Speech recognition for unbalanced dataset of voice commands using Mel-spectrogram

Matteo Bianco
Politecnico di Torino
s300781
s300781@studenti.polito.it

Filippo Grobbo
Politecnico di Torino
s305723
s305723@studenti.polito.it

***Abstract*—In this report we show our approach to a Natural Language Processing task whose aim is identifying underlying intention or purpose of a spoken statement. Classification of audios is performed looking at their Mel-spectrograms and inferring different types of features from each of them. Two different models are compared after performing proper hyperparameters tuning and final results far exceed baseline.**

***Index Terms*—NLP, speech recognition, Mel-spectrogram, SVM, random forest**

I. Problem Overview

The proposed assignment is a classification problem in the context of NLP (Natural Language Processing) and speech recognition. The goal is predicting an action-object pair (e.g. 'increase volume') requested in an audio sample.

The dataset is composed by a collection of audio files in WAV format, each associated with some additional informations regarding speakers. It is divided in two parts:

- a development set, consisting of 9854 labeled samples
- an evaluation set, consisting of 1455 samples to be labeled

Informations regarding speakers include for example *speakerId* and *gender*. In brief there are 5 categorical features and 1 alphanumeric string for each audio.

We straightaway notice that classes in the development set are unbalanced. The least represented one (*deactivatelights*) has just 552 samples, while the most present one (*increasevolume*) has 2614 observations. Moreover, making an analysis about audio duration, we find what is shown is Figure 1.

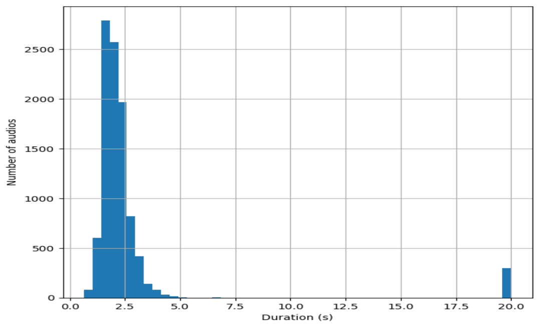

Fig. 1. Histogram of audio durations

Some audios have an anomalous time length of around 20 seconds. Listening to some of them we realize that they are composed by a lot of silence. This could be seen looking at Figure 2, where there are lot of low amplitude values.

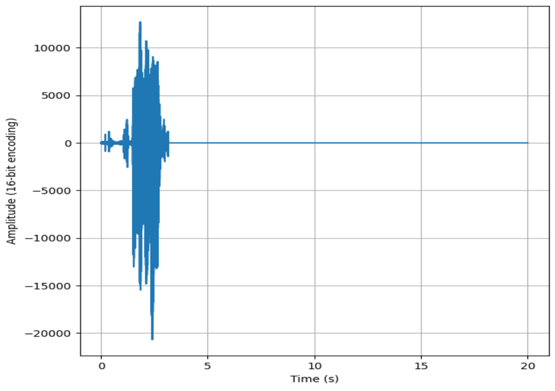

Fig. 2. Audio with id 616, represented in time domain

Finally, sampling rate values for audios are 16000 Hz or 22050 Hz and amplitude values range from -32767 to +32767, so they all are in a 16-bit encoding.

II. Proposed Approach

A. Data Preprocessing

Since we noticed that classes are unbalaced in term of number of samples, we decide to perform a random oversampling strategy to even out groups. In partcular, we consider the set with the highest number of samples as reference and duplicate samples from minority classes in order to balance class distributions. In order to do it practically we explot the python library *Imbalanced-learn* [1]. We finally obtain a training set with 18286 samples and we shuffle them in order not to have similar datas next to each other.

The next step consists in trimming audio silent sections. Indeed, as found before, in many audios the part containing speech is much shorter than the total length. To trim audios we fix a threshold amplitude value under which cutting silence. In particular, we scroll audios starting from the beginning until first value above the threshold is found. Next, the same is done starting from the end. Given these two points we cut audios maintaining only values between them. We think in fact that silence in the middle of the speech can be significative to recognize action-object pairs. In Figure 3 it is shown the trimming of audio in Figure 2.

