# Spoken Language Intent Detection Classification Problem

Isak Sundstroem, Eleonora Quaranta
*Politecnico di Torino*
s306740 s316198
s306740@studenti.polito.it s316198@studenti.polito.it

***Abstract*—This report presents a possible solution to a classification problem regarding intent detection from audio recordings. This solution is based on the spectral analysis of audios, performed by means of a spectrogram that is split into a defined number of blocks. Each block is then used to extract a summary of statistical properties used by the classification model. The result will be evaluated in terms of accuracy of the predicted intent of each recording.**

## I. Problem Overview

The dataset provided for this project contains a collection of audio recordings of people uttering commands meant for a voice controlled AI assistant. Each recording is supposed to convey an "object" on which to be acted upon, and an "action" to use on the "object". The purpose of this project is to develop a classification pipeline to predict both the "object" and the "action". Each data object also contains features for: "self-reported fluency level", "first language spoken", "current language used for work/school", "gender", and "age range". For the purpose of this project all these features were considered and encoded: features like the gender and the age group can be useful because they allow to take into account possible differences in the pitch of the audios, whereas features regarding the fluency and the different languages spoken allow to identify the possible presence of accents, which can have a meaningful impact on the way an audio recording is labelled.

The dataset is composed of a development set containing 9855 objects, and an evaluation set containing 1455 objects.

In order to better understand the data on which we will be working, it is possible to plot the recordings both in the time and in the frequency domain, as shown in Figure 1 and Figure 2; these two representations are complementary, as it is possible to extract data with respect to the frequency domain from the time domain, by using the fast Fourier transformation function available from NumPy.

Firstly, to gain some insight to the nature of these recordings, we begin by plotting them all on top of each other with time on the x axis and amplitude on the y axis (see Figure 3).

An issue with these recordings is that their duration varies significantly. As can be seen in Figure 3 and Figure 4, some of the recordings are as short as just a fraction of a second while some are as long as 20 seconds. When looking at the plots one of the longer recordings, it is evident that that most of these longer duration can be attributed to long trailing silences and random noise not related to the spoken words.

