Regression and Classification

Machine Learning

Instituto Superior Técnico

Ivan Figueiredo | 93386
Maria Costa | 93399

Group 36

November 16, 2021

I. Part 1

I.1. Problem 1

$$\mathcal{T}_r = \{(x^{(1)}, y^{(1)}), ..., (x^{(100)}, y^{(100)})\}$$

$$x^{(i)} \in \mathbb{R}^{20}$$

$$y^{(i)} \in \mathbb{R}$$

In this problem, we were asked to train predictors $\hat{y} = f(x)$. Firstly, we started by importing the data from the two files that were provided. In this way, the array `X` and `Y` received, respectively, the data from the file that contains the train set of the variable x and y; the data associated to the test train of x was assigned to `Xeval`.

After that, we proceeded to center the train set data using the command `np.mean()`, in such a way that

$$x'^{(i)} = x^{(i)} - \overline{x} \qquad\qquad y'^{(i)} = y^{(i)} - \overline{y}$$

in which $x'^{(i)}$ and $y'^{(i)}$ correspond to the centered data and $x^{(i)}$ and $y^{(i)}$ to the noncentered data.

Secondly, the command `train_test_split()` was used in order to split the provided data (`X` and `Y`) into four arrays: `Xtrain`, `Xtest`, `Ytrain` and `Ytest`. We chose `test_size = 0.20`, which means that 20% of the data in `'Xtrain_Regression_Parte1.npy'` and `'Ytrain_Regression_Parte1.npy'` was utilized in the testing set (`Xtest` and `Ytest`), and 80% of the data in the training set (`Xtrain` and `Ytrain`).

Then, we started by training three predictive linear models: linear regression, Ridge regression and Lasso regression. These three methods consist of a linear regression with parameter β, differing in the way this value is calculated, as is visible with the following equations.

$$\textbf{Linear regression:} \qquad \hat{\beta}_{\text{linear}} = \arg\min_{\beta} \|y - X\beta\|_2^2 \tag{1}$$

$$\textbf{Ridge regression:} \qquad \hat{\beta}_{\text{Ridge}} = \arg\min_{\beta} \|y - X\beta\|_2^2 + \alpha\|\beta\|_2^2 \tag{2}$$

$$\textbf{Lasso regression:} \qquad \hat{\beta}_{\text{Lassso}} = \arg\min_{\beta} \|y - X\beta\|_2^2 + \alpha\|\beta\|_1 \tag{3}$$

The commands from the `scikit-learn` library that were used to train the predictive models were the following ones, which correspond to estimators: `LinearRegression()`, `linear_model.Lasso()`

and `linear_model.Ridge()`.

After that, we plotted the cross validation score as a function of α, the parameter which controls the penalty in Lasso and Ridge regression models.

	Minimum MSE	Maximum score
Linear regression	0.016258	**0.998681**
Ridge regression	0.016263	0.998681
Lasso regression	0.016559	0.998686

Table 1: Minimum mean squared error (MSE) and maximum score values obtained for each used predictive model

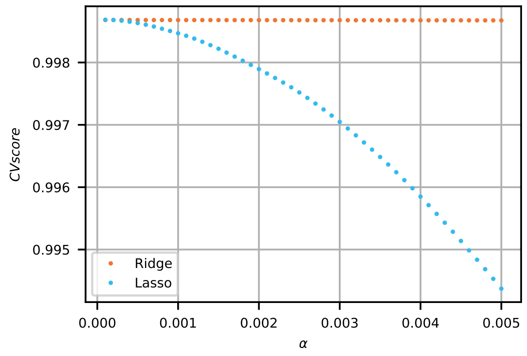

Figure 1 – Score for Lasso and Ridge regressions

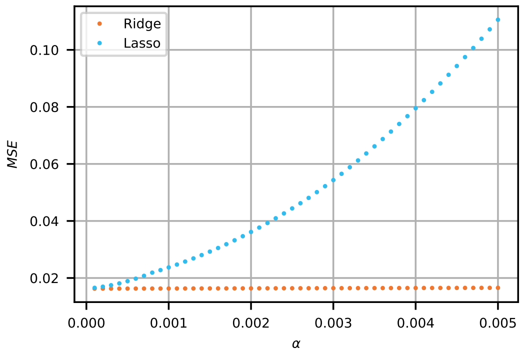

Figure 2 – Mean Squared Error for Lasso and Ridge regressions

In the table 1, the minimum values of MSE obtained for each regression model are presented, as well as the maximum score for the Ridge and Lasso regressions. Comparing the values of MSE, we have chosen the linear regression as the best model to predict the `Ytest` set.

