Machine Learning, Homework 1 Report

Simone Palumbo 1938214, Sapienza University of Rome

2023-2024

Contents

Chapter 1

Overview

In the following report, I describe the task of devising solutions for two 10-class classification problem:

In Chapter 2 I describe the preprocessing technique used on the two datasets

In Chapter 3 I delineate with which method I solved the problems

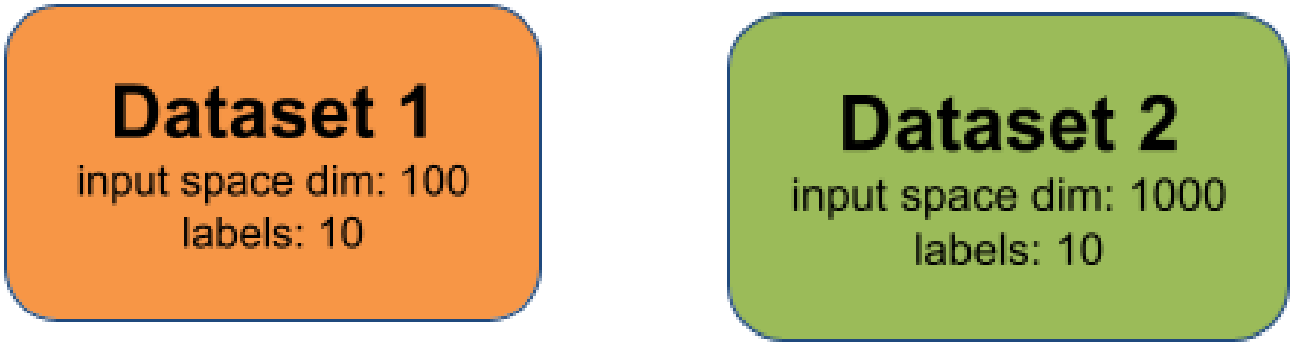
In Chapter 4 I show the results and compare the models in several configuration

Chapter 2

Preprocessing

2.1 Description of the Datasets

Both problems are 10-class classification problems, so both of our datasets present 10 labels. They differ in the dimension of the input space.



(a) The first dataset, the easier one

(b) The 2nd one, with 10x the features

Figure 2.1: Comparison of the two datasets

2.1.1 Data Loading

The *load_data* function does the job of reading the training dataset from the corresponding CSV file, eventually creating two pandas Dataframes: $\boldsymbol{X_{df}}$ (50000 × d, with d number of features) for the input values and $\boldsymbol{Y_{df}}$ (50000 × 1) for the labels.

2.1.2 Some Information and Statistics:

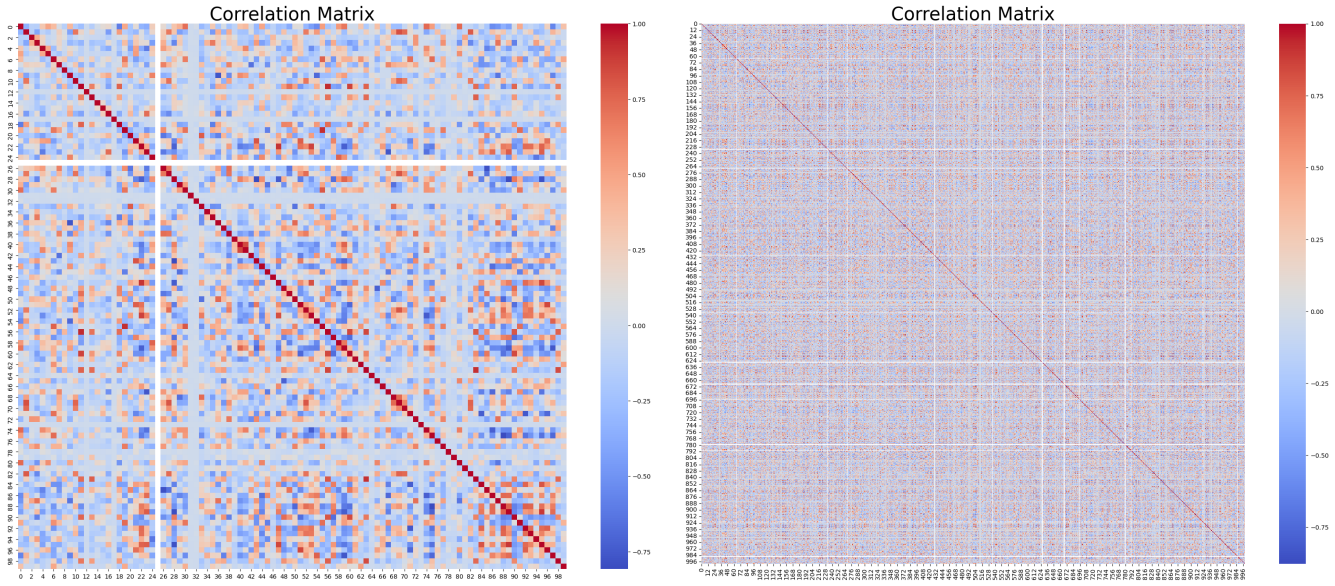
With pandas *info()* method, we can extract information about the Dataframes. The first Dataset has 100 float64 features, while the second one has 1000 float64 features. Not even one of the 50000 samples for both dataset has a null field. For each column of a Dataframe, we can use the *describe()* method to get some descriptive statistics about our dataset, such as mean, std, minimum and maximum value and the percentiles. These statistics can be useful to summarize the central tendency, dispersion and shape of a dataset's distribution. We will use mean and std in Section 2.2