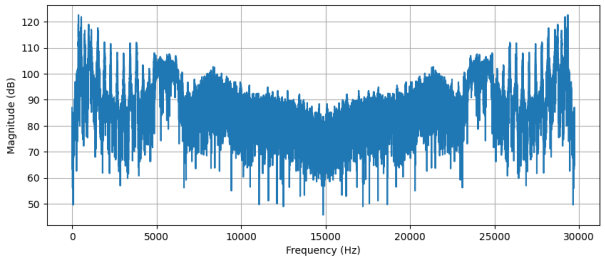


Fig. 1. Visual representation of an audio recording in the frequency domain

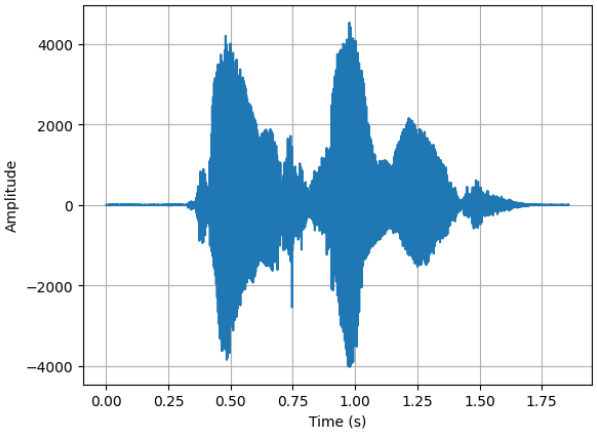


Fig. 2. Visual representation of an audio recording in the time domain

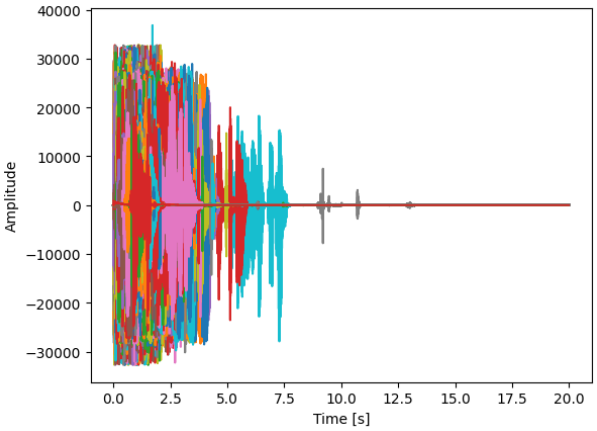


Fig. 3. Original recordings

Another consideration that has been made regards the different amplitude of the signals, visible in Figure 3: while some recordings reach higher amplitude values, others don't. This could have an impact on the training of the classifier, and therefore should be taken into consideration.

The presented problems will be addressed in the preprocessing stage (section II-A).

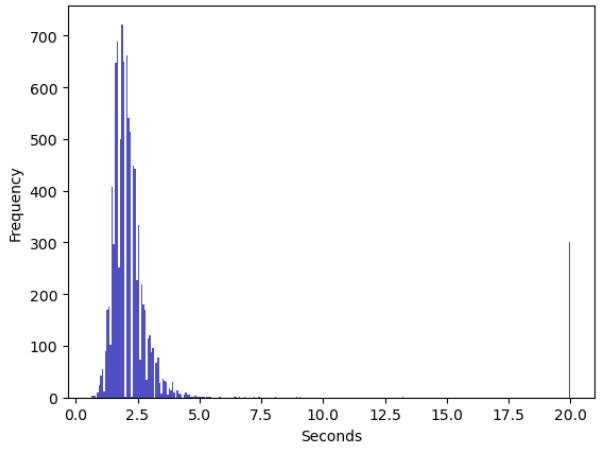


Fig. 4. Distributions of the recordings' duration

## II. Proposed approach

### A. Preprocessing

For practical reasons, we introduced two additional features to the dataset:

- "AudioData", to make sure that each recording is read only once and immediately stored in the dataset.
- "SampleRates", to analyse the potential differences among records.

These two additional features will make it easier to access audio data of each recording without having to repeat reading operations more than one time. Additionally, from an analysis of the unique values of the "SampleRates" attribute, it is possible to notice how the majority of the recordings in the evaluation set have been sampled at a frequency of 16kHz, whereas a minority of them has been sampled at 22050 Hz. In order to standardise this feature, we can re-sample the recordings having a higher sample rate such that, after this procedure, we are able to work with recordings evenly sampled at 16 kHz.

After this, in order to address the different amplitudes of the recordings, we scaled them by applying z-score normalisation. The change in amplitude after the scaling process is performed can be observed in Figure 5.

As mentioned in the previous section, another problem that needs to be solved is the variation in the length of the recordings. To address this issue, we can use the "effects.trim" function in the Librosa library, which trims trailing and leading silences from audio recordings. We found that using a threshold of 30 dB provided the best balance between removing the noise and not cutting too much of the actual voice lines. This however does not sufficiently uniform the duration of the recordings. In order to address this, we can use a somewhat "quick and dirty" solution by simply cutting off each recording after a certain amount of time, in this case after three seconds. This will not remove any random noise in the beginning of the recordings, but as we can see in Figure 5, the majority of relevant data is usually in the first seconds of the recordings, so the negative impact should be minimal. After trimming and cutting the files, we get a much more reasonable distribution of duration, shown in Figure 6. Figure 7 shows all of the audios

