Sapienza University of Rome

Machine Learning

Course code: 1022858

Homework 1.
10-class classification

Author:
Carmine Fabbri - 2133421

November 6, 2023

Contents

1 Introduction

This is my report on the first assignment of the machine learning class course. I was engaged in training various machine learning models for two different datasets in order to perform the correct classification of 10 classes. The methods used for pre-processing the input data, training the various types of models, and evaluating the performance using different evaluation techniques will be explained below.

2 The dataset

The assignment focuses on two datasets fed with synthetic data used for 10-class classification problems; there are also two blind test datasets used to evaluate the performance of the trained models. Each of them has N samples and are in a different d input space, specifically:

- *dataset1.csv*: has an input space of 100 and 10 labels.
- *dataset2.csv*: it has an input space of 1000 and 10 labels.

The feature vectors of each dataset are represented by the entries V^i in the X column. The related true labels C_k (k=0,...,9) are found in the Y column.

As a result, each row j of column X is a feature vector arranged as follows:

$$[V_1^i, V_2^i, V_3^i, ..., V_d^i]$$

In particular, d represents the size of the feature vector, and i represents the total number of samples.

So, to be more precise the target function is:

$$f : [V_1^i, V_2^i, V_3^i, ..., V_d^i] \rightarrow \{C_k\} \text{ where } k = (0, ..., 9)$$

Once the algorithm training is finished, it will be necessary to run an evaluation test on the model using the respective blind tests to check its accuracy and precision. It is important to note that the blind tests do not have the true label column because, based on the results of our models, a specific *csv* file will be generated with all the predictions performed by the algorithm, so that a ranking will be obtained for each dataset.

2.1 Data preprocessing

2.1.1 Tools used

I opted for Python as a programming language because of its well-documented libraries and modules. Rather than reinventing everything from scratch, I built, trained, and evaluated the classifier using commonly available tools like *Scikit-Learn*, *Matplot*, and *Numpy*.

I've published the source code as a Jupyter Notebook (with a.ipynb extension) for readability and simplicity of code composition. This approach not only helps code comprehension but also makes it easier to create code snippets. Checkpoints in the notebook also serve to save interim results and display debugging information and graphics immediately within the notebook.

2.1.2 Preparing the dataset

In the field of data analysis and machine learning, the quality and characteristics of the data set are critical to the success of model accuracy. It is not uncommon to find ourselves in circumstances where data points cannot be linearly separated. In fact, as in this case, the data characterizing our dataset are not linearly separable. For this reason, traditional linear classification models may not produce meaningful results and it is therefore necessary to use learning models such as **SVM** (*Support Vector Machine*), **Random Forest**, and **K-Nearest neighbors**.

To address this challenge and improve the effectiveness of the solution, in addition to selecting an appropriate training model, it is necessary to pre-process the data in such a way as to make the aforementioned learning model more accurate and effective. For the purposes of this task and given the nature of my chosen models, I used the technique of *standardization*, also known as feature scaling, which helps to transform the data to a common scale. The formula used under the hood to convert each point is the following:

$$x_{scaled} = \frac{x-\mu}{\sigma}$$

Where:

- x: is the original input data point
- μ: represent the mean value

- σ: indicates the standard deviation

More specifically the standardization method transforms a dataset so that it has a mean (average) of 0 and a standard deviation of 1.

2.2 Model selection

Even though some models like SVM may benefit from feature scaling, others, like Random Forest, are generally not sensitive to it due to its nature.

For the purpose of this homework, I preferred to choose different types of models for each dataset. In the first exercise (*dataset1.csv*), to gain the most precise classification, I have implemented two variants of the Support Vector Machine model, one based on the **linear kernel** with hyper-parameter *C=1* and the other one with a **polynomial kernel** with hyper-parameter *C=1* and the *degree* of 3.

