Machine Learning approaches to Intent Detection

Luca Catalano
Politecnico di Torino
Student id: 308658
s308658@studenti.polito.it

Sara Montese
Politecnico di Torino
Student id: 308482
s308482@studenti.polito.it

***Abstract*—In this paper, we introduce two possible Machine Learning approaches to address the problem of intent detection. The solution compares different classification models: Random Forests and Support Vector Machines. By leveraging both time and frequency-based features of the audio files, both models achieve promising results.**

***Index Terms*—Machine Learning, Random Forest, SVM, Audio Classification, Intent Detection**

I. Problem overview

Intent detection aims to recognize the action that the user wants to perform. The proposed task focuses on classifying the user's intent, given a dataset of spoken utterances. The dataset consists of two parts:

- Audio files: audio samples in *.wav* format
- Metadata: files *.csv* that contain information about each audio sample in the dataset and about its speaker.

The dataset provided is composed of 11,309 recordings (*.wav*) of user queries with English pronunciation. The metadata have been distributed in two separate collections:

- a *development* set, containing 9,854 records of recordings for which the intent is known;
- an *evaluation* set, containing 1,455 records of recordings without the target variable.

We used the development set to build a classification model capable of classifying user intentions for each record of the evaluation set. We can make some considerations based on the development set. First of all, the problem is not perfectly balanced: the number of samples for each of the 7 intents is quite different (Table I).

TABLE I
Intent labels and number of utterances in each label

Intent	Number of samples
IncreaseVolume	2212
DecreaseVolume	2163
IncreaseHeat	1108
DecreaseHeat	1095
ChangeLanguageNone	1044
ActivateMusic	747
DeactivateLights	506

We notice there is an error in one of the intents of the training samples ('ChangeLanguageNone' should not contain the substring 'None'), nevertheless we keep this format to label the evaluation set samples. An important step was to carry out data exploration in order to derive an overview of the data. We started by visualizing the distribution of the features values. With reference to the attributes related to the language of the speaker (Self-reported fluency level, First Language spoken and Current language used for work/school), we noticed that approximately 96% of samples assume the following same values: Self-reported fluency level: native, First Language spoken: English (United States), Current language used for work/school: English (United States). In second place, we visualized the distribution of the audio lengths. As shown in Figure 1, we notice that there are some outliers that have a length significantly different from the mean duration (2.64s). Upon manual inspection, those signals contain some silence following the utterance and should not pose any particular problem. However, since the entries have different lengths, we needed to devise an approach to extract the same number of features from all data points (since most classification models require a fixed number of features). Furthermore, we notice

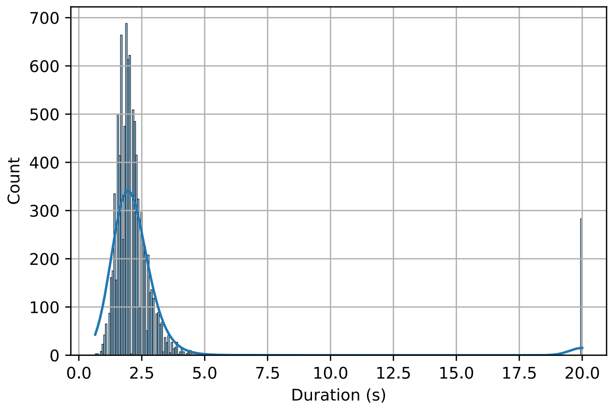

Fig. 1. Length distribution of audio signals

that in the training set most of the recordings have been sampled at the frequency of 16 kHz, and a minority (283 samples) have a sample rate of 22.05 kHz. On the other hand, in the evaluation set all recordings have been sampled at the rate of 16 kHz. To get a better understanding of the type of data, we can inspect some signals visually, both in time and in frequency domains. These two domains introduce useful and complementary information on each signal. Figure 2 shows one signal in the time domain. We can immediately notice that there can be some pauses before and after the utterance.

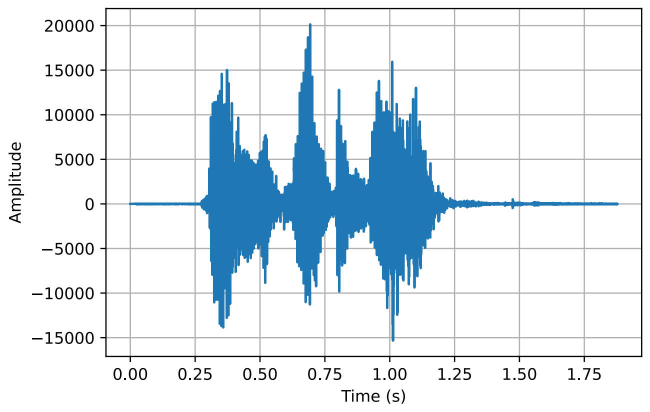

Fig. 2. Representation of a recording in the time domain

Figures 3 and 4 show the signal in the frequency domain (in linear and in decibel scale). From this, we can observe that some frequencies carry more energy than others. The linear scale shows a mostly flat plot with only a limited number of high-energy frequencies. The decibel scale, on the other hand, is a logarithmic scale and highlights a wider range of involved frequencies. Since the human perception of sounds is logarithmic in nature [1], we can consider the dB scale to be a more meaningful representation than the linear one. We will adopt this representation during the preprocessing phase.

