION			
Len	11999	11159	6694
N°	5830	1	1

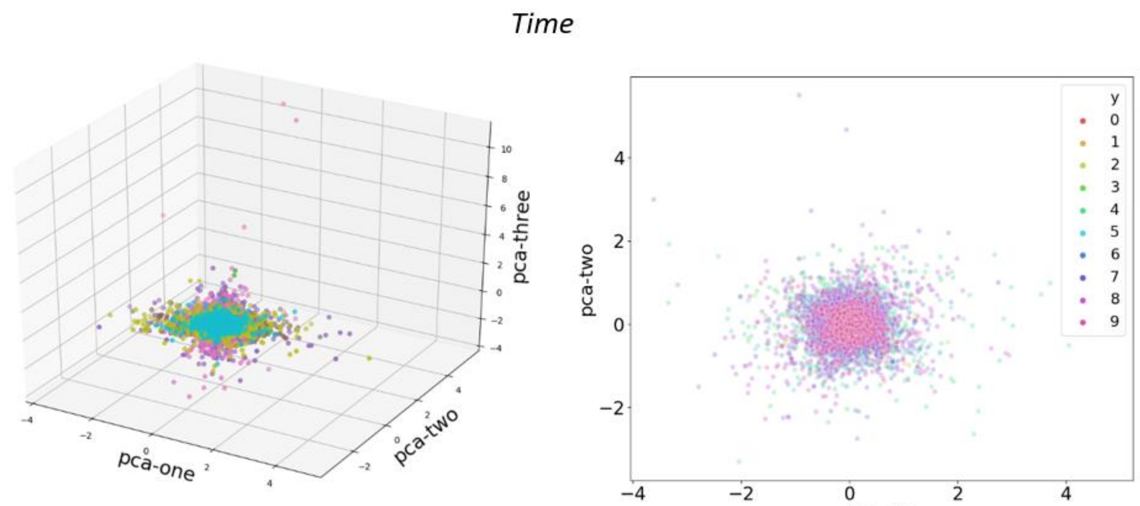
To evaluate all the tracks properly and to do further analysis, I decide to **pad** each record with the track's mean due to have all lengths=12000.

The number of records for each person instead is distributed as below with **around 2500 sample** for each person**.**

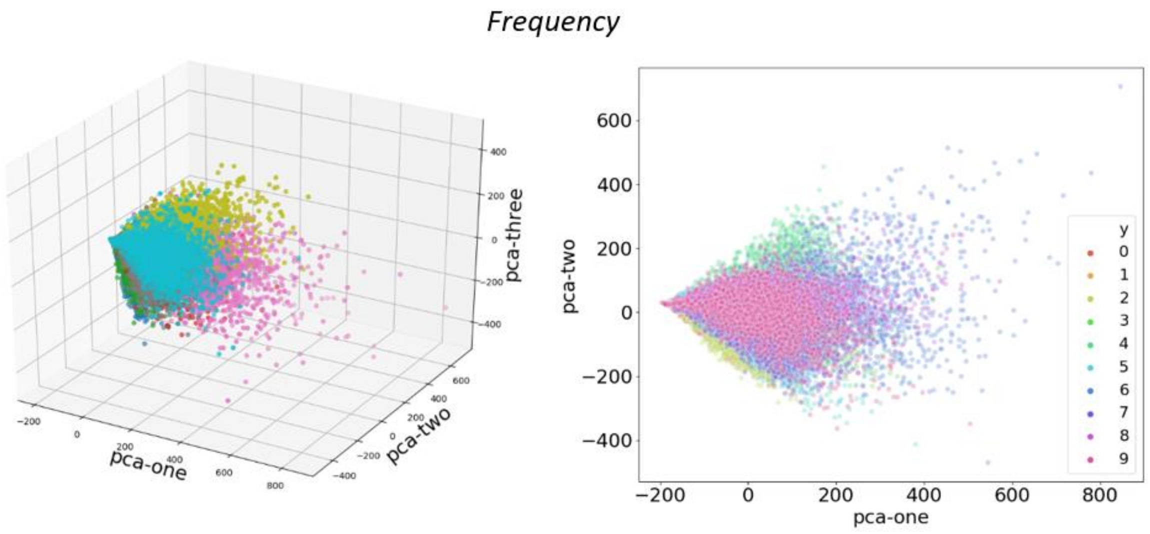
A	B	C	D	E	F	G	H	I	J
2403	2478	2481	2431	2439	2401	2475	2483	2400	2458

About the records, I **check the std** of each one and I find out that 37 tracks have it equal to 0, so I decide to delete them since they don't provide any important information. This consideration is enforced by the check of their spectrogram that are empty. After this selection I have 24412 records.