2.1.3 Correlation:

A correlation matrix has the rows and columns correspond to the features, and each cell in the matrix represents the correlation between the variables. Correlation measures the strength and direction of the linear relationship between two variables. The correlation coefficient ranges from -1 to +1, having:

- +1 indicating a perfect positive linear relationship between two variables, meaning that as one variable increases, the other variable also increases proportionally.
- -1 indicating a perfect negative linear relationship between two variables, meaning that as one variable increases, the other variable decreases proportionally.
- values close to 0 indicating a weak linear relationship between the variables.

I thought it would be a good idea to plot the correlation matrices, even if the number of features is very high, to highlight any peculiarities of the dataset. Indeed, as shown in Figure 2.2, I found out that are some attributes with 0 correlation with all the other attributes. This features are constant and forms "white crosses" in the matrix plot. I will deal with that in Subsection 2.3.1.



(a) Correlation Matrix for Dataset 1

(b) Correlation Matrix for Dataset 2

Figure 2.2: 100 x 100 correlation matrix for Dataset 1. Red for positive correlation, blue for negative

2.1.4 Label Distribution

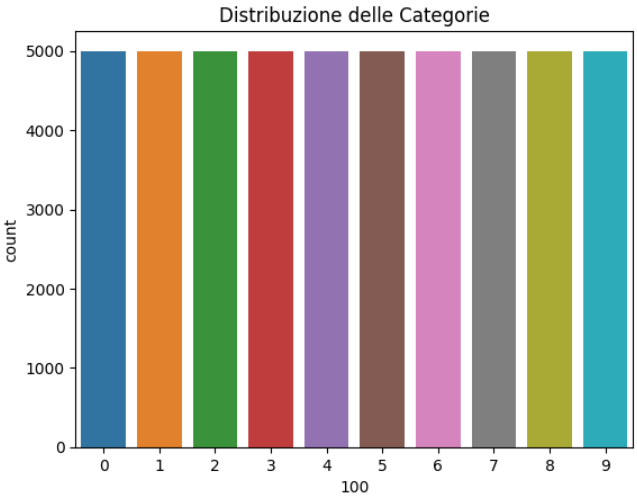
We can use Seaborn *countplot()* when we want to visualize the frequency of each category in a categorical data. In our case, I used it to find out that both datasets are equally distributed in terms of label, as shown in Figure 2.3. The fact that the datasets are balanced, implies that *accuracy is a good metrics for evaluation.*