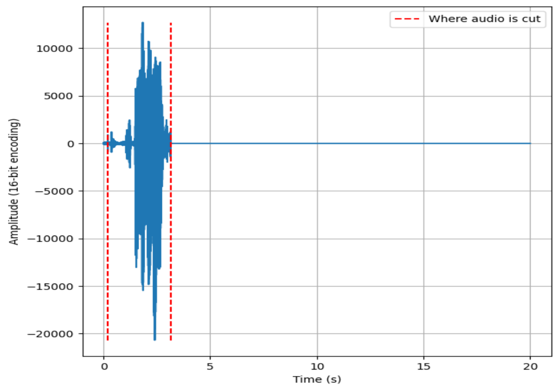

Fig. 3. Trimming procedure of audio id 616

Concerning the threshold value, we firstly took the mean of absolute amplitude values (957.46). After the trim, we listened to some audios and noticed that many were cutted too much. At this point we decided to try different lower values to be more conservative. Eventually we use a value equal to 500. It is worth observing that some audios are completely trimmed by the procedure. The reason behind is that, listening to all of them, we discover they do not contain any speech but just white noise. Thus, this method allows us also to discover the presence of some wrong data that we decide to drop.

Subsequently, we handle the problem of noise reduction. We find on literature [2] a noise reduction algorithm that reduces noise in time-domain signals like speech. It relies on a method called "spectral gating" which is a form of noise gate. It works by computing a spectrogram of a signal and estimating a noise threshold (or gate) for each frequency band of that audio. That value is used to compute a mask, which gates noise below the frequency-varying threshold.

Once we have cleaned audio signals, we compute a Mel-scaled spectrogram of each sample, as suggested in [3]. We adopt this scale because it is a way to represent audio data trying to mimic the way human ear perceives different frequencies. Mel-scaled spectrogram output is in a logarithmic scale on the frequency domain, while it is still linear in power units. For this reason we convert the power scale to decibel (calculated using the highest value as a reference) for better visualization and interpretation of the data. Figure 4 shows Mel-spectrogram computed on the same audio considered in previous plots. More details about different time length of Figure 3 and 4 are given in following section.

Concerning feature engineering, we work directly on spectrograms to extract attributes to characterize data. Since each audio has a different duration, we choose to divide every spec-

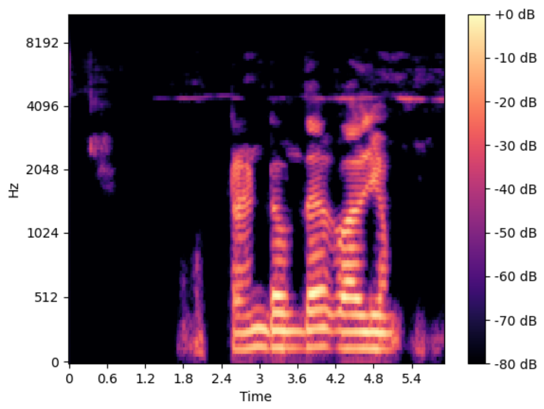

Fig. 4. Mel-spectrogram of audio id 616 after being trimmed

trogram in a fixed number of blocks with possibly different shapes. Then, we compute the following statistics on each block:

- mean value
- standard deviation
- minimum value
- maximum value
- 0.25-quantile
- 0.75-quantile

Afterwards we add a feature for each sample containing time duration of the trimmed audio.

In conclusion we perform label encoding for features *ageRange* and *Self reported fluency level*. This is a natural consequence of the fact that their values are ordered: *22-40 41-65* and *65+* for the first and *basic*, *intermediate*, *advanced* and *native* for the second. Moreover we apply one-hot encoding for *gender* and we drop attributes *Id*, *path*, *speakerId*, *First Language spoken* and *Current language used for work/school*. Reason for this choices will be given in the following section.

B. Model Selection

The following algorithms have been tested:

- *SVM*: this method works by finding the best hyperplane to separate different classes in the data. Moreover it is also able to discover non linear boundaries exploiting kernel functions. Samples were initially shuffled because in *scikit-learn* [4] the class for SVM does not have a parameter for shuffling. This classifier is used after normalizing features, since it typically benefits from it.
- *Random forest*: ensemble learning technique characterized by a collection of decision trees, where each tree is trained on a different subset of the data. This method allows also to compute features importance. Dropped features at the end of previous section were the ones with lower feature importances in a first preliminary exploration.

We decide to apply these methods because they are known in literature to be good classifiers for speech recognition [5].

C. Hyperparameters tuning

Before tuning more tecnical hyperparameters related to preprocessing and model choices, we take a look at Mel-spectrogram parameters. Given an audio, the length of time window and the degree of overlapping frames are two elements that can affect resulting spectrogram. We chose to fix their values in order to have a high level of overlapping frames. Consequences of this assumption is that we are taking more frames to cover same audio duration, which implies changing time duration of the spectrogram. This is why Figure 3 and 4 have different time length on x axis.