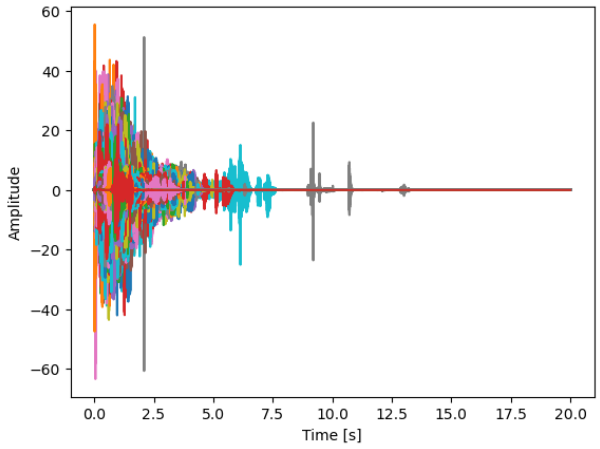


Fig. 5. Recordings after normalisation

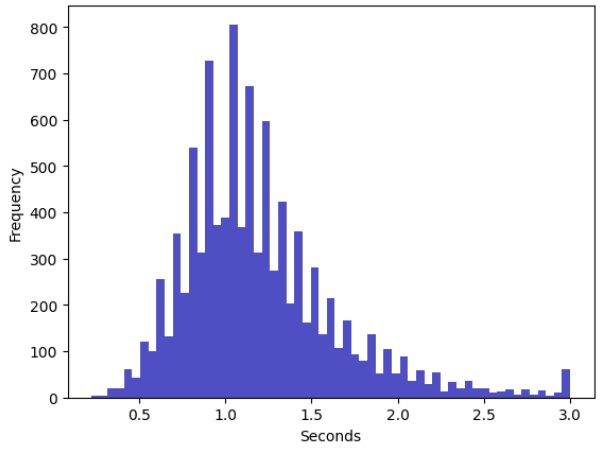


Fig. 6. Distributions of the recordings' durations, trimmed and cut

plotted on top of each other after these first modifications.

In order to classify different recordings, it is crucial to define what information can be used to identify patterns and other peculiar characteristics in each audio: the study of audio signals can be conducted both on the time and on the frequency domain. A tool that allows to join the analysis on the two domains is the spectrogram (shown in Figure 8), which is able to show the variation of the frequency of a signal over time. For its characteristics is largely used

