Data Science Lab: Process and methods
Politecnico di Torino

Project report
Student ID: s280107

1 Data exploration (max. 400 words)

The dataset used has been extracted from LibriVox [1], a website that provides public audiobooks. 30,281 recordings, sampled at 24kHz, have been extracted from 10 different people (females and males). Then, the whole dataset has been divided in two different sets: development set (24449 samples) and evaluation set. For the development set the labels were given, while not for the evaluation part.
Studying the development dataset we can see that the classes are well balanced, having about 2400 samples per class. The samples are very similar in length (0.5 seconds), with only a few outliers on the whole dataset.
The distribution of the two datasets is very similar, this means that the recordings were captured in the same conditions and the performance of the classifier on the test set should be very similar to the train set. This can be proved by looking at the standard deviation of the signals as shown in Figure 1.

The biggest problem was that many samples were only noise or just silence, making the classification task very difficult. Given the fact that the development and evaluation sets were taken from the same distribution, these outliers not only were in the development set, but also in the evaluation set. Using the index signal-to-noise ratio SNR (given by the mean of the signal divided by the standard deviation) we can clearly see that there is a great amount of noise in many signals (there is a peak at 1), and this is for both development and evaluation sets as we can see in Figure

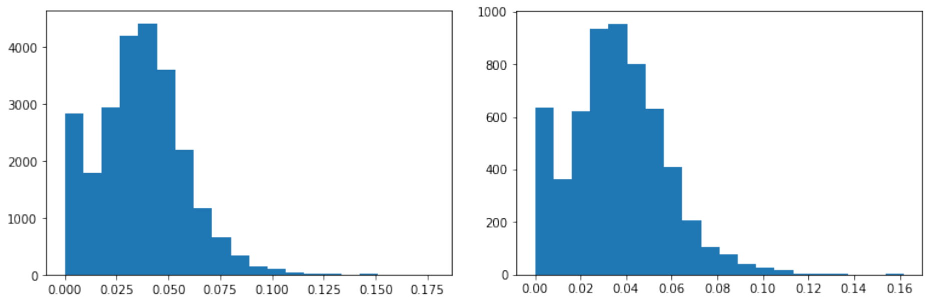

Figure 1: Standard deviation of signals of the development set (left) and evaluation set (right)

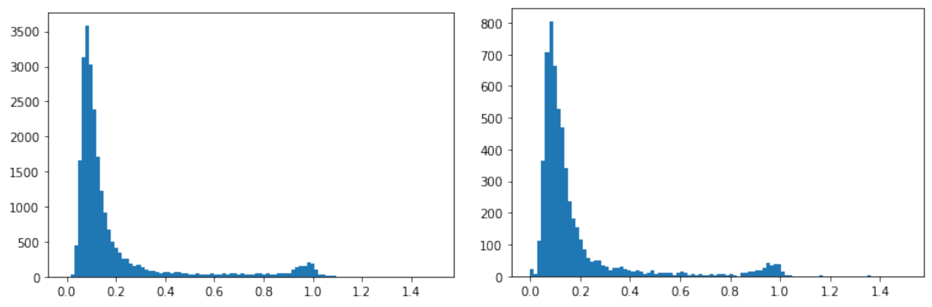

Figure 2: SNR for development set (left) and evaluation set (right)

2. This is certainly going to affect the performance of the classifier negatively.

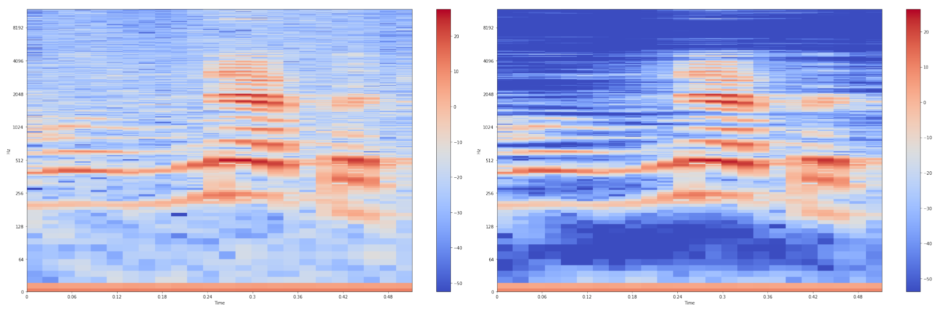

Figure 3: Spectogram of a recording before (left) and after (right) denoising

2 Preprocessing (max. 400 words)

The most challenging part of this project was the preprocessing of the audio signals.

2.1 Removing short signals

The first thing I did was to delete some outliers, for example signals shorter than a certain threshold, this was done by checking the length of the arrays.