Next, and using the entire provided dataset (instead of spliting the data into `Xtrain`, `Ytrain`, `Xtest` and `Ytest`), the chosen regression model was used in order to compute the $\hat{\beta}'$ vector that best fitted the data (`X`, `Y`). The $\hat{\beta}'$ vector is the solution of the following equation:

$$y' = \overline{x'}^T \beta'$$

However, the $\hat{\beta}'$ is associated to the centered data, but the original data was noncentered. In such manner, we had to invert the pre-processing, which was made when the data was centered by using the command `np.mean()`. The $\hat{\beta}$ vector can be obtained from the $\hat{\beta}'$ vector by doing

$$\hat{\beta} = [\hat{\beta}_0 \;\; \hat{\beta}'^T], \quad \text{where } \hat{\beta}_0 = \overline{y} - \overline{x'}^T \hat{\beta}'.$$

The $\hat{\beta}'$ vector that we obtained in this problem is printed below.

```
[[-1.65926404e-02  2.17327582e-02  5.52578871e-04  7.80913919e-02
   3.31357433e-01 -6.81661406e-01  1.69721957e+00  4.98734472e-02
   1.81267959e+00  7.78111741e-03 -1.44714336e-02 -1.45742338e+00
  -7.07007047e-01  3.27382534e-02 -6.17384811e-01  6.92794585e-03
  -3.76264812e-01 -1.29490509e-01 -1.36598513e+00 -1.27365158e+00
   9.58563665e-01]]
```

Since the X matrix used in the model

$$\nabla\left[\mathrm{SSE}(\beta)\right] = 0 \quad \Leftrightarrow \quad \nabla\Big[\, ||y - X\beta|| \,\Big] = 0$$

needs to have the following shape,

$$X = \begin{bmatrix} 1 & x_1^{(1)} & \cdots & x_{20}^{(1)} \\ 1 & x_1^{(2)} & \cdots & x_{20}^{(2)} \\ \vdots & \vdots & \cdots & \vdots \\ 1 & x_1^{(100)} & \cdots & x_{20}^{(100)} \end{bmatrix}, \tag{4}$$

we had to add a column with '1s' at the beginning of the matrix `Xeval`, obtaining the `X_final` matrix.

Finally, with the $\hat{\beta}$ vector, which is associated to the noncentered data, it was possible to obtain the prediction to the `Yeval` vector:

$$\texttt{Yeval}^{(i)} = \texttt{X\_final}^{(i,:)}\, \hat{\beta},$$

in which $\texttt{X\_final}^{(i,:)}$ denotes the i-th row of the matrix `X_final`.

I.2. Problem 2

Regarding this problem, we were asked to first identify outliers, given the information that there was a contamination rate smaller than 10%, and then proceed to train the data set as in the previous problem. Also, we are supposed to train one or more predictors valid for the majority of the examples.

In order to identify and exclude the outliers, points generated by a different model than the remaining ones, we experimented with different methods as **Z-score**, **Interquartile method**, **Isolation Forest method**, **Local Outlier Factor method**, **SVM One Class** and **Mahalanobis Distance**. In spite the complexity of the latter, the results obtained after the data was fitted were not satisfactory, namely the mean squared error value, score used to evaluate the results.

As we did in the problem 1, we started by importing the provided data and splitting it into the four arrays already mentioned before, considering `test_size` $= 0.20$: `Xtrain`, `Ytrain`, `Xtest` and `Ytest`.

I.2.1. Z-score

This outlier detection method assumes that the provided data follows a Gaussian pattern. In R^p, the probability density function $p(\mathbf{x})$ of the normal distribution is given by

$$p(\mathbf{x}) = A\ \exp\left\{ -\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\, C^{-1}\, (\mathbf{x}-\boldsymbol{\mu}) \right\},$$

where A is a constant, $\boldsymbol{\mu} \in \mathbb{R}^p$ is the mean vector and C is the covariance matrix.

