Machine learning project: Speech recognition challenge

Massimiliano Conte
massimiliano.conte.2@studenti.unipd.it

Pierpaolo D'Odorico
pierpaolo.dodorico@studenti.unipd.it

1. Introduction

The faced problem is the speech recognition challenge, where the goal is to build a system that can automatically recognize spoken words. In particular, the spoken words are given as input in form of audio recordings of one second length, while the output of the system is the text of the recognized words. All the recordings are waveform of one of the following 8 words:

- Down;
- Go;
- Left;
- Off;
- On;
- Right;
- Stop;
- Up.

This task is a key component in many artificial intelligence services, such as virtual assistants, and more generally speech recognition is part of the natural language processing domain.
Our work begin after the features extraction provided us by the professors, in form of log mel-spectrogram, that is a bidimensional representation of the recordings, involving frequency and time. After that, the images of the spectrograms are first resized and than reshaped in 1024-dimensional vectors.
We have faced the problem by applying several machine learning methods, from the easier ones to the more complicated ones, and choosing the best performing method based on accuracy. This led us to a system capable of classifying spoken words with an accuracy of more that 95%.

2. Dataset

The used dataset is a reduced version of TensorFlow Speech Commands Dataset v0.0.1 [1]. The data provided us is alredy divided in training and validation set. The training set is composed of 1600 samples, 200 for each of the 8 classes, and the validation set is composed of 109 samples. Every recording has a sampling frequency of 16 kHz. The hop length used for constructing the log Mel-spectrogram is 512, while it is not specified the window size. The feature extraction mimics the humany auditory system, providing a representation of the audio taking into account the fact that humans perceve both the frequencies and the amplitude of the sound logarithmically. The whole preocess of extracting the features for each recording can be summarized in:

- **Create the windows**, by sampling the recording and making hops of size 512;
- **Compute the discrete Fuorier transform** for each window, using the fast Fourier transform algorithm;
- **Convert to the Mel scale**: changing the representation from of the frequencies from Hz to the Mel scale (the scale of pitches judged by listeners to be equal in distance one from another, a kind of human perceving frequency scale);
- **Create the features**, by computing the log Mel-spectrogram (that is an image), resizing and reshaping it to a 1024 dimensional vector.

The dataset provided us alredy contains the extracted features and the correct class of the samples. The dataset doesn't have unbalanced classes problem.

3. Method

The methods used in this project are machine learning tecniques that are well suited for multiclass classification tasks. We tried the performance of various methods under different configurations, starting from the basic models and ending with the more complicated ones. Many models we tried are based on binary classification, the way we used such models are using the one-vs-all strategy, i. e. training one binary classifier per class and than predict the instances by loocking at the classifier that maximizes the confidence score.

3.1. List of used methods

Every method can be tuned by changing some hyperparameters. In the following table we reported the tecniques, which of those hyperparameters we taked into account for

selecting the best configuration, and how that method handle multiclass classification.

Method	Hyperparameters	handling
Linear Classification	C : Regularization	One-vs-all
Logistic Regression	C : Regularization	One-vs-all
K-Neighbors Classifier	K : number of neighbors	Majority vote of the K-neighbors
Classification tree	Max depth; Min samples leaf; Min impurity decrease	Naturally handle multiclass
Random forest	Number of trees	Naturally handle multiclass
Support Vector Machine	C : Regularization; Type of kernel	One-vs-all
Neural network		Softmax activation on the output layer

Table 1. List of used methods.

All the methods are implementes using *Python* [2] and *Sklearn* [3], exept for the neural networks where we used *Keras* [4]. The host for the challenge is *Kaggle* [5].

3.2. Metodology

For every listed method we first set a grid of hyperparameters configurations, then trained one model for each configuration and then choosed the best performing one. For fairness reasons, we didn't use the validation set for choosing the best configuration, keeping the validation set unseen by the models until the decision of the overall best performing method.
In particular, we divided the training set in two parts, a sort of training and validation sets for the hyperparameter search. The models were trained on the "little" training set and the best configuration was choosen on the "little" validation set, by looking at the highest accuracy. Then the model is trained again on the entire training set, using the found configuration, in order to take advantage of all the data.
Once that we found the best configuration for each model, we used the unseen validation set for choosing the overall best method.
Usually, this kind of split is referred as training, validation and test set, but here the set used as test set was alredy named validation set.