2.2 Remove background noise

After removing short signals, I realized that in many clips there was a noisy background, probably caused by the microphone or not a perfect recording, and I decided to remove it. I chose an outlier identified before and used it in order to remove that noise sample from other signals. This was done using a Python library called *noisereduce* [3] that takes as input an audio signal and a noise signal and performs a filter based on the noise given.
The noise signal was a signal found in the ones shorter than the rest of the recordings and sounds like this: (to hear audio please use Adobe Acrobat Reader)

noise

The effect this preprocess has on recordings can be heard in this signal sampled from the dataset:

before de-noising

after de-noising

The difference can also be seen on the spectogram of the audio as we can see in Figure 3. Frequencies which are not useful and that are background noise are deleted, this improved the performance of the classifier.

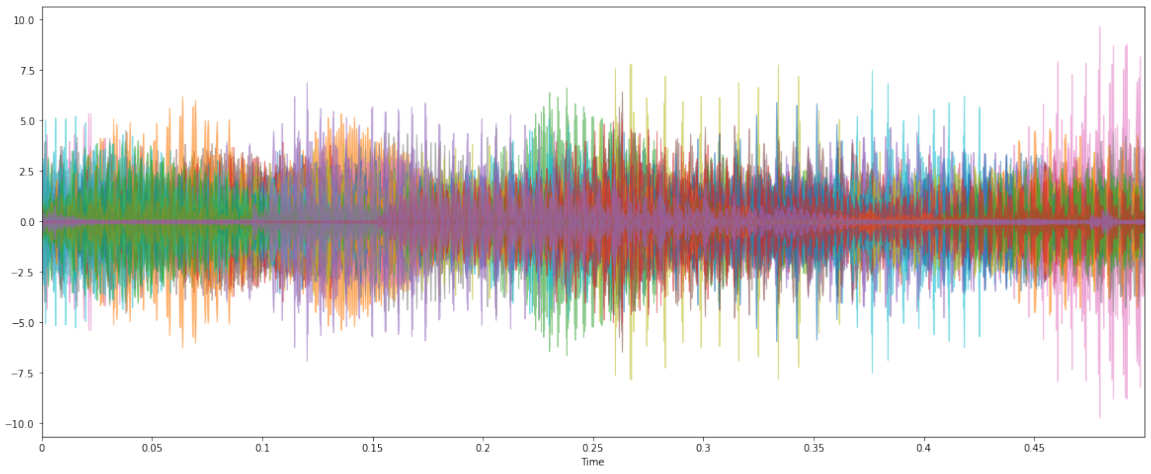

Figure 4: Sample of normalized signals

2.3 Normalize the signals

Another important aspect of preprocessing was the normalization of the signals using the Z score, so that all the audios had zero mean and standard deviation equals to one (Figure 4).

2.4 Other attempts

I also tried removing many of the noisy samples (SNR over certain threshold), but this approach led to a lower score. I think that this behaviour is given by the fact that the noisy samples are also in evaluation set, therefore training on them could help the classifier recognize the type of noise and classify it correctly. I also tried a band-pass filter in order to have only human voice frequencies in the recordings, but this method led to less information in the signals which caused a drop in accuracy of the classifier.

2.5 Feature extraction

For the feature extraction part I used a Python library called *librosa* [2] which is a library made for signal processing. I extracted 96 features for each audio clip which are:

- Zero Crossing Rate feature
- Root Mean Squared feature
- Spectral Centroid feature
- Spectral Bandwidth feature
- Spectral Rolloff feature
- 7 Spectral Contrast features
- 12 Chromagrams features