Regarding hyperparameters tuning we focus essentially on:

- number of blocks into which divide each Mel-spectrogram
- specific hyperparameters of SVM and Random forest

Concerning the first type of hyperparameters, we notice that in the training set there are some spectrograms having just 6 columns. Thus we decide to fix number of vertical partitions to 6 and concentate our analysis just on the number n of horizontal partitions. The tuning of this parameter and of classifiers is done in a 5-fold Cross Validation setting. Considered values are shown in table I.

TABLE I
HYPERPARAMETERS USED IN GRID SEARCH

Model	Hyperparameters	Values
Preprocessing	n	{16,32}
SVM	*kernel*	{rbf,poly}
	C	{1,5,10,50}
	degree	{3,5,7}
Random forest	*n_estimators*	{160,170,180}
	min_sample_split	{2,4}
	max_samples	{0.7, 0.8}

It is important to notice that, concerning SVM, the hyperparameter *degree* applies just when the *kernel* is *poly*.

III. Results

In general we can see that both models perform quite well, even if SVM is more powerful than Random forest. The best value for hyperparameter n is 16 for SVM and 32 for Random forest. Regarding classifiers hyperparameters, best combination for random forest is *n_estimators*=180, *min_sample_split*=4, *max_samples*=0.7 (mean accuracy after cross validation=0.883), while for SVM is *kernel*=rbf, C=50 (mean accuracy after cross validation=0.923). For SVM it is useful to observe that highest value used for C is 50 in the gridsearch. This suggests us that using an even higher value may exceed performances achieved so far. However, fitting new SVMs with C=100, 200 or 500, we get lower accuracy values. Thus we conclude that the best value for hyperparameter C is 50. Moreover, we notice that with these last three values we get the exactly same accuracy score on each of the 5 fold of the cross validation. Since increasing C means making SVM with harder margins and thus tolerating fewer errors, this fact probably means that for $C \geq 100$ errors are not allowed anymore, so that it is indifferent how big C is. We finally train the best SVM on all training set and obtain public accuracy score of 0.849 on the evaluation set. Doing the same with Random forest the score is 0.745.

IV. Discussion

The proposed approach obtains good results. Main strenghts are related to the using of Mel-spectrogram and the great effort put in feature engineering. However, we have identified some aspects that may improve results obtained:

- Different resampling methods can be adopted. For example, we can generate new observations instead of just duplicate existing ones, as done by SMOTE [1]. Apart from this, also the strategy adopted using random oversampling can be improved. Looking at number of samples in each class we chose to equalize classes with reference to the one with the largest number of observations. Different tradeoffs can be studied to obtain lower generalization error.
- The fact that scores obtained during cross validation are fairly different from the ones in evaluation set highlights a bit of overfitting. Actually this happens due to the stategy we adopted to balance classes: during cross validation, it is common that a copy of a data used for the training is present also in the validation set. In addition, having such a large number of statistics computed for each block in the Mel-spectrogram may have led models to focus too much on the distribution of training data. In future analysis might be better to study this aspect more in detail.
- A very typical strategy in the context of speech recognition is using Mel-frequency cepstral coefficients (MFCCs) to characterize each audio [6]. This approach is so widespread in NLP that we think a more thorough analysis has to be done.
- Another very common approach in NLP context is using convolutional neural networks (CNN) for speech recognition [3]. Since they they represent the state of the art in this field, we believe that the results can improve with their use.
- Different combinations of both preprocessing and model hyperparameters can be also explored to obtain stronger results.

Despite all these possible improvements, we think that our approach already gives encouraging results.

References

[1] G. Lemaître and F. Nogueira and C. K. Aridas, ”Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning”, vol.18, Journal of Machine Learning Research, 2017, pp.1–5. Accessed: Jan. 21, 2023. [Installed]. Available: http://jmlr.org/papers/v18/16-365.html.

[2] T. Sainburg. *timsainb/noisereduce: v1.0*. (2019). Zenodo. Accessed: Jan. 21, 2023. [Installed]. Available: https://doi.org/10.5281/zenodo.3243139.

[3] C. Hickey, B. Zhang, ”Classifying verbal commands using Convolutional Neural Networks and Mel-spectrographic sound representations”, KIISE Transactions on Information Science and Engineering, 2020, pp.428–430.

[4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel et al., ”Scikit-learn: Machine Learning in Python”, vol. 12, Journal of Machine Learning Research, 2011, pp.2825–2830.

[5] P. Dhakal, P. Damacharla, A. Y. Javaid, V. Devabhaktuni, ”A Near Real-Time Automatic Speaker Recognition Architecture for Voice-Based User Interface”, Machine Learning and Knowledge Extraction, vol. 1, 2019, pp.504–520.

[6] C. Ittichaichareon, S. Suksri, T. Yingthawornsuk, ”Speech recognition using MFCC”, vol.9, International conference on computer graphics, simulation and modeling, 2012