UNIVERSITÀ DEGLI STUDI DI MILANO - DIPARTIMENTO DI INFORMATICA

Experimental project - "Statistical methods for machine learning" Urban Sound Classification with Neural Networks

Pedretti Davide, 957314

Academic year 2020/2021

Contents

Chapter 1

Introduction

Declaration: I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

In this report I analyze the experimental results from neural networks trained to perform a multiclass classification of sound events. They were based on audio files from the *UrbanSound8K*[1] dataset.
It consist of one of the most famous and used datasets in the field of urban sound classification. It is composed of 10 different folders. Neural networks (NNs), or ANNs (Artificial Neural Networks) are a large class of predictors that can be applied to a wide range of tasks. In this particular case I have used them in sound classification.
The work was divided as follow: the first phase was feature extraction. In this part I took the audio files and extracted some valuable information from them. Then, I performed the multiclass classification through neural networks. An important step at this point was the tuning of the hyperparameters. Finally, I reported the obtained avarage accuracy, which is a metric that gives an idea of how did the model performed, and the standard deviation across the different tested folders. I divided the work into two scripts: the first one is dedicated to the feature extraction phase and the second one to the model. They were called *feature_extraction.py* and *neural_network.py*.
I used *Python* [2] as programming language because of its flexibility and its great diffusion. Since it is very used, it happens quite often to find people dealing with recurrent problems and tips. Then, I used *Spyder* [3] from the *Anaconda* [4] package as integrated development environment.

Chapter 2

Dataset and feature extraction

Environmental sounds classification is an important study area of artificial intelligence anche machine learning applications. The *"UrbanSound8K"* is surely one of the most used datasets while testing the performances of multiclass classification model which goal is to use audio files (in this case in .wav format) that are not composed of speech or music.
It is possible to find in the literature many works conduced on environmental sound source classification, both using unsupervised and supervised methods.
"UrbanSound8K" is composed of 8732 audio samples coming from a urban environment. They belong to ten different classes:

- Air Conditioner
- Car Horn
- Children Playing
- Dog Barking
- Drilling
- Engine Idling
- Gunshot
- Jackhammer
- Siren
- Street Music

The dataset is divided into ten different predefined folders. Each of them contains labelled audio files from different classes.
It is possible to find the label indicating the class in the file name. For instance, the "6988-5-0-2.wav" file from "fold3", represents a sound coming from the "engine idling" class. It is visible through the number "5" after the first "-" in the file name, which indicates the class.
Furthermore there are other extra information, called meta-data. They are contained in a *csv* file. Some of them are the start and the end of the original recording of each file. Also, there is the folder to which the file has been allocated. Then, the class identifier. In particular, I used this information to have a clear mapping between the class identifier and the effective noun of each class.
The first phase of this project was the **feature extraction**. It is mainly used to identify some key features in the data. Later on, these features were extracted and used to feed the model. In particular, I focused on extracting three audio features using a very diffused library in *Python*, called *Librosa* [5]. As stated in *Librosa* official documentation, it *provides the building blocks necessary to create music information retrieval systems.*
The chosen features were the following:

- *Mel-frequency cepstral coefficients (MFCCs)* [6]
- *Chroma features* [7]

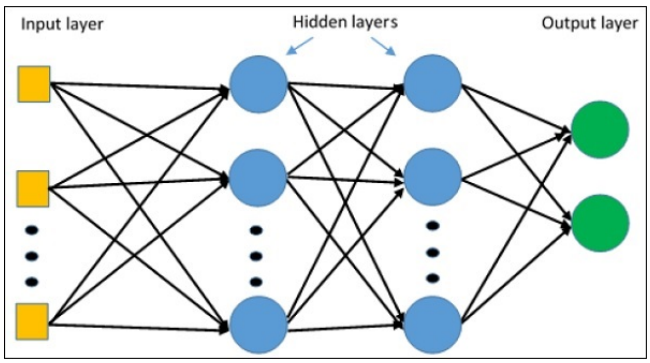
- *Root-mean-square (RMS)* [8]