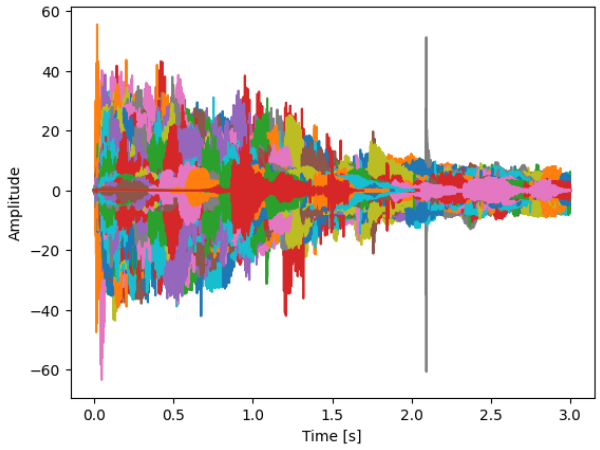


Fig. 7. Recordings, trimmed and cut

in speech recognition tasks [1] . In order to define how to

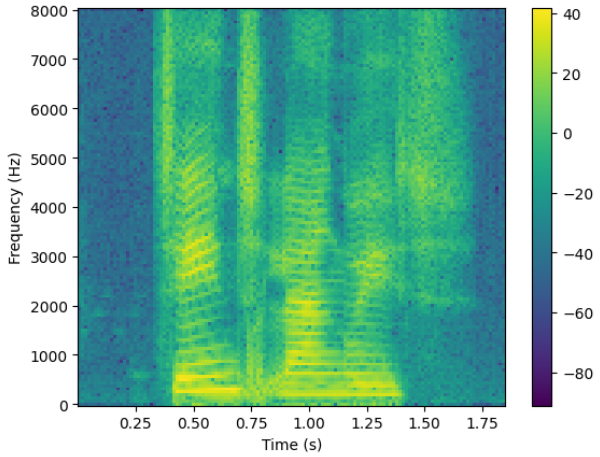


Fig. 8. Spectrogram of an audio recording

incorporate it in a classification pipeline, it is important to define how a spectrogram can be computationally represented: a spectrogram can be stored into a 2-dimensional array of dimension $N_f \times N_t$, where $N_f$ corresponds to the number of frequency bins and $N_t$ to the number of time bins. One of the possible ways to incorporate the spectrogram in the classification is to consider the $N_f \times N_t$ elements of the array and use them to characterise each recording. The problem with this approach though is that the number of blocks is not homogeneous among the recordings, and therefore neither the number of features per audio would be. One possible solution to this problem is to implement a division of the spectrogram into a fixed number of blocks, such that all the recordings can be characterised by the same number of features regardless of their length. The implementation of this approach consists in splitting the spectrogram array into an arbitrary number of blocks $n \times n$, where the decision of having the same number of splits on the two dimensions is given by a necessity to simplify the model and to make it less computationally expensive. Figure 9 shows an example of a split spectrogram. For each block obtained from the spectrogram, a statistical summary of its values is computed and stored; in particular, for each block will be computed:

- its mean value;
- its standard deviation.

It can also be pointed out how this approach allows to reduce the number of features used, therefore simplifying the model: instead of using all the values composing the spectrogram, we only consider a reduced number of statistically summarised elements.

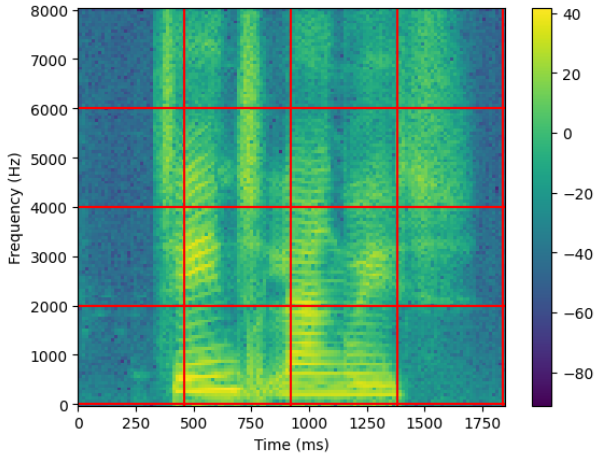


Fig. 9. Example of a spectrogram split in $4^2 = 16$ blocks

### B. Model selection

The model selected for this classification task is the random forest classifier; this choice has been made based on both the overall performance of the classifier in terms of accuracy and its robustness to outliers and noise. It is relevant to mention that also the SVM model was initially considered, since it is largely used alongside random forests for speech classification tasks [2], but since the high dimensionality of the dataset used would have resulted in a major impact on the complexity of the hyperparameter tuning and on the training of the SVM, we decided to focus on the random forest classifier. The model adopted is suitable to handle high-dimensionality data thanks to the combination of bootstrap aggregation and feature bagging. These techniques also reduce the risk of overfitting and ensure the decorellation among the decision trees.

### C. Hyperparameters tuning

The hyperparameter tuning stage is a crucial part of the classification process, since it allows to properly calibrate the classifier based on the accuracy obtained with the different configurations on a test set. The set used to evaluate the different hyperparameters combinations is obtained from the development set, using an 80/20 train/test split. The parameters that need to be tested are:

- Random forest parameters;

- n for the spectrogram splitting.

As for what concerns the parameters of the classifier, a way to assess which configuration achieves the best results in terms of accuracy is to run a grid search on it, and use the accuracy score to evaluate the results. Since we must also tune the number of blocks in which every dimension of the spectrogram is split, it is necessary to find a way to implement the two processes. Two possible options were considered:

1) For each candidate n, run a grid search on the classifier and store the combination of parameters that achieves the highest accuracy;
2) For each candidate n, train the classifier using the default parameters and select the value of n that allows to achieve the optimal accuracy, and then use it to perform an hyperparameter grid search on the random forest.

While the first option may be more complete since it examines all possible combinations of hyperparameter configurations and values of n, it is also undeniably computationally expensive. In order to simplify the model used, we can assume that the second option provides a good trade-off between efficiency and completeness, and therefore opt for it. The parameters values considered during the tuning process are shown in Table I. Note how the maximum tested values for n is 14: this choice was made in order to avoid working on an excessively big dataset; since n is the number of time and frequency bins in which each spectrogram is divided, the actual number of blocks we are working with will be $n^2$, meaning that if $n = 14$ we would be handling $14^2 = 196$ additional features for each statistical measure computed on each block.