For the second exercise (*dataset2.csv*), I've used 3 different models in order to make an even more accurate comparison in terms of performance between them. The first one I chose is the Random Forest algorithm, the second one is K Neighbors with *neighbors* param set to 19, and the last one is an SVM model with *RBF* (Radial Basis Function) kernel with hyper-parameter *C=1*.

2.3 Model training

Once I had preprocessed the data from the csv files and chosen the training method I thought best, I proceeded to split the data, through the use of the *train test split* method, into a train and test dataset in order to correctly fit the training models.

3 The results

3.1 Performance evaluation metrics

In order to optimally evaluate the performance of the various models, in addition to evaluating all parameters such as *Accuracy*, *Recall*, and *Precision*, obtained as follows:

- Accuracy: $\frac{True\ Positives + True\ Negatives}{All\ Instances}$

- Recall: $\frac{True\ Positives}{True\ Positives + False\ Negatives}$
- Precision: $\frac{True\ Positives}{True\ Positives + False\ Positives}$

Also, other evaluation methods such as the *cross-validation* algorithm and confusion matrix were used. In Cross-validation the dataset was split into five folds, the model was fitted and trained using four of these five sets during the assessment process, and it was then tested on the fifth set. The average of the aforementioned measures acquired at the conclusion of each evaluation phase was used to calculate the final metrics like: *Standard deviation* and the *Mean accuracy*.

Finally, the confusion matrix represents in each entry how many instances of class C_j are misclassified as elements of class C_j and the main diagonal contains accuracy for each class. The confusion matrix for the best-performing algorithms of each exercise was reported in the following subchapter.

3.2 Model performance comparison

The comparison between the different algorithms in each exercise was not done only by considering the nature of the model, but also by changing the various hyper-parameters in order to find the best ones.

3.3 First exercise

As mentioned in the previous chapters, this exercise makes use of the following dataset: *dataset1.csv*. The model chosen to perform the blind test is the SVM Model with a Polynomial kernel.

3.3.1 SVM - Linear kernel

This model had the following performance:

- Accuracy: 0.9853
- Recall: 0.9853
- Precision: 0.9855

3.3.2 SVM - Polynomial kernel

I tested this type of model with different values of hyper-parameters, such as degree. The one with a degree of 3 worked better than the one with a higher degree. Below are the results.

Polynomial kernel with *Degree* of 3:

- Accuracy: 0.98
- Recall: 0.98
- Precision: 0.98

Polynomial kernel with *Degree* of 7:

- Accuracy: 0.88
- Recall: 0.88
- Precision: 0.88

More generally, I can say that I noticed that given this dataset and this type of model, the higher the degree of the polynomial kernel was, the worse the performance.

3.4 Second exercise

This exercise is based on the use of the following dataset: *dataset2.csv*. For running the blind test, I've used the best-performing model, in this case, the SVM Model with Radial Basis Function. Here below are the scores of all the models.

3.4.1 Random Forest

This model had the following performance:

- Accuracy: 0.9706
- Recall: 0.9706
- Precision: 0.9706

3.4.2 K-Nearest neighbors

In order to find the best hyper-parameter for this model, given the dataset, I performed the *Grid Search* technique for classification, finding out that the optimal number of neighbors to search is 19.

Performance obtained with param *neighbors* of 19:

- Accuracy: 0.9702
- Recall: 0.9702
- Precision: 0.9702

Performance obtained with param *neighbors* of 3:

- Accuracy: 0.964
- Recall: 0.965
- Precision: 0.964

3.4.3 SVM - Radial Basis Function kernel

This model had the following performance:

- Accuracy: 0.971
- Recall: 0.971
- Precision: 0.971

3.5 Confusion matrix

The following confusion matrix was plotted using the following libraries: *Matplot* and *Seaborn*. From the images, we can see that the solutions for both exercises are performing very well.