I used those features because they are particularly representative of the audio files. I took the first 12 MFCCs, the first 12 chroma features and the RMS for each audio file. Then, I calculated some statistics (mean, median, max and min) on them.
Finally, I obtained a feature vector using the concatenation function of a *Python*'s library called *Numpy* [9]. This feature vector is composed of (x,100) rows and columns. The number of rows strictly depends on the considered number of files. For example the training and the testing files will contain a different number of rows, depending on the number of files contained in each folder. The number of columns depends on the features. I added an extra column containing the class ID of each file.
The last step was to save those audio features into a file. I aimed to get a clear representation of the selected features to feed the network later on in the work. The chosen format was the *comma-separated values (csv)* [10]. The first column indicates the class for each audio file and the remaining indicate all the selected features.
In particular, I saved the training data into a *csv* file containing the features for the folders 1, 2, 3, 4 and 6. Then, I saved the testing data into five different *csv*'s, one for each of the testing folders (5, 7, 8, 9 and 10).

Chapter 3

Experimental setup

Once data were loaded using the *genfromtxt* function from *Numpy*, the first thing to do was to standardize them using the *StandardScaler* from *sklearn.preprocessing* module. It was useful to rescale features removing the mean and scaling to unit variance.
Then, I tried to apply a dimensionality reduction method called *PCA* (Principal component analysis). It is an unsupervised method and it is mainly exploited in exploratory data analysis. Specifically, I used *PCA* from the *sklearn.decomposition* module. Its main parameter is *n_components* and here an integer can be specified to indicate the amount of features to keep. Hovewer, if this number is between 0 and 1 the PCA *selects the number of components such that the amount of variance that needs to be explained is greater than the percentage specified by n_components*[11]. Exploiting this method and choosing 0.99 as parameter, the remained features were only 59.
The following phase was the classification one. In this work, I decided to use a multi-layer perceptron [12], which is a class of feedforward artificial neural network.



To do this, I imported *Tensorflow* [13], which is an open source library dedicated to machine learning. In addition, I used *Keras* [14], which is integrated in *Tensorflow*. It helped me to create the structure of the network, which is made of at least three layers: one input layer, one hidden layer and one output layer. In this case I used 3 hidden layers, each of them composed of a certain number of neurons.
Since the number of neurons in each layer is tightly connected to the complexity of the data and the problem, I tried different network architectures in order to find the best one. To do this, I changed some parameters in the way explained in the next chapter.
Once decided the number of layers and the number of neurons for each of them, another important parameter to specify was the *activation function*. I used a *relu* activation function for the hidden layers and a *softmax* one for the output layer.

This last chosen was taken due to the fact that it is usually used when there are multi-class classification tasks, especially when class membership is required on more than two class labels.
Finally, I selected the last parameters while compiling the model. In particular, I chose *sparse_categorical_crossentropy* as a loss because it is used in these types of problems. For example, if it was a regression problem, I wouldn't have used it. Then, I selected the Adaptive Moment Estimation (*Adam*) optimizer in order to calculate errors and to update parameters. It is a learning rate optimization algorithm that has been designed for training deep neural networks.