For the distribution of data, I decide to look for the time and frequency domain. To have a visual representation, I use **PCA** (n_components=3) on the data for time and on the FFT (taken in absolute value and computed with rfft of numpy) for frequency.



On left the 3D plot, on right the 2D plot



On left the 3D plot, on right the 2D plot

Figure 1.

In the time domain the representation is more confused, and all the points overlap each other; instead for frequency we see, especially from the 3D plot, how the points of the same person are more grouped together and better divided. So, I proceed with frequency domain.

Another analysis is about **outliers**. I take the tracks and, by taking 800 (step) values at time, I compute for them some statistics (Mean/std/ecc) with in total 15 windows.
Then, for each person, I average those values and I check if the mean of each window of every record of that person is inside the averaged_mean ± averaged_std (outers not included) of the corresponding window of the average of that person. At the end I have 24397 records.

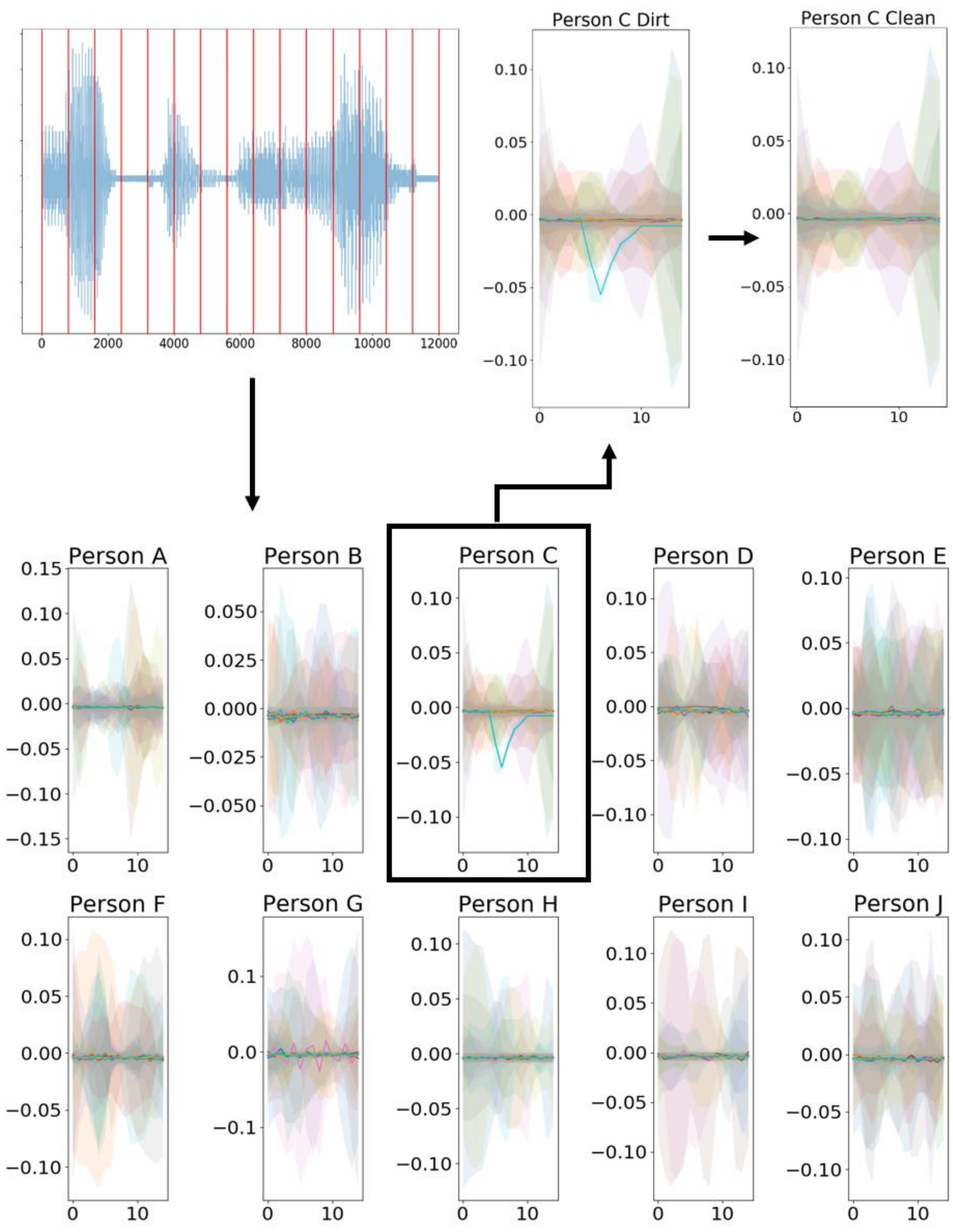


Figure 2.

I have done a further consideration on the records' spectrogram, there are many of them in the development that show low values of decibel on the corresponding