Using the command `np.std(X, axis = 0)`, we computed the standard deviation `X_Std` for each component $x^{(i)}$, with $i \in [1,\ 20]$. Then, for each one of those components, we calculated the vector $\mathtt{z}^{(i)}$ with the following instruction:

$$\mathtt{z} = \mathtt{X}[:, i]\ /\ \mathtt{X\_Std}[i],$$

which corresponds to a normalization of the columns of the matrix `X`.

Defining a threshold equal to 2.5, the data points selected as outliers were those ones which verified the condition

$$\mathtt{z} > \mathtt{threshold}.$$

I.2.2. Interquartile method

The interquartile range (IQR) corresponds to the difference between the 75th and the 25th percentiles of the data [1]. Defining limits on the sample values that are a factor k below the 25th percentile or above the 75 percentile, it is possible to use the IQR method as an outlier detection method, since this way is removes the data that is outside that interval. As suggested in [1], we used $k = 1.5$.

I.2.3. Mahalanobis Distance method

The Mahalanobis distance is a measure of the distance between a point P and a distribution D [2]. This distance works with multivariate data, since it uses the covariance matrix C of the variables to compute the distance between data points and the center [2]. The covariance matrix C works as a measure of the way that variables variate together.

The Mahalanobis distance between two n-dimensional points A and B can be computed in the following way:

$$D^2_{\text{Mahalanobis}} = (X_A - X_B)^T \, C^{-1} \, (X_A - X_B),$$

where X_A and X_B are, respectively, the coordinates of A and B in the n-dimensional space.

For each one of the 20 variables $x^{(i)}$, we had to calculate the center point with the command `np.mean()` and compute the covariance matrix C between $\texttt{Xtrain}^{(i,:)}$, which denotes the $i-th$ row of the matrix `Xtrain`, and the vector `Ytrain`. Then, we computed the Mahalanobis distance between each point of the form $(\texttt{Xtrain}^{(i,j)}, \texttt{Ytrain}^{(j)})$ and the center of the distribution of the variable $x^{(i)}$. These distances were saved in the `distances` array.

This method to remove outliers requires the use of a cutoff from the Chi-Square distribution [2]. In such wise, the indexes of the points that correspond to outliers can be found with the following command:

```
outlierIndexes = np.where(distances > cutoff),
```

i.e, we only want to select the points whose distance to the center of the distribution is less or equal to the selected cutoff.

Now we present some of the obtained results. In the figures below, the points located outside the drawn ellipse are considered as outliers, i.e, points that do not follow the same pattern as the others.

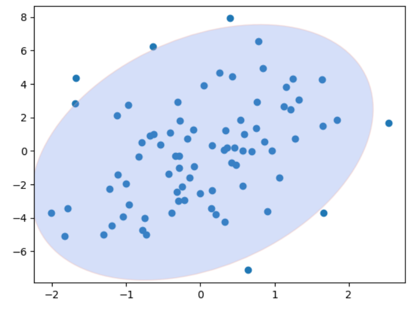

Figure 3 – Variable x_5

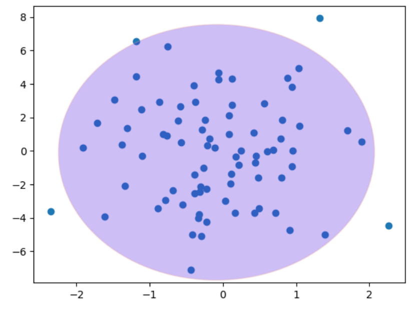

Figure 4 – Variable x_{10}

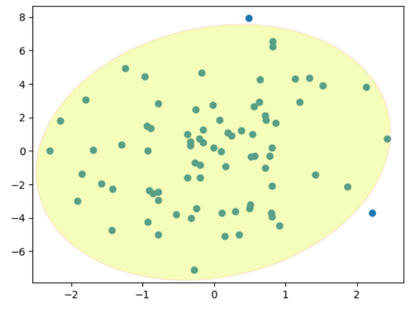

Figure 5 – Variable x_{16}

I.2.4. Isolation Forest, Local Outlier Factor and SVM One Class methods

The Isolation Forest method isolates observations by selecting, in a random way, a feature and then selecting a split value between the minimum and maximum values of that feature. The `contamination` variable corresponds to the proportion of outliers in the data set. In each iteration of a `for` cycle, the value of this variable assumed particular value, whose maximum was 0.1, since the given information said that there was a contamination rate smaller than 10%.