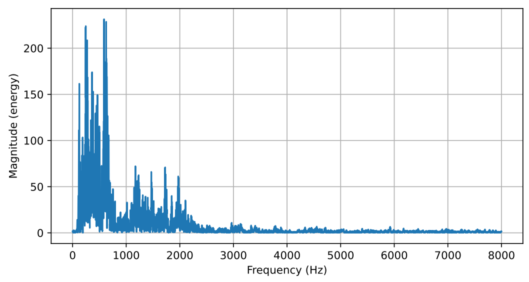

Fig. 3. Magnitude of a recording in linear scale

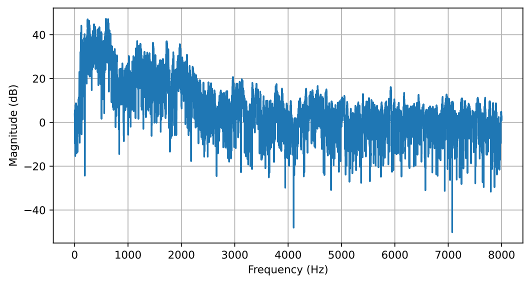

Fig. 4. Magnitude of a recording in dB scale

II. Proposed approach

A. Preprocessing

The training data in *.wav* format cannot be input directly into the model. We have to load the audio data from the files and process them in order to reach the quality and the format the model expects. We load and resample all the audios to the same sampling rate of 16 kHz. By analyzing the length distribution of the signals, we observe that the 95th percentile of audio lengths is around 3.6 seconds; we proceed by considering anything below 20 db as silence, and so removing leading and trailing silence from all signals. Since the audio data provided cannot be understood by models, we have to convert it into an understandable format by building a vector representation of the data. We do not consider the metadata as significant features for our models. We proceed by extracting several features out of our initial representation of the recordings. Since we are dealing with audio signals, we can choose to work:

- On the *time* domain. The features are obtained by splitting the signal into chunks and characterize them by means of statistical measures.
- On the *frequency* domain. The features depend on the frequencies contained in the signal. This involves reshaping the signal using a transformation function (e.g. Fourier transform) and work on its spectrum of frequencies.

Since both time and frequency domains contain useful information regarding the recordings, we can leverage both by using the *spectrogram* of each signal. In a spectoral representation of audio signals, we get time on x-axis and different frequencies on y-axis. Values in the matrix represent different properties of the audio signal related to particular time and frequency (e.g. amplitude, power, etc). An example of a spectrogram of a signal is shown in Figure 5. To extract features from

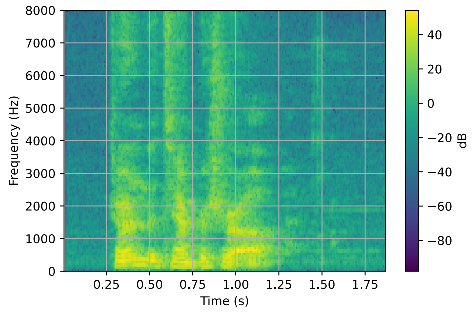

Fig. 5. Spectrogram of a signal

a spectogram, it can be seen as an N × M matrix (with N frequency bins and M time bins). We can divide this matrix into n_γ rows and n_ρ columns. This produces $n_\gamma \cdot n_\rho$ blocks. For each block, then, we can compute some summary statistics (in this case, mean and standard deviation) and consider them as features sets. The number of sub-matrices is considered an *hyperparameter* for classifier. Section II-C explores how we can define meaningful values for n_γ and n_ρ. To better analyze the audio samples, other important features have been extracted [2]:

- *Zero Crossing Rate*: is the rate of sign-changes along a signal (the rate at which the signal changes from positive to negative or back);
- *Root-Mean-Square* (RMS): it refers to the total magnitude of the signal, which in layman terms can be interpreted as the loudness or energy parameter of the audio file;
- *Spectral centroid*: it indicates where the "centre of mass" for a sound is located and is calculated as the weighted mean of the frequencies present in the sound;
- *Spectral rolloff*: is the frequency below which a specified percentage of the total spectral energy lies (in our case 85%);
- *Spectral bandwidth*: is the variance with respect to the spectral centroid;
- *Spectral flatness*: it quantifies how noisy the wave signal is;
- *Spectral flux*: it measures how quickly the power spectrum changes across a signal;
- *Spectral contrast*: it is the difference between the spectral energy peak and its minimum.