frequencies of human voice. However, removing them or trying to remove the noise is way useless since also many tracks in the evaluation present this behavior.
So, I decide to keep those values as a reference of **noise** connected to a person, in this way it is a further feature to consider next.

About implementation: labels where converted to numbers (0:9) and the loading of files has been done through *Librosa*[1].

2. Preprocessing

The **feature extraction** from the recordings is done by using the functions provided by the *Librosa*[1] library. As described before, all values are in the Frequency domain.
About the computation, all these features are compute by **averaging on the rows**, this concept is really important since we keep information only about the frequencies and not about time, that could have yielded to a different task.
Where is present the *Power_to_db* function, it is applied on the mean vector described before (for STFT on the squared mean vector).

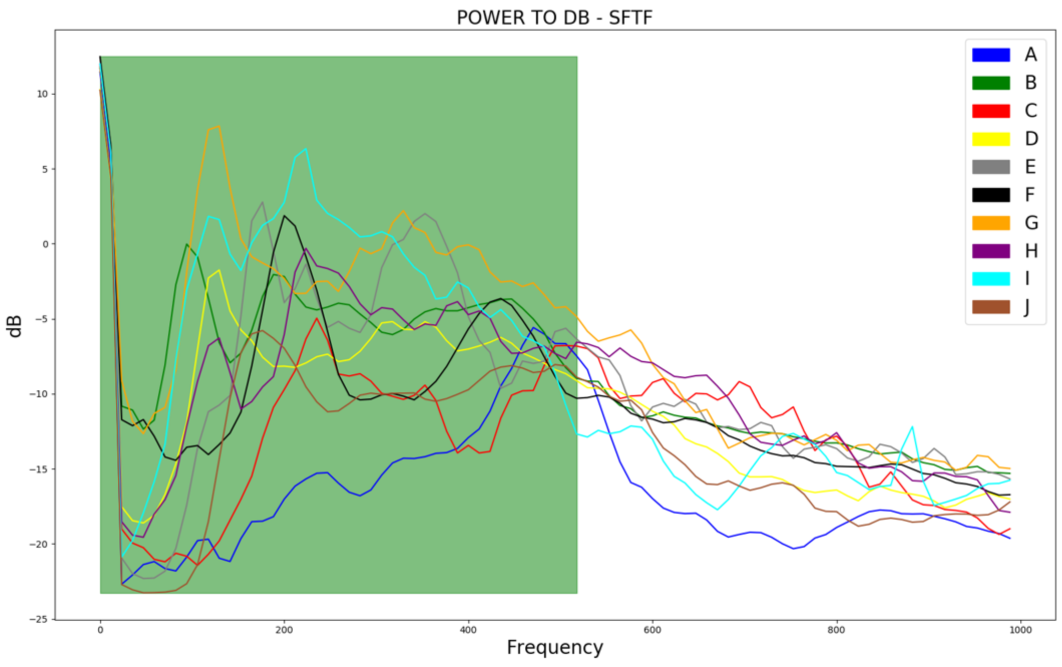
Features extracted for each record:

- **MFCC**, by taking 40 of them. The dimensionality is (24397,40).