TABLE I
VALUES CONSIDERED FOR HYPERPARAMETERS TUNING

| Model | Parameters | Values |
|---|---|---|
| Spectrogram splitting | n | $2 \rightarrow 14$, step 2 |
| Random forest | n_estimators | {50, 100, 200, 300} |
| | criterion | {"gini", "entropy"} |
| | max_depth | { 10, 20, 50, 100, None} |
| | min_samples_split | {2, 5, 10} |
| | min_samples_leaf | {1, 2, 5} |

## III. RESULTS

The variation of accuracy with respect to different values of n is shown in Figure 10; from this representation we can notice how the accuracy achieved by the random forest classifier improves when n increases, and therefore we can set $n = 14$. At this point, it is possible to run the grid search to identify the best configuration for the random forest classifier. The grid search returned this as the optimal configurations among those tested: {n_estimators=300, criterion="entropy", max_depth=None, min_samples_split=2, min_samples_leaf=1}. At this point, it was possible to load the evaluation set and perform the same preprocessing operations performed on the evaluation set, and initialise the random forest classifier using the parameters just found. The prediction achieves a public score of 0.674, double the provided baseline of 0.334. We also built a "naive" solution using a random forest classifier with its default parameters and $n = 14$, to investigate the impact of the hyperparameter tuning: this solution obtained a public accuracy score of 0.626. It is immediately noticeable how this value is not far from the accuracy score obtained with the tuned model: as can be seen from Figure 10, the performance of the default classifier improves significantly when n increases.

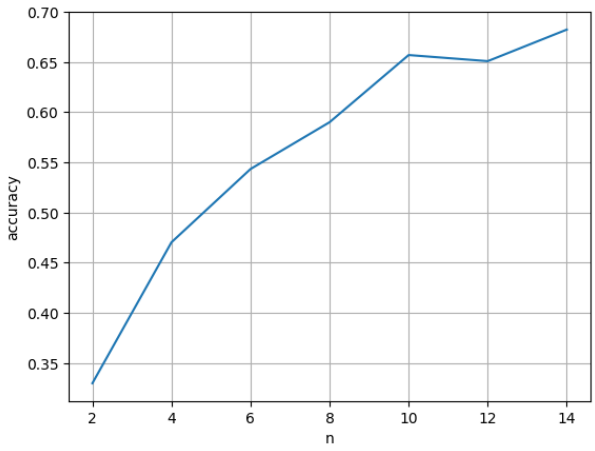


Fig. 10. Accuracy of RF default classifiers with respect to n

## IV. DISCUSSION

In this project we were able to process audio recordings into usable features for a speech recognition classification pipeline. We used the processed audio recordings combined with the provided categorical features to train a random forest classifier, which was able to achieve an accuracy well above the naive baseline accuracy score for this problem. Despite this, some improvements that could be made to further improve the solution are:

- Implement a more thorough processing of the audio recordings, including the removal of noise at the beginning and end of them, and of background noise; this would contribute to make the voices more clear and understandable.
- Implement more elaborate methods for classification: among the most used we can find artificial neural networks [3], which are indeed complex models that allow to reach higher accuracy levels. The trade-off when using neural networks is that their training is more complex than the training of simple classifiers. Another promising option is to use Linear Predictive Coding (LPC) and Mel Frequency Cepstral Coefficients (MFCCs) [2] alongside with a random forest or an SVM classifier.

## REFERENCES

[1] V. Zue and L. Lamel, "An expert spectrogram reader: A knowledge-based approach to speech recognition," in *ICASSP '86. IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 11, pp. 1197–1200, 1986.

[2] M. Gupta, S. S. Bharti, and S. Agarwal, "Implicit language identification system based on random forest and support vector machine for speech," in *2017 4th International Conference on Power, Control  Embedded Systems (ICPCES)*, pp. 1–6, 2017.

[3] L. Deng, G. Hinton, and B. Kingsbury, "New types of deep neural network learning for speech recognition and related applications: an overview," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 8599–8603, 2013.