The Local Outlier Factor method measures the local deviation of the density of a particular sample with respect to the samples that are its neighbors. In a similar way to the Isolation Forest method, this method also uses the parameter `contamination`, which controls the proportion of outliers in the data set.

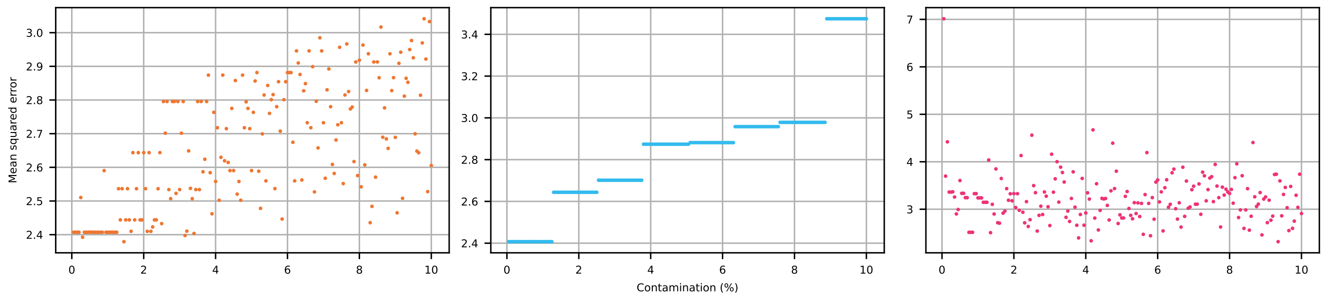

Figure 6 – Mean Squared Error for Isolation Forest, Local Outlier and SVM One Class methods, respectively

I.2.5. The selected method to remove the outliers

These methods did not performed as expected which lead us, given that the data set was not too large, to compute the squared error of each point regarding a linear regression, in order to observe the ones that most stood out. Three points yield significantly larger values for the error computed and were then eliminated from the data set.

We then proceeded to train the new data set with the following predictive models, as we did in the problem 1: linear regression, Ridge regression and Lasso regression, as we were not satisfied with the results of the first part. Comparing the cross validation score of these three models, we selected the Lasso regression model as the best predictor to be used in the provided data, as it presented the best values regarding that particular metric, even though the linear regression presented a inferior value of the mean squared error.

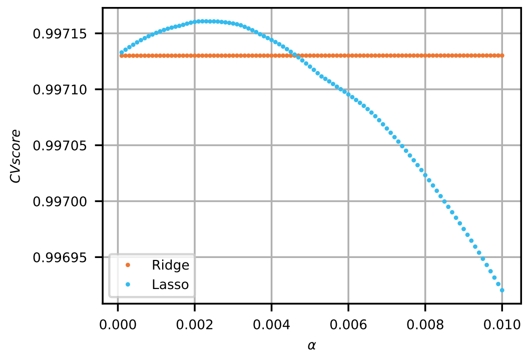

Figure 7 – Score for Lasso and Ridge regressions

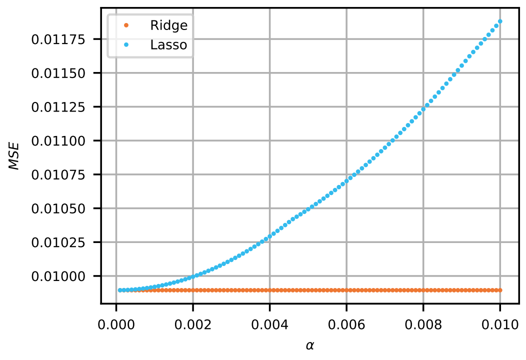

Figure 8 – Mean Squared Error for Lasso and Ridge regressions

II. Part 2

The second part of this project consists of image classification with labels previously provided by the assignment. There also two different problems to this part, the first one regarding identification by gender and the second by ethnicity.

II.1. Task 1

Regarding this problem, we were asked to classify a set of 1164 images with respect to the gender of the people in them. The labels to attribute were 0 (male) or 1 (female).