- The **Spectral_Contrast** of the absolute of the **STFT**, that provides the energy values for each frequency bands. Moreover, high contrast values generally correspond to clear, narrow-band signals, while low contrast values correspond to broad-band noise.
  The dimensionality is (24397,7).

- I evaluate the **Spectral_Flatness**, which quantify how much noise-like is a sound, and it is important since the audios are really noisy, as it is possible to see from the spectrograms.
  The dimensionality is (24397,1).

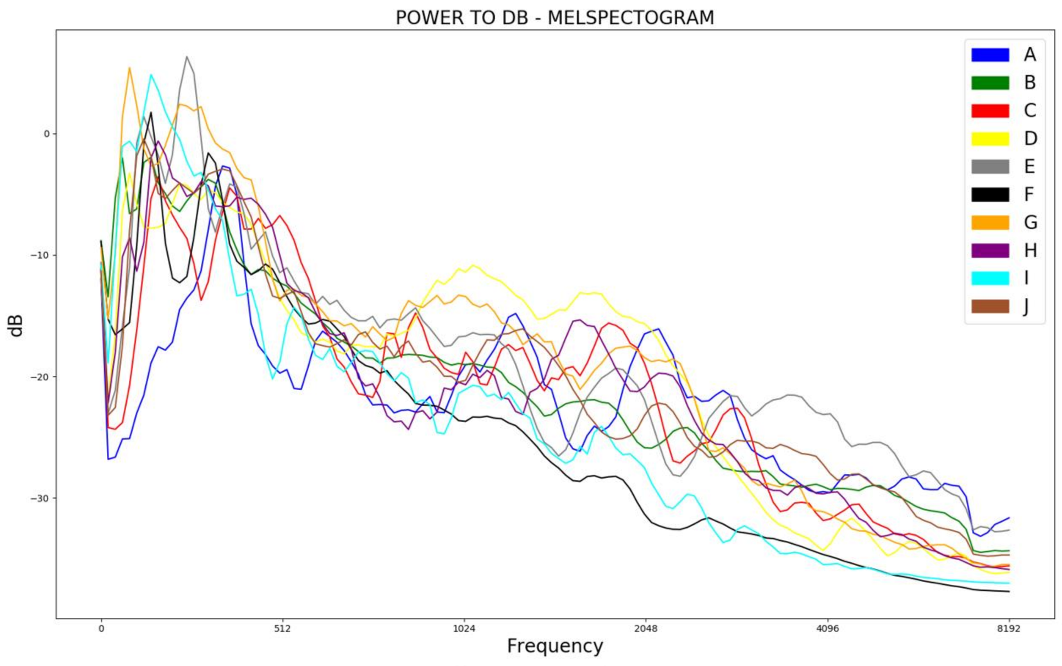
- **Power_to_db** of the absolute of the **STFT**. They provide a relationship between decibels and frequencies, in particular I decide to focus on the values between the range 0-512 Hz (range of index 0:44) since they include the frequencies of the human voice and some higher picks that are present in some combinations of vowels and consonant. For this consideration I discard higher values inside the range of 1kHz since are rarer in human voice and in general other values, considered as noise.
  The dimensionality is (24397,44).



Plot average for each person - in green frequencies between 0 and 512 Hz

Figure 3.

- **Power_to_db** of the **Melspectogram**. About them I take all values (now are only 128 and not 1025 as above, so there will be less issues about the dimensionality), so this time I also include information about noise(high frequencies), that in general is hugely present in most of the tracks.
  The dimensionality is (24397,128).



Plot average each person

Figure 4.

- I consider the **RMS**, a value of the energy of the record. The dimensionality is (24397,1).

- Finally, the **Chroma_CENS**, that are about the energy of in each Pitch Class. I decide to compute them on the records passed through an harmonic effect in order to take into consideration only what concern the human voice[2]. Between the different versions I choose the CENS since it provides stabler value over time (Figure 5), which make a huge difference when I average them.
  The dimensionality is (24397,12).