2.2 Standardization

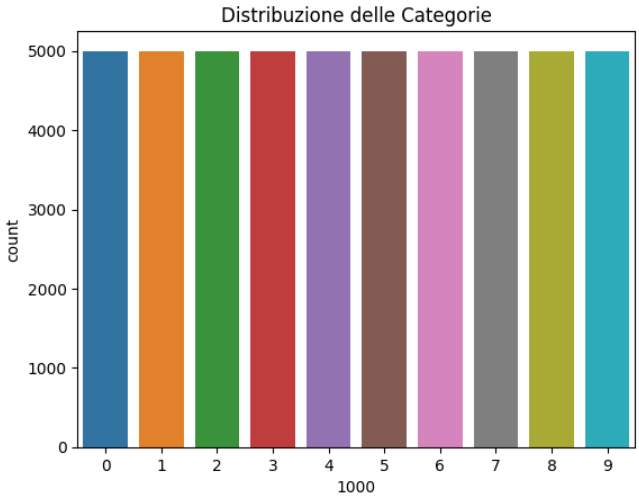
Standardization (or feature scaling) is a preprocessing technique. Many ML algorithms such as SVM, that we used in Chapter 4, perform better when the features are on a similar scale. It's important to note that not all algorithms require standardized features. For instance, tree-based models like Random Forests, used Chapter in 4, are less sensitive to feature scaling. However, it's generally considered good practice to standardize features. I used scikit-learn *StandardScaler()*, that performs the following standardization:

$$z = \frac{x - \text{mean}(x)}{\text{std}(x)}$$

Where x is a feature and mean(x) and std(x) are its mean and its standard deviation, respectively. We compute this formula for each x. After standardization, the scaled data will have features with mean 0 and unitary standard deviation.



(a) Label distribution for dataset 1



(b) Label distribution for dataset 2

Figure 2.3: These two histograms show us that the two datasets are balanced

2.3 Feature Selection:

To avoid the so-called *curse of dimensionality*, we do feature selection, that allows us to select a subset of relevant features for training the ML algorithm. In chapter 4 I will analyze how the performances changes using and not using feature selection. Specifically, I used two feature selection techniques: Constant Feature selection and Correlated Feature selection.

2.3.1 Constant Features:

The constant features, since they have the same value for all samples, do not provide any useful information for predictive modeling as they carry no variability within the data. To perform removal of constant feature, we use scikit-learn *VarianceThreshold()*, that removes all low-variance features below a certain threshold that we specify. If we specify a threshold that is > 0, we are doing a *quasi-constant features* dropping. In our case we remove only constant attributes, so our variance threshold will be 0. For dataset 1, we have just 1 constant attribute (the 25th). For dataset 2, 13 values, so 1.3% of the dataset in terms of attributes. Other experiments could have been conducted using different non-zero values for the variance threshold to determine the number of features that would have been dropped and to assess the potential changes in performance and execution time.

2.3.2 Highly (positively) correlated features:

Pearson correlation coefficient measures the linear relationship between two features, as we already saw in section 2.1.3. Feature selection so becomes selecting those features that are highly correlated with each other or with the target variable. In our case, I decided to compute the correlation only between features. Features highly correlated with the target variable are often considered important for predictive modeling, whereas pairs of features that are highly correlated with each other might indicate redundancy, and one of the correlated features could be removed. For doing so, we define a threshold value for the correlation coefficient. Features with correlation coefficients above this threshold are considered relevant and kept in the model, while others will be discarded. Note that *I decided to consider only high positive correlation*, since a negative correlation could be a useful information to build the machine learning algorithm, and so I prefer to not remove it. Other experiments would have been possible, verifying *how much the elimination of even highly negative features would have influenced execution time and performance.* To make an example, with 96% as the threshold I found out that

- for dataset 1, no attributes lies in the correlation range [96%, 100%], so no attribute should be removed with this technique.
- for dataset 2, 20 atttributes lies in the correlation range [96%, 100%], so 2% of the initial dataset.

Criticality of Feature Selection with Pearson coefficient:

It's important to note that Pearson correlation assumes a linear relationship between variables, so it might not capture nonlinear associations, and it's sensitive to outliers. Moreover, high correlation doesn't imply causation. Eventually, we have to consider that in datasets with a large number of features such as Dataset 2, using correlation for feature selection might not be efficient, as it doesn't account for complex relationships or interactions between variables, which might be better captured using more advanced feature selection techniques that are able to handle high-dimensional data. **This might be one of the reasons why the performance with the feature selection techniques I chose didn't improve significantly** compared to the performance without employing them, although we have to consider that all the models achieved notably high accuracy and F1 scores "effortlessly", making it challenging to substantially improve beyond those already high levels.