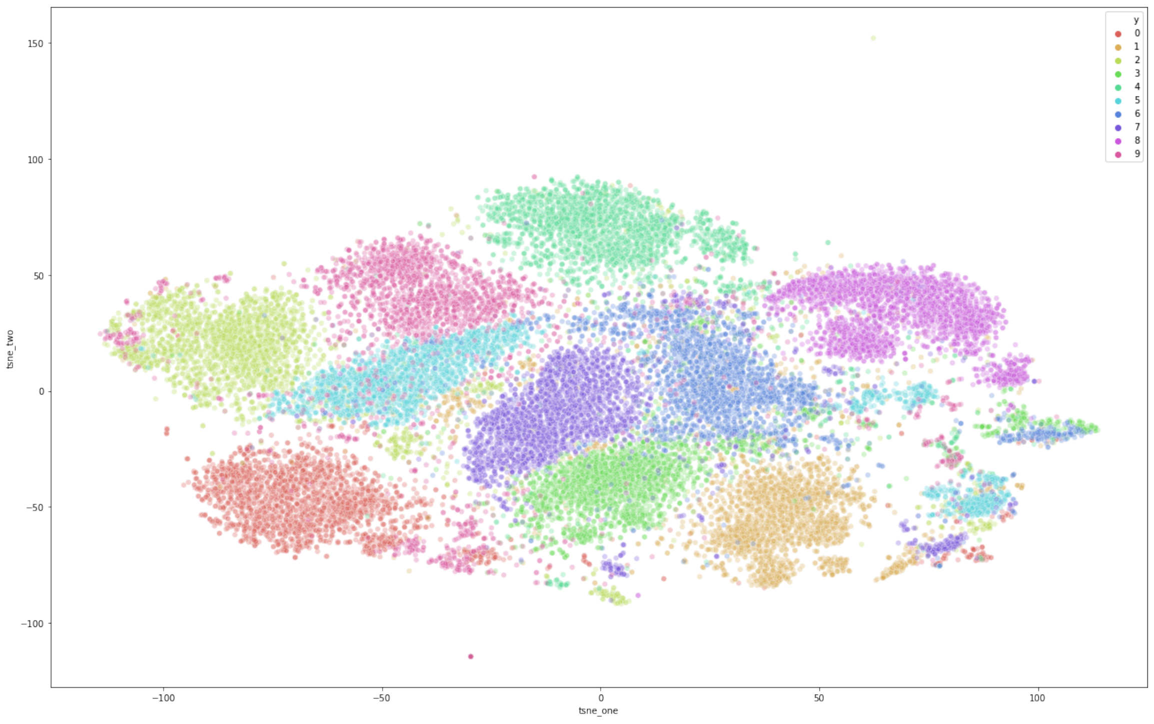

Figure 5: t-SNE algorithm performed on the extracted features in the development set

- 26 MFCC features
- 26 Deltas for MFCC features
- 20 Mel Spectogram features

The features were calculated using a moving window of 0.03 seconds. After computing the feature for each window, the mean of the values for each feature was taken to represent the signal and then it was standardized on the whole dataset.
Using t-SNE algorithm to reduce the features to 2 dimensions, we can clearly see (Figure 5) the clusters for each class, proving that the features are well selected. However, noise and outliers are present too, mainly in between different clusters which are not very far.

3 Algorithm choice (max. 400 words)

I used two different algorithms which are:

- Support Vector Machines (SVM)
- Artificial Neural Network (ANN)

Both are largely and commonly used for classification tasks.
SVMs try to learn the decision boundaries (not necessarly linear) using the training points and then classify the test points based on where inside the decision boundaries they fall. In this case, in which the distributions of development and evaluation sets where identical, this seemed a good choice. The most important parameters for the SVM algorithm which have been tuned are the kernel function (mapping of data to higher dimension) and the cost parameter C (which is the cost of misclassified points, the higher the more complex will be the decision function, but more difficult to have misclassified points).
I also used an ANN (multi-layer perceptron) which usually is very robust to outliers, it seemed a good choice because in this case there were many. A common problem when using ANNs is that, in order to properly train a network, we need huge amount of data, and in this case, more training data could have helped achieve better results. The ANN I used consists in three layers: an input layer, a hidden one and an output one. I didn't want to have a deep network or a very large one in order to prevent overfitting. The activation function I used was the ReLU because it is very stable, even though the model was not deep and there were no problems of vanishing gradient. I tuned hyperparameters as learning rate (the rate at which the algorithm updates the weights), number of neurons for hidden layers and the optimizer used.
The training process of the two algorithms is very different both in time and method. SVM iterates over data until a certain stopping criterion is matched and every time a wrongly assigned point is found, it updates the decision boundary. It also uses a kernel to map the points to a high dimensional space where data is separable by means of hyperplanes. ANN instead needs more time to train because the whole training set is used many times. The network is trained by feeding into it a batch of data at a time and, using backpropagation algorithm, the weights and biases of the network are updated.
A problem about these two algorithms is the interpretability. They both are sort of black boxes that give the desired result, but is very difficult to understand why a certain decision was made.

4 Tuning and validation (max. 400 words)

The implementation of both algorithms was done using scikit-learn Python library. In order to find the best configuration of the hyperparameters for both algorithms a 5-fold cross validation strategy was used. This validation method is used to build a robust classifier with a good performance estimation because each configuration is tried 5 times, each one with a different validation set, and the average score of the 5 runs is returned. The best configuration is then used to build the best possible classifier using all the training points.