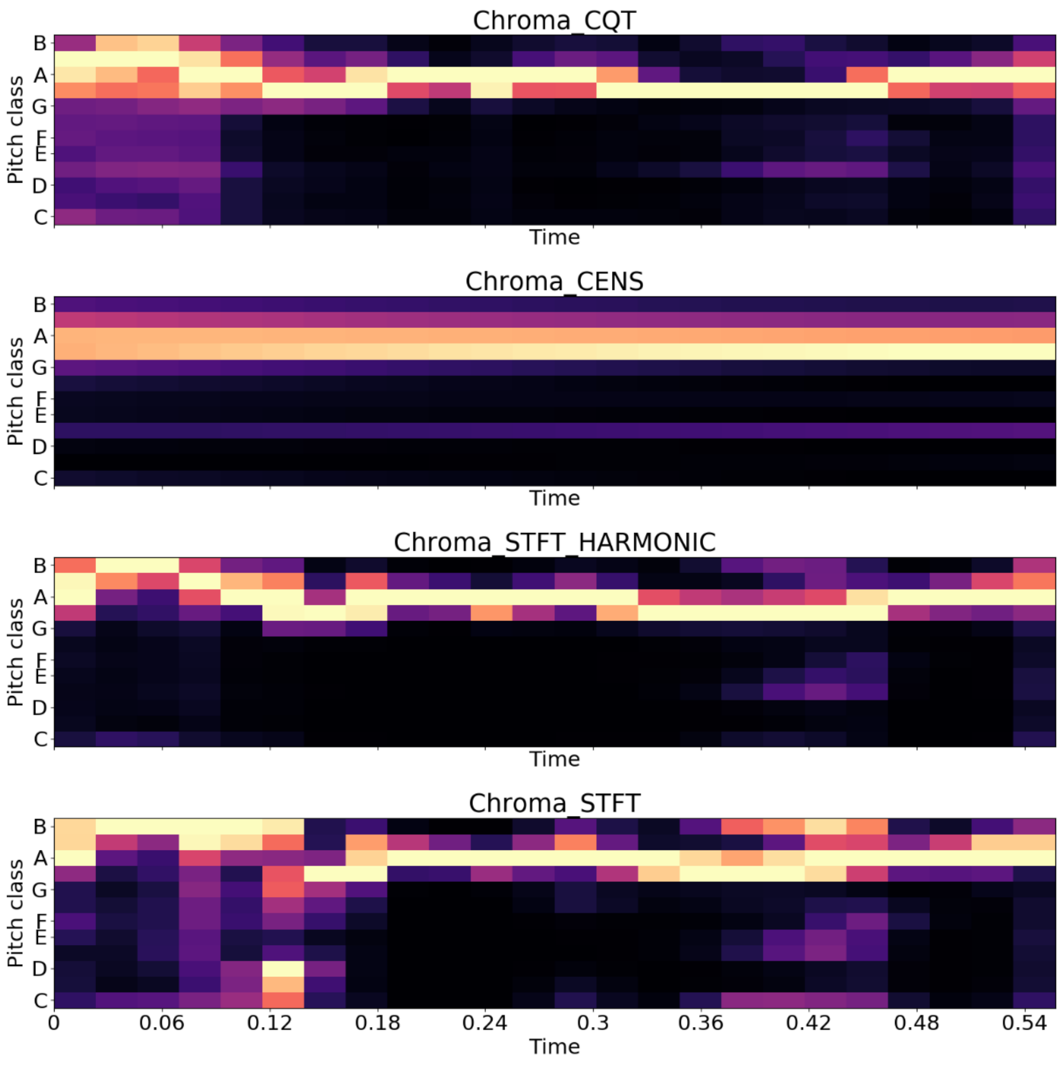


Figure 5.

All these features are concatenated horizontally with a final dimension of (24398,233).
I try other features provided by *Librosa*[1], but they are discarded since not suitable for this task.