Chapter 4

Hyperparameter tuning

One of the most relevant steps of this work is the *hyperparameter tuning*. The aim of this step is to improve the machine learning model and, as a consequence, its performances.
Since every machine learning model make use of some external parameters, called *hyperparameters*, it becomes crucial to change some of them and not to use the provided default values.
Anyway, it is also necessary to find a trade off between the execution time and the performances: in fact, it is possibile to tune those hyperparameters in an exhaustive way (meaning that all the possible combinations of them are tested) or in a randomic one. These two different ways were exploited in this work in order to see in a practical way the points for and against them.
More formally, they are called **grid search** and **random search**. I used the *sklearn* library for it because it provides a fast way to test those approaches (*GridSearchCV* and *RandomizedSearchCV* from the *sklearn.model_selection* module).
As an intuition, the first one will provide better performances and the second one will provide a faster execution time. Indeed, the randomized search is very fast because it only depends by the quantity of data and the number of iterations specified. Nevertheless there is an important point to consider: since the grid search in exhaustive, there could be some cases in which a model tuned with the grid search overfits and, as a consequence, results less accurate on unseen data than a model tuned with the randomized search.
This was the reason why I decided to use the randomized search: it reduces chances of overfitting and it is also much quicker. So, I left in the code the grid search commented and the randomized one available for being used.
The main hyperparameters on which I focused my attention were the following:

- Number of neurons in the first, second and third hidden layer
- Learning rate
- Batch size
- Epochs

Of course they are not the only hyperparameters: there are many more of them, including activation functions, loss function etc.. I selected only some of them to reduce the execution times and I took autonomous decisions for the remaining.
As mentioned in the previous chapter, I opted for a *relu* activation function for the hidden layers and for a *softmax* activation function for the ouput one without any tuning. Another relevant chosen parameter without tuning was the *dropout rate*. It is mostly used to prevent overfitting, which is one of the most important problems when dealing with machine learning models. It refers to a situation in which a model learns the training data too much; it has bad consequences while dealing with never seen data.
Dropout is a regularization method and can be implemented in some hidden layers or in all of them. Furthermore, it is not used on the output layer. The parameter

that needs to be specified is the probability of training a certain node in a layer: 1.0 stands for no dropout. In this work I manually tested some different values of it and, in conclusion, I opted for a dropout rate of 0.4 for all the hidden layers. Technically, I added a dropout layer for each hidden layer.
All the attempts, both in grid search and in random search, were run based on a grid of reasonable hyperparametes. In particular, I chose the following:

- first hidden layer size = [96, 128]
- second hidden layer size = [48, 64]
- third hidden layer size = [16,32]
- batch size = [32, 64]
- epochs = [40, 70, 100]
- learning rate = [0.01,0.001]

For example I selected those values for the learning rate because they are standard values and they are very used in practice. As precedently said, in the end I selected the randomized search with the number of iterations set to 10. The hyperparameters were stored in a dictionary called *final_parameters*.