Chapter 3

Methods and Algorithms

3.1 Choice of the Model

I decided to use these three models:

- Support Vector Machine
- Random Forest
- GaussianNB

3.1.1 Hyperparameter Tuning:

Hyperparameter tuning is the process of selecting the optimal set of hyperparameters for a machine learning model. I used *Grid Search Cross-Validation* to see how much the performances would improve in certain situation. For instance, in Chapter 4 is reported the case of tuning some SVM hyperparameters for getting better results after a drastic feature dropping (924 + 13 features dropped over 1000).

3.2 Training Phase

3.2.1 K-Fold Cross Validation

I used *k-fold cross validation* to train, evaluate and validate our models. The process involves splitting the dataset into k equal-sized subsets or "folds." The model is trained and evaluated k times, using a different fold as the validation set each time while using the remaining k-1 folds for training. The performance metrics are calculated for each iteration using the validation set, and they are eventually averaged to obtain a more reliable estimation of the model's performance.

Why I used K-Fold: I used K-fold to maximize the use of available data, since all samples are used for both training and validation. K-fold also reduces the variance in the evaluation metrics compared to a single train-test split, and provides a more accurate estimate of how the model will perform on unseen data.

Value of k: Common choices for k in k-fold cross-validation are 5 or 10. I chose 5.

Chapter 4

Evaluation

4.1 Metrics Used

I used:

- Accuracy, that measures the proportion of correctly classified instances among the total number of instances.

$$Accuracy = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

  While accuracy provides an overall view of the model's performance, it might not be suitable for imbalanced datasets where the classes are not evenly distributed. Since in our case both datasets are balanced, accuracy is a suitable metric.

- Precision, that measures the accuracy of positive predictions. It is the ratio of correctly predicted positive observations to the total predicted positive observations.

$$Precision_{\ Class\ k} = \frac{\text{True Positives}_{\ Class\ k}}{\text{True Positives}_{\ Class\ k} + \text{False Positivesx}_{\ Class\ k}}$$

  Precision focuses on the relevance of the positive predictions. Note the subscript of Precision, since we are dealing with multiclass classification and we have to compute Precision for each class k = 0, ..., 9. To get a single value for Precision, we can make an average of the values for each class.

- Recall, that measures the ratio of correctly predicted positive observations to the total actual positives in the dataset.

$$Recall_{\ Class\ k} = \frac{\text{True Positives}_{\ Class\ k}}{\text{True Positives}_{\ Class\ k} + \text{False Negatives}_{\ Class\ k}}$$

  Recall is crucial when the cost of false negatives is high. It indicates how effectively the model can identify positive instances.

- F1-Score, that is the harmonic mean of precision and recall. It combines both precision and recall into a single metric.

$$F1\ Score = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

  F1 score provides a balance between precision and recall. It is particularly useful when you want to seek a balance between identifying positive instances correctly and minimizing false positives and false negatives.

Actually, *accuracy could be enough for our purposes*, since the dataset is balanced. Indeed, F1-score, considering precision and recall, is really useful when there is an imbalanced dataset, meaning one class significantly outnumbers the other, and so accuracy might not be the best metric to evaluate model performance.

4.2 Results

The performance are high and doesn't improve much using feature selection. This is probably due to the semplicity of the datasets and of the feature selection technique used, as previously discussed in Section 2.3.2. Despite these factors, I think it's worthwhile to investigate the aspect of feature selection, particularly to analyze the differences in execution times resulting from dataset reduction. The results have been rounded to the third decimal place.

4.2.1 Results for Dataset 1

Training on dataset 1 is in the order of seconds, with and without feature selection. Results are shown below. *The model with better performances is Support Vector Machine*, since, as you can see from Table 4.5, the performances remain the highest even when the dataset is drastically reduced[1]. Note that the tables that take into account feature selection with Pearson correlation also consider the 13 constant dropped features.