4. Experiments

In this section we describe the hyperparameters we choosed and show, where it is possible with graphs, what's the best configuration that we found. After that, we compare the methods accuracy on the validation set.

4.1. Linear classification

The Hyperparameter used in this method is *alpha*, a regularization tuner. The larger *alpha*, the stronger the regularization. It refers to the *l2*, which is the standard regularizer based on the euclidian norm of the parameters vector.

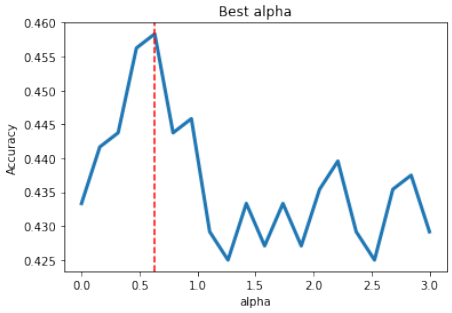

Figure 1. Accuracy as the intensity of the regularization varies.

4.2. Logistic regression

The Hyperparameter used in this method is *C*, which has a similar role as the *alpha* decribed before, tuning the intensity of the regularization. The larger the *C*, the weaker the regularization. Can be seen as $\frac{1}{alpha}$.

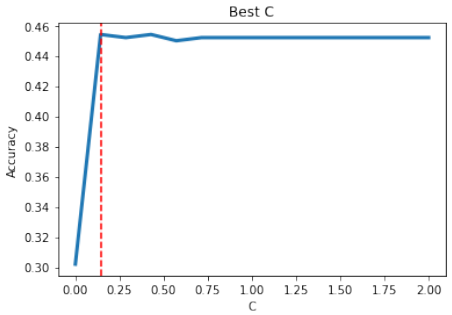

Figure 2. Accuracy as the intensity of the regularization varies.

4.3. KNN

The Hyperparameter used in this method is *K*, the number of neirest neighbors examined for the majority vote.

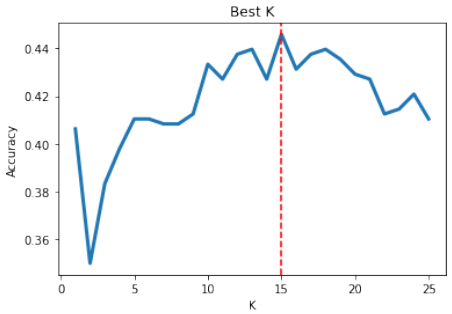

Figure 3. Accuracy as the number of the considered neighbors varies.

4.4. Decision tree

The criterion used for measuring the impurity on the leafs is the gini impuruty:
$G(leaf) = \sum_{i=1}^{8} n(i, leaf)(1 - n(i, leaf))$ where $n(i, leaf)$ is the number of samples in the $leaf$ that belong to the i-th class.
The Hyperparameters used in this method are:

- *Max depth*;
- *Min samples leaf*, the minimum number of samples required for each leaf;
- *Min impurity decrease*, the minimum decrease of impurity required for each split.

The best configuration found is:

- *Max depth* = 9;
- *Min samples leaf* = 2;
- *Min impurity decrease* = 0.00316.

4.4.1 Random forest

This method is the aggragation of *n trees*, where each was trained on a different set, obtained via bootstrap replications of the training set. *n trees* was set to be 1000. Each tree uses the previous found best hyperparameter configuration.

4.5. Support vector machine

The Hyperparameters used in this method are:

- *C*, the same regularization parameter as before;
- *Kernel*, the type of kernel used by the method, choosed between polynomial or gaussian.

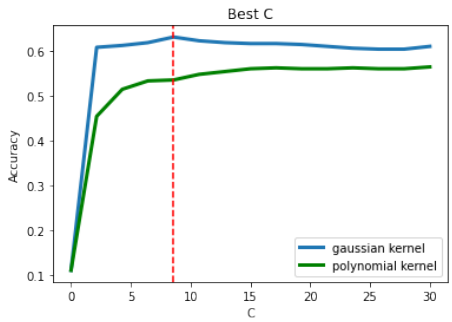

Figure 4. Accuracy for polynomial and gaussian kernels as the intensity of the regularization varies.

4.6. Neural network

References

[1] Martın Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[2] Guido Van Rossum and Fred L Drake Jr. *Python tutorial*. Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands, 1995.

[3] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[4] Francois Chollet et al. *Keras*. 2015. URL: https://github.com/fchollet/keras.

[5] *Kaggle*. https://www.kaggle.com.