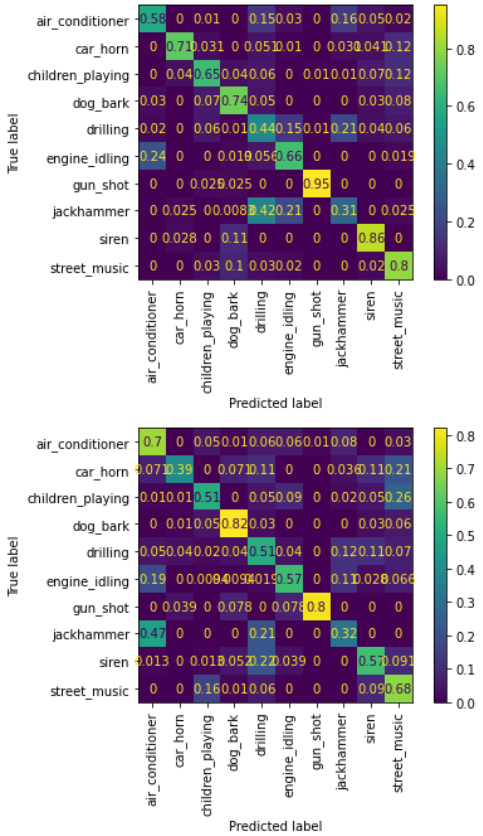
Chapter 5

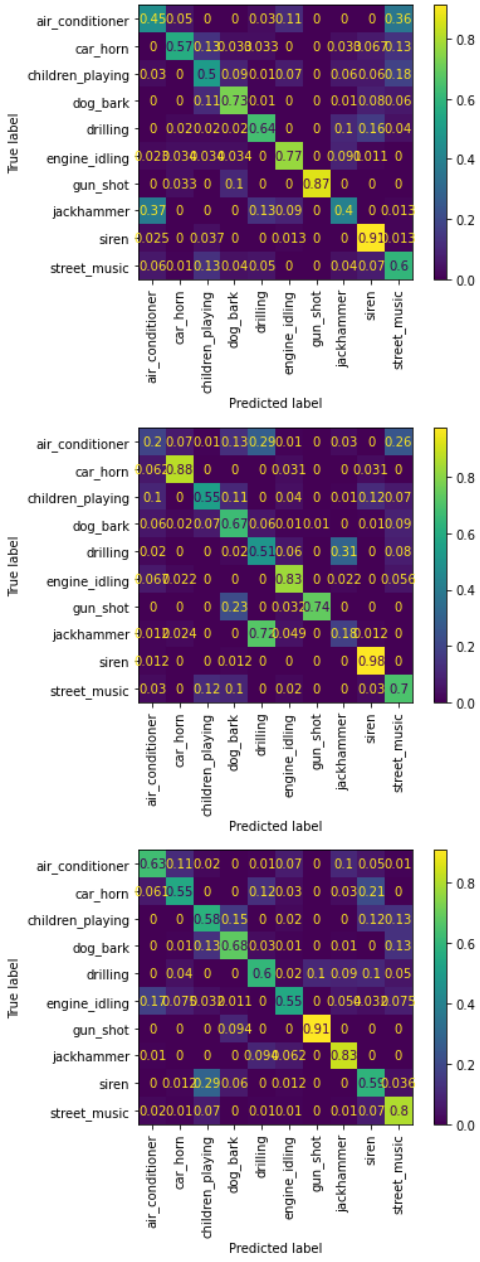
Conclusion

In order to get reproducible results, I decided to add a random seed at the beginning of the code in the classification script. The *PCA* method described in chapter 3 was not used in the end. It was tested but it did not help to reach a better test accuracy. For this reason, I left it commented. Once selected the randomized search as a method for the hyperparameter tuning, and selected 10 as *n_iter* parameter, the obtained accuracy and the standard deviation across the folders were the following:

```
The avarage accuracy across the test folders is:  0.6246310876255106
The standard deviation across the test folders is:  0.026531687223190978
```

In addition, I attached the confusion matrix for each of the testing folders. They are respectively representing the situation in folders 5, 7, 8, 9 and 10. This matrix consists of a practical representation that summarize the performance of a classification algorithm.





This work can obviously be expanded by exploiting different audio features or taking even more values of the same features. For instance, I could have used also the first and the second derivative of MFCCs. Then, different neural networks like CNNs could have been exploited.

Bibliography

[1] UrbanSound8K, https://urbansounddataset.weebly.com/urbansound8k.html

[2] Python, https://www.python.org/

[3] Spyder, https://www.spyder-ide.org/

[4] Anaconda, https://www.anaconda.com/products/individual

[5] Librosa, https://librosa.org/doc/latest/index.html

[6] MFCCs,https://medium.com/linagoralabs/computing-mfccs-voice-recognition-features-on-arm-systems-dae45f016eb6

[7] Chroma feature, https://en.wikipedia.org/wiki/Chroma_feature

[8] Rms, https://librosa.org/doc/main/generated/librosa.feature.rms.html

[9] numpy, https://numpy.org/

[10] csv, https://it.wikipedia.org/wiki/Comma-separated_values

[11] PCA, https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

[12] Multilayer perceptron, https://en.wikipedia.org/wiki/Multilayer_perceptron

[13] Tensorflow, https://www.tensorflow.org/

[14] Keras, https://keras.io/