3. Algorithm choice

In order to choose the appropriate model for this kind of task different classifiers have been evaluated, by starting on the consideration of the **distribution of the features extracted.**
To make this analysis, I plot the distribution after the application of the **t-SNE** (n_components=2, perplexity=20) on the features extracted before**.**

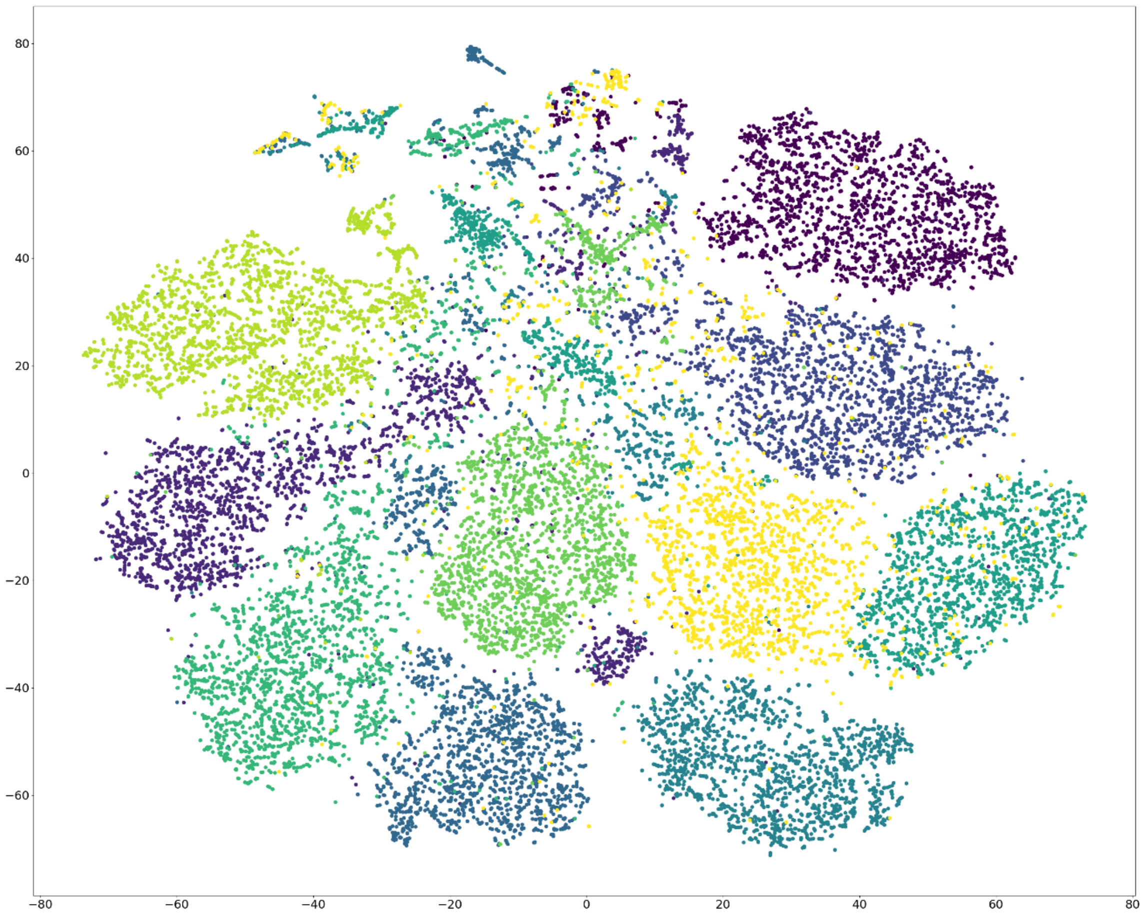


Figure 6.

As we can see the data are quite well divided, moreover the division margins are quite linear so my presumption is that algorithms like **RandomForest** and **SVM** with **linear** kernel will quite work well, but with the first one with some issues due to the fact that the margins are not orthogonal. Instead **KNN** could provide good scores, however it could have same issues in the fuzzier regions and in general with the curse of dimensionality since the big number of features. Overall an algorithm that change the dimensionality of the data, like **SVM** with **rbf** kernel, or that tries to make **polynomial** margins can help in the classification task. A last attempt is done with a **Naïve Bayes**