In order to do that we had to train a model that took the images, more specifically their features, and predicted the corresponding labels. There are different types of algorithms that receive images as inputs and predict a label based on their features.

Once again, we started by dividing the data set into training and validation sets, with a splitting ratio of 0.8 to 0.2 correspondingly.

The first approach we took was to build a Convolutional Neural Network (CNN from now on), with the `keras` libraries avaailable for *python*. These networks consist basically of three different layers:

- Convolutional Layers;
- Pooling Layers;
- Connection Layers.

The first 2 types of layers were stacked a total of 3 times, and only then connected. Since there were only two possible labels, the number of output neurons was set to 1.

The activation function used was ReLU in every layer except for the output one, in which we chose sigmoid activation.

The compiling phase of the network allows for the configuration of the metrics to be used, namely a loss, and an optimizer. Given the data characteristics we chose the binary cross entropy loss, fit to binary applications and tried two different optimizers: Adam [3] and SGD [4].

The role of these algorithms is to change the attributes of your neural network such as weights and learning rate in order to reduce the loss, being therefore important to choose the appropriate one. After analysing the results obtained with the two, we choose to work with Adam as it provided the best results.

Training this model requires a not so small number of epochs, and it's computationally expensive, being therefore necessary to have a metric that, applied to the validation set, would give us an idea on the progress and also a stopping criteria. With that in mind, we chose to compute the

Binary Accuracy, which simply counts how many times the predictions match the binary labels and divides it by the total size of the set in hand, leaving us with a probability of a correct prediction.

We then trained the model for a total of 1000 epochs, while the early stopping mechanism had a patience of 50, meaning that it'd stop if the metric didn't improve in that number of epochs. Doing several runs, the algorithm would usually converge after a total of roughly 60 epochs, being the value 72 for the final run.

The following plots better demonstrate the evolution of the discussed network.

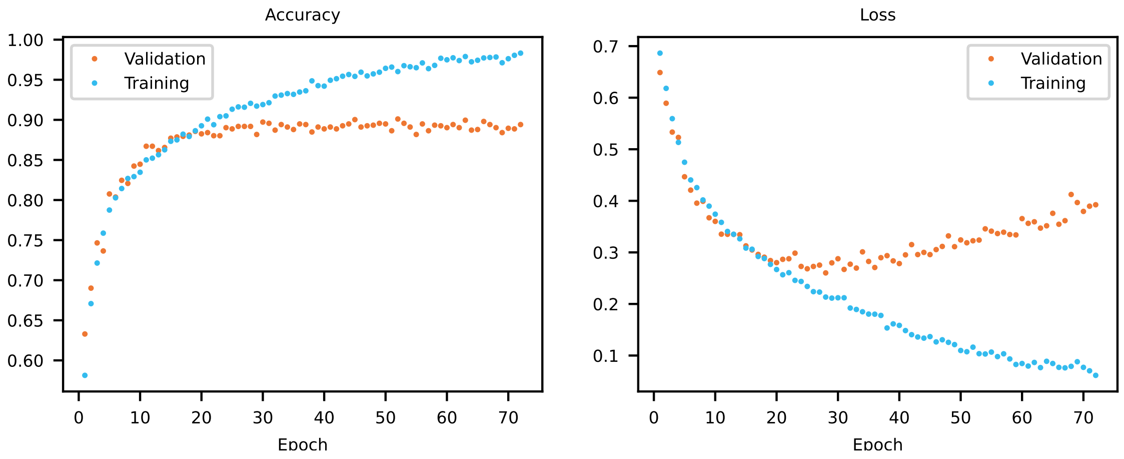

Figure 9 – Binary Accuracy and Loss

An aspect worth noting is the increasing loss values for the validation set, in contrast with the accuracy, that reaches a stagnation value after around 30 epochs. This can mean that the algorithm is performing on overfitting the training set, despite the measures taken to prevent that.

To prevent the overfitting of the training set features and as the data set was not so large, we further implemented two methods, kernel regularization and random dropout.

The former one consists of applying a penalty to the layer's kernel, based on the L2 regularization process and the latter involves the random setting of inputs to zero, with a rate we set up to 0.2. We placed two different dropout layers, both on hidden layers, after the max pooling.

Given the trained model, we then predicted a vector of labels based on the training set images', with which we computed the value of the Balanced Accuracy Score (BACC), yielding a value of 0.896.