B. Model selection

We will now discuss more in depth the two models proposed for the classification task:

- *Support Vector Machine*: this algorithm applies a transformation to the data using a kernel function. It tries to find the hyperplane which maximizes the margin of errors between predicted and actual values that are tolerated. This model assumes that the data are in a standard range, so it requires feature vectors normalization.
- *Random Forest*: this algorithm belongs to the category of *ensemble* techniques because it combines multiple decision trees (trained on different portions of the training set and subsets of features) to classify data. This is generally done to enhance the performance and to avoid overfitting. RFs work on one feature at a time, thus there is no need to normalize the dataset.

For the Random Forest and SVM classifiers, the best-working configuration of hyperparameters has been identified through a grid search, as explained in the following section.

C. Hyperparameters tuning

There are two main sets of hyperparameters to be tuned:

- n_γ and n_ρ for the preprocessing
- Random Forest and SVM parameters

By assuming that the two are uncorrelated, we can first select n_γ and n_ρ and then move on to the classifiers' parameters. Another simplifying assumption we can make is that n_γ=n_ρ. This helps us to significantly lower the number of combinations we need to consider, as we now need to try n=n_γ=n_ρ configurations. In order to tune the value of n we first need to split the development data set. We decide to keep 80% of the observations in the training set, and use the remaining 20% for the test set. We run a grid search on n; to this end, we train a Random Forest and a SVM classifier with their default configurations and assess their performance based on the resulting accuracy score. After choosing a value for n, we can run a grid search[1] on both SVM classifier and Random Forest to find the best configuration of hyperparameters for our models. The fine-tuning process uses a cross-validation approach with five folds. The search space is summarized in Table II.

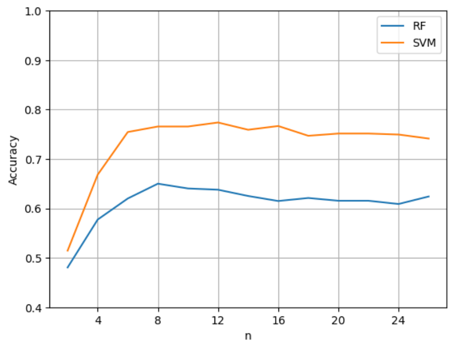

Fig. 6. Performance of default SVM and Random Forest as n varies

TABLE II
HYPERPARAMETERS CONSIDERED

Model	**Parameter**	**Values**
Preprocessing	n_γ, n_ρ	2 $\rightarrow$ 26, step 2
Random Forest	*criterion*	{entropy, gini}
	min_impurity_decrease	{0.0, 0.05, 0.1}
	n_estimators	{100, 150, 200}
SVM	*C*	{1, 10, 20, 30, 40}
	kernel	{rbf, poly, sigmoid}

We evaluate each configuration on the test set using the accuracy score, both for the RF and the SVM classifier.

III. Results

In Figure 6 we can immediately see that SVM is more stable than Random Forest as n increases. Both classifiers achieve satisfactory performance for n = 12, which we can select as a reasonable value.

With this value of n, we run the grid search for the classifiers and we identify the following best configurations:

- SVM: {C: 10, kernel: rbf}. With this configuration, we obtain an accuracy score of 0.817 on the test set.
- RF: {criterion: entropy, min_impurity_decrease: 0.0, n_estimators: 200}. With this configuration, we obtain an accuracy-score of 0.661 on the test set.

Regarding the *public score* performance, since the SVM classifier is the best performing model, it has been used to label the evaluation set. The classification has been submitted to the leaderboard achieving an accuracy score of 0.872. We

[1]With another 80/20 train/test split

notice that the public score is higher than the accuracy reached on the test set; hence the private score should almost reflect the public one because there should not be overfitting. Also the RF classification result has been submitted achieving, as expected, lower scores. Comparing the SVM classifier's score with the leaderboard baseline, the quality of the classification is improved from an accuracy score of 0.334 to an accuracy score of 0.872. Also, the difference between this solution and the head of the leaderboard is around 0.10.

IV. Discussion

We believe that our approach and experimentation to this classification problem has been quite successful. Our belief mainly lies in the preprocessing and in the feature extraction steps, which are the true cores of the entire pipeline. The consideration of statistical features, time domain features and spectral features in frequency domain turned out to be an exhaustive and meaningful way of performing feature engineering from original audio recordings. The following are some aspects that might be worth examining to further improve the obtained results:

- Run a more exhaustive grid search on the classifiers' hyperparameters. Moreover, additional classification models might be considered.
- Apply data augmentation to our training audio files. This process creates many variations of natural data, and can act as a regularizer to reduce the problem of overfitting and improve the generalization ability of our models.
- Consider other feature extraction approaches, for example by computing Mel Frequency Cepstral Coefficients (MFCC) and use them as input to a CNN classification model [3].

References

[1] E. B. Goldstein, Sensation and perception (3rd ed.). Wadsworth/Thomson Learning, 1989.

[2] McFee, B. et al., 2015. librosa: Audio and music signal analysis in python. In Proceedings of the 14th python in science conference.

[3] V. Bansal, G. Pahwa and N. Kannan, "Cough Classification for COVID-19 based on audio mfcc features using Convolutional Neural Networks," 2020 IEEE International Conference on Computing, Power and Communication Technologies, India, 2020.