classifier, however I suppose to obtain low performance since it is based on the assumption that features are statistically independent.
I test these different algorithms in order to confirm my assumptions and all of them are evaluated inside a **cross-validated gridsearch** done with **3 folds** and with a division between **train and test of 70:30** (17077 sample for train and 7320 for test) done with a fixed random_state=42, for reproducibility, and with the stratify parameter set equal to the labels due to keep their distribution inside the split.
An important step that I do is to standardize the train and the test set with **StandardScaler** before to pass them to the model; instead I don't considered required the application of an algorithm of dimensionality reduction since the number of feature are reasonable and there are no issues about the computation time. For completeness I try in any case a PCA (n_components:[50,100,150]) but I always see less accurate results (respectively for the previous parameter: 0.9789, 0.9886, 0.9863 with SVM-rbf kernel and hyperparameters in the table below).
The scores shown in the table below are the ones computed on the test set and represent the **f1_score** with macro average.

Algorithm	**Parameters**	**Best Params**	**Score**
RandomForest	n_estimators=[100,200,500,700,1000]	1000	0.9662

KNN	k=[1,3,5,7,10]
	weights=['distance','uniform']
	p=[1,2,3]
		k=3
		distance
		p=1
			0.9519

SVM-linear	C=[0.01,0.1,1,10,100]
	gamma=[0.001,0.01,0.1,1,10,100,1000]
		C=0.1
		gamma=0.001
			0.9842

SVM-poly	C=[0.01,0.1,1,10,100]
	gamma=[0.01,0.1,1,10,100]
		C=1
		gamma=0.01
			0.9812

SVM-rbf	C=[0.01,0.1,1,10,100]
	gamma=[0.001,0.01,0.1,1,10,100,1000]
		C=10
		gamma=0.001
			0.9901

Gaussian NB	default	-	0.7588

As it is possible to see from the previous table, my considerations are confirmed to be true and so for the last step I proceed with an **SVM with rfb kernel** classifier.

4. Tuning and validation

For the final model with a **SVM with rbf kernel** classifier I decide to apply a 3 folds grid search cross validation to finetune the following hyperparameters:

- *C*: 10, 20, 30, 40
- *Gamma*: 0.0001, 0.005, 0.001, 0.05, 0.01
- *Tol*: 1e-3, 1e-6
- *Class_weight*: None, Balanced

The holdout is the same cited in the previous point (30%), the metric used is f1_score with average macro as stated in the assignment and it is applied the same scaler on data as already mentioned in point 3. The features used are the same obtained in point 2.
As you can see, for the hyperparameters I change the interval from both *C* and *Gamma* since, from the verbose of the step in the Algorithm Choice, I have seen that combinations with too small or too big *C* and *Gamma* provided poor results. Another motivation to change scale of hyperparameters comes from their manual tuning.
I obtain the best results with ***C*=20**, ***Class_weight*=None**, ***Gamma*=0.005** and ***Tol*=1e-3** with a local score on the test set of 0.9907 and with a score on the leaderboard of **0.9710** for the evaluation set**.**

	Precision	**Recall**	**F1-score**	**Support**
A	0.99	0.99	0.99	721
B	0.99	0.99	0.99	742
C	1	0.99	0.99	742
D	0.99	0.99	0.99	724
E	1	0.99	0.99	728
F	0.98	1	0.99	720
G	0.99	0.99	0.99	743
H	0.99	1	1	745
I	1	1	1	720
J	0.98	0.98	0.98	735
Accuracy			0.99	7320
Macro avg	0.99	0.99	0.99	7320
Weighted avg	0.99	0.99	0.99	7320

As it is shown in the table, the general prediction are very good, however there are some issues in particular with F, where the algorithm tends to assign this label too many times, and then there is J that have the same issue of F and it is also kind of "frequent" to be misclassified with another label.

Overall the metrics used for the score can be considered fair as we can see from the support column, nevertheless, as you can see from the list of parameters before, I try to balance the classes with the hyperparameter class_weight=balanced but without any significant results.
As stated in point 2, I also finetuned the various features provided by *Librosa*[1], like spectral_centroid, spectral_rolloff, tonnetz, zero_crossing_rate, ecc.., but with a lot of analysis and information found on documentations and on internet I end up by choosing the features reported in "Preprocesing".

5. References

[1] LIbrosa at: https://librosa.org/doc/latest/index.html

[2] Wikipedia at: https://en.wikipedia.org/wiki/Voice_frequency