Another method of image classification is based on the Multi Layer Perceptron Classifier, an architecture of multiple weighted units organised into similar layers, in contrast with the previous method in which the layers had diverse purposes.

We again chose ReLU as the activation function, and analysed both Adam and SGD as solvers for weight optimization.

However this method proved to yield worst accuracy results, even after many epochs, making us turn our attention solely to the CNN.

Given the trained CNN model we then proceeded to predict the final vector of binary labels. After submission and some problem fixing due to a innocent computational error, we could obtain with the teacher a value of 0.880 for the BACC.

II.2. Task 2

Still on the topic of image classification, the second task was to classify a set of 1290 images regarding the person's ethnicity in each of them, given the 4 labels: 0 - Caucasian, 1 - African, 2 - Asian and 3 - Indian.

The provided vector of labels consisted on these four integer values which we began to convert to one hot encoding format, having therefore 4 classes each one with a binary label, converting the label data set to a 7366 x 4 matrix.

Given the work done on the previous task, we built a similar CNN as it performed better with the previous data sets.

The only difference in structure with respect to the previous regards the number of output neurons, now set to 4 and the addition of a layer that performs random horizontal flips of the images, a process known as data augmentation, giving the model more images to train with. Other methods of data augmentation were tried but didn't reveal better results.

Regarding the activation functions, we used the ReLU as before on every layer except for the output one, on which we chose the softmax function, as it yield better results.

To calculate the loss in each epoch, we chose the categorical cross entropy function, properly fit to multiple class data sets and accordingly the evaluation metric was changed to categorical accuracy. The optimizer used was the same, the Adam algorithm.

Onto the training part, with a total of 1000 epochs and a early stopping with 50 epoch patience, during the several performed runs the algorithm would converge after a total around of 120 epochs.

The following plots show the evolution of loss and accuracy metrics with each epoch.

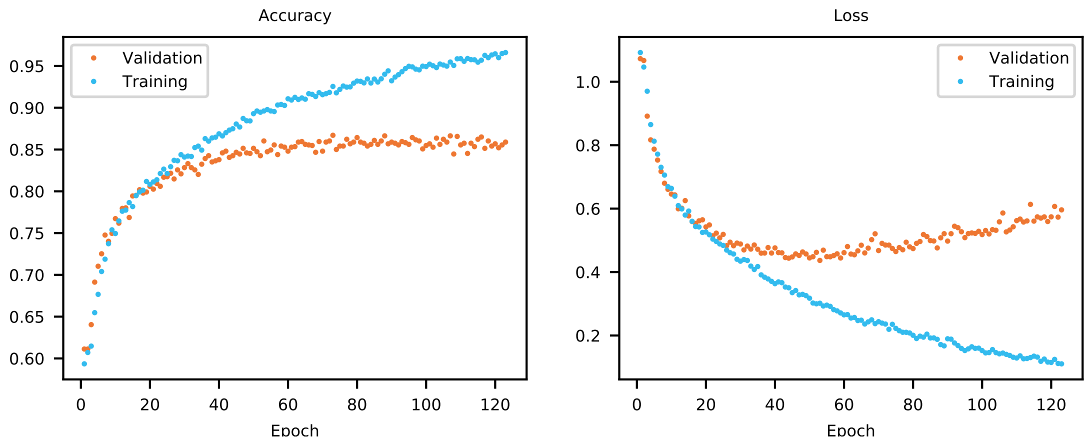

Figure 10 – Categorical Accuracy and Loss

Again we note the increase of loss value with increasing epoch, an aspect we were unable to fix since the previous task. The value of BACC obtained for the validation set was 0.752.

Given the trained model, we then used it to, with the evaluation data set, predict the corresponding labels which we then uncoded from their matrix form into a vector with integer values from 0 to 3. After submission, we obtained a value of 0.766 for BACC metric.

References

[1] How to remove outliers for machine learning. `https://machinelearningmastery.com/how-to-use-statistics-to-identify-outliers-in-data/`, 2021.

[2] Multivariate outlier detection in python. `https://towardsdatascience.com/multivariate-outlier-detection-in-python-e946cfc843b3`, 2021.

[3] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[4] Bottou L. Online algorithms and stochastic approximations, 1998.