4.1 SVM

For the SVM algorithm I tuned the hyperparameters using this grid:

kernel: [rbf, poly, sigmoid, linear]
gamma: [auto, scale]
C: [0.01, 0.1, 1, 5, 10, 50, 100]

The search for optimal hyperparameters was done by using the aforementioned method of cross validation, where the training set was 80% of the size of the whole development set. At the end of the grid search, the best grid was used to train the final classifier using all available development set. Initially the range in which I searched for the best C value was larger, but after many tries it resulted that the best performance was usually accomplished by an SVM with C between 5 and 10 and with a RBF kernel. With this configuration the model reached a 0.9665 f1 score on the public evaluation set.

4.2 ANN

For the ANN I used the MPLClassifier class of scikit-learn library. The way I tuned the best hyperparameters for the ANN is the same as for the SVM, using a 5-fold cross validation with 80% of the development set. The hyperparameters which I tuned are:

hidden_layer_sizes: [(512,), (1024,), (2048,), (4096,), (512, 512)]
learning_rate_init: [0.0001, 0.001, 0.01]
max_iter: [200, 400, 600]
solver: [sgd, adam]

The best configuration found was a single hidden layer of size 4096, Adam optimizer and a learning rate of 0.0001. This achieved a f1 score of 0.9681 on the public evaluation set.

4.3 Performance evaluation

In order to check and visualize the performance of the classifier I used the confusion matrix which encodes how well a classifier performs given the correct labels. An example can be seen on Figure 6. Most of the classes are correctly classified and the biggest misclassification can be seen between speaker "g" and "d". This is plausible given the fact that both are male speakers and are likely to share common features.

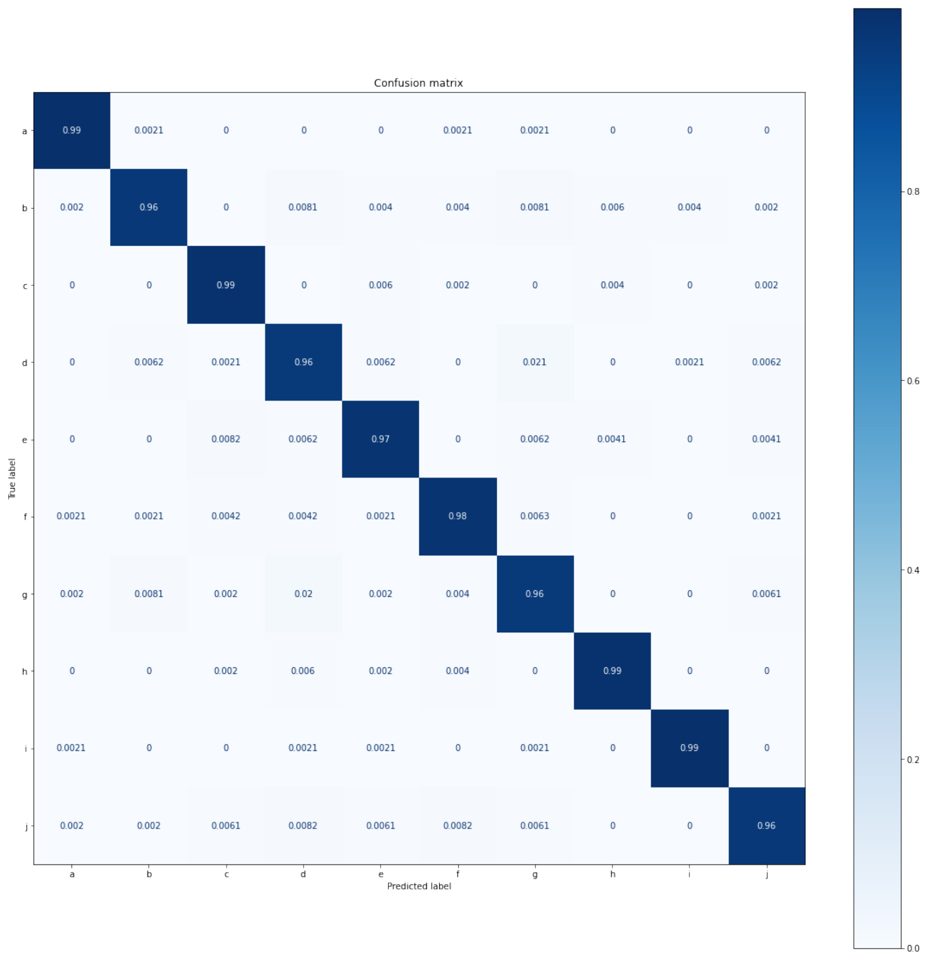

Figure 6: Confusion matrix for SVM

References

[1] Librivox, https://librivox.org.

[2] librosa, https://librosa.org.

[3] noisereduce, https://pypi.org/project/noisereduce/.