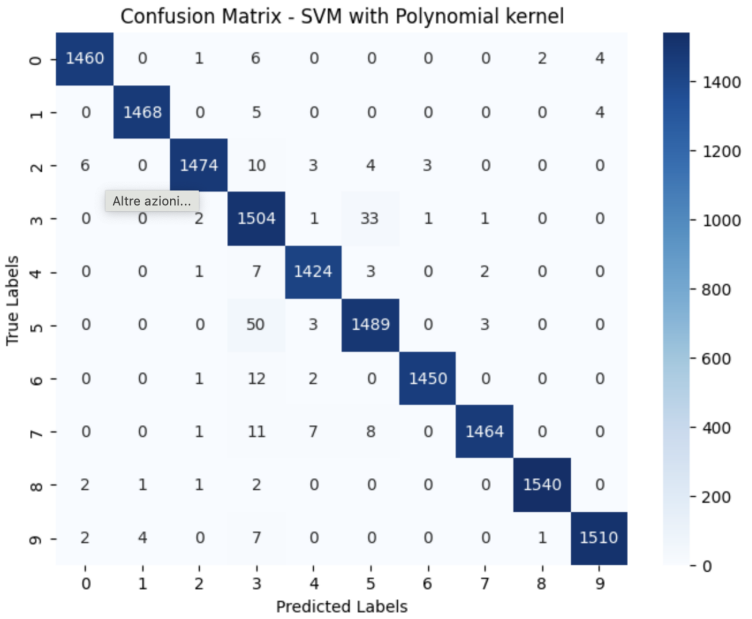

Figure 1: Error matrix for the first dataset

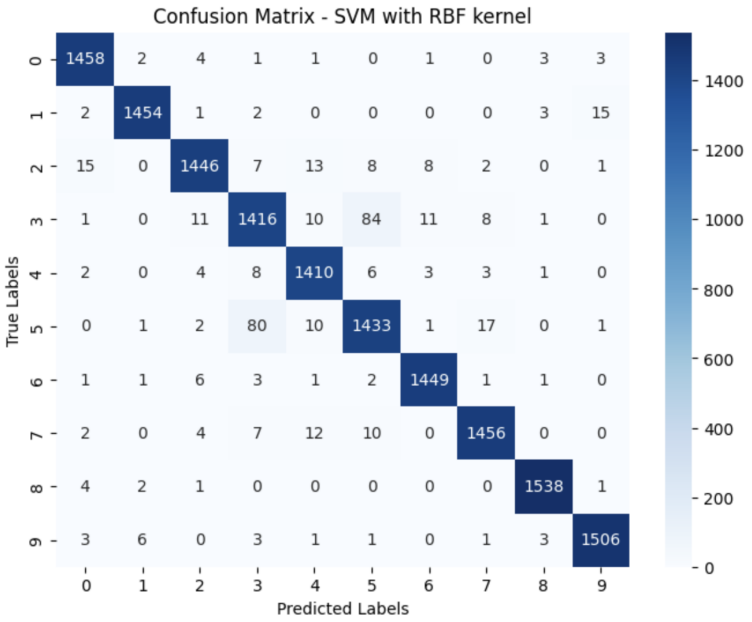

Figure 2: Error matrix for the second dataset

3.6 Computational training time

During the training phase of various models, I observed that the SVM algorithm with the RBF kernel was notably the slowest in the training phase. In contrast, the K-Nearest Neighbors (KNN) algorithm proved to be significantly faster in completing the learning process.

For example, the training time for the second dataset was also significantly different depending on the hyperparameters; in fact, the SVM model with the RBF kernel, when trained with the default parameters, finishes training in about 1 minute. The issue changes as soon as different values of the parameter *Gamma* are set. In the latter case, the training time even increased to more than 30 minutes.

4 Conclusion

In conclusion, despite the computational training time, as mentioned in the previous chapter, I preferred to use the SVM model with a polynomial kernel for the first exercise and the SVM model with an RBF kernel for the second exercise due to its performance and versatility.