Table 4.1: Performances of the models for dataset 1, without feature selection

Model	Execution Time	Accuracy	Precision	Recall	F1
GaussianNB	1 sec	0.976	0.978	0.976	0.977
Random Forest	1 min	0.965	0.966	0.965	0.965
SVM	53 sec	0.987	0.987	0.987	0.987

Table 4.2: Effect of feature selection with Pearson correlation, using dataset 2

Correlation Treshold	Number of dropped features
90%	4
70%	42
50%	69

Table 4.3: Performances of Random Forest, using dataset 1

Correlation Treshold	Execution Time	Accuracy	Precision	Recall	F1
90%	1 min	0.942	0.951	0.942	0.941
70%	$\approx$ 1 min	0.956	0.959	0.956	0.955
50%	40 sec	0.944	0.945	0.944	0.943

Table 4.4: Performances of GaussianNB, using dataset 1

Correlation Treshold	Execution Time	Accuracy	Precision	Recall	F1
90%	1 sec	0.977	0.979	0.978	0.978
70%	$<$ 1 sec	0.973	0.974	0.973	0.973
50%	$<$ 1 sec	0.955	0.957	0.955	0.954

Table 4.5: Performances of SVM, using dataset 1

Correlation Treshold	Execution Time	Accuracy	Precision	Recall	F1
90%	1 min	0.987	0.987	0.987	0.987
70%	42 sec	0.987	0.987	0.987	0.987
50%	39 sec	0.987	0.987	0.987	0.987

[1] As I said, the results have been rounded to the third decimal place, so they are not actually the same value for all the correlation threshold chosen, as one might mistakenly think by looking at the table

4.2.2 Results for Dataset 2

Dataset 2, being 10 times bigger in terms of features, implies higher computation times. SVM and Random Forest have the highest computational times than GaussianNB. I tend to attribute this to the generation of Support Vectors in SVM training and the creation of trees during Random Forest training, that are computationally expensive. If we don't consider the high execution time has a malus for the evaluation, *SVM has, again, the best performances*, since the metrics remain high and stable even for low levels of correlation threshold, *indicating effective training on smaller subsets of the dataset*. Conversely, Random Forest and GaussianNB show decline in performances down to less than 90% for correlation threshold of 50%, as shown in Table 4.8 and 4.9.

Table 4.6: Performances of the models for dataset 2, without feature selection

Model	Execution Time	Accuracy	Precision	Recall	F1
GaussianNB	26 sec	0.968	0.969	0.968	0.968
Random Forest	8 min	0.942	0.943	0.942	0.942
SVM	28 min	0.968	0.968	0.968	0.968

Table 4.7: Effect of feature selection with Pearson correlation, using dataset 2

Correlation Treshold	Number of dropped features
90%	210
70%	830
50%	924

Table 4.8: Performances of Random Forest, using dataset 2

Correlation Treshold	Execution Time	Accuracy	Precision	Recall	F1
90%	6 min	0.956	0.956	0.956	0.956
70%	1 min	0.947	0.949	0.947	0.947
50%	37 sec	0.942	0.943	0.942	0.942

Table 4.9: Performances of GaussianNB, using dataset 2

Correlation Treshold	Execution Time	Accuracy	Precision	Recall	F1
90%	19 sec	0.968	0.969	0.968	0.968
70%	3 sec	0.954	0.957	0.954	0.954
50%	2 sec	0.882	0.892	0.882	0.878

Table 4.10: Performances of SVM, using dataset 2

Correlation Treshold	Execution Time	Accuracy	Precision	Recall	F1
90%	23 min	0.969	0.969	0.969	0.969
70%	4 min	0.972	0.972	0.972	0.972
50%	3 min	0.968	0.968	0.968	0.968

Hypertuning for SVM Using 50% dataset, I tried hypertuning to see how much the performances of SVM increases. Using Gridsearch with CV = 5 and the following hyperparameters:

- "C", the regularization, which I imposed can take value from [0.1, 1, 100]
- "kernel", the kernel function for SVM, with the possibility of being linear or the radial basis function (rbf)
- "gamma". This is a kernel coefficient used in radial basis function (as well as in other kernel function not covered here). I give gamma the possibility of being 'scale' ($\frac{1}{n\_features * X.var()}$) or 'auto' ($\frac{1}{n\_features}$).

I get that {'C': 100, 'gamma': 'scale', 'kernel': 'rbf'} is the best set of parameters, getting accuracy: 0.969, 0.057 points above the accuracy get without hypertuning. The time of execution was about 30 minutes, which highlights how Gridsearch is computationally expensive.