TÉCNICO LISBOA

Machine Learning

Instituto Superior Técnico

1º Semester - 2021/2022

MEEC

Final Report

Group Nº72:
nº 84095 João Martins
nº 90817 Vasco Araújo

November 2021

Contents

Introduction

For this semester, the Machine Learning project was divided into two parts. The first on was on regression and the second one on classification

1 Regression Part 1

For the first part of the Regression problem we were given a training set with 100 examples. The goal was to train a predictor that could generalize well the data enough to still perform well with unseen data.

The training data was a *Numpy* array with 100 examples (lines) and 20 columns for each.

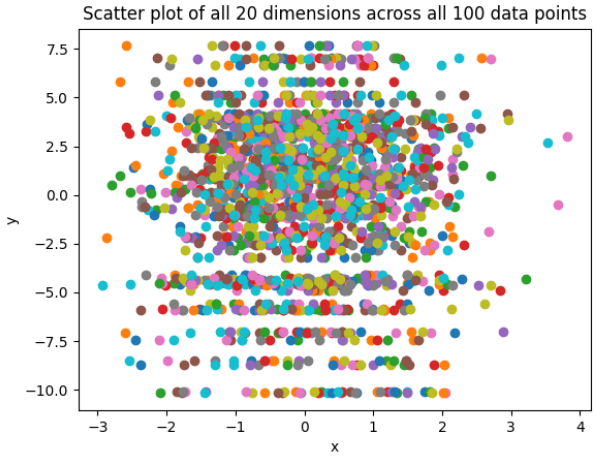

Figure 1: Scatter plot of all columns

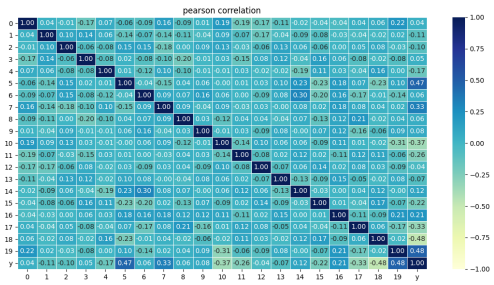

Figure 2: Correlation matrix of all columns

As we can see from Figures 1 the data points don't appear to form a coherent pattern, which is confirmed with the results from Figure 2, where no single column has a high correlation with the target y. This means that the best model will have to take into account every data points available to compute the regression that fits the best.

So, in order to fit the best model, we tried three different kinds of predictors from *sklearn*: Linear, Lasso and Ridge.

The Linear regression is the most basic one. It computes a linear predictor that tries to minimize the residual sum of squares between train data and the target points. While it is very simple, it is also very effective and the best way to start experimenting on a regression problem.

The Lasso model adds complexity to the simple Linear regression by adding a weight penalization to the coefficients of the model (on this case, the columns). This weight penalization is controlled by a new hyperparameter, α. The objective goal for minimizing during the training is:

$$\frac{1}{2 * n} * \sum_{i=1}^{n} ||y - Xw|| + \alpha * ||w|| \tag{1}$$

The Ridge model is close to Lasso, but has a L2 regularization instead of the L1 that Lasso has. This means that the weights are squared and the objective goal for minimizing during the training becomes:

$$\frac{1}{2*n} * \sum_{i=1}^{n} ||y - Xw|| + \alpha * ||w|| \tag{2}$$

For training the three models, we added K-Fold cross-validation. This is a method used to test how the model is generalizing to unseen data and avoid overfitting. This method starts by dividing the training data into K number of groups. Then, one of the K groups is used as a test group and the remaining groups are used for training. The model is then evaluated on the (unseen) test group. This process is repeated K times such that the test group is always a new one, and the accuracy of the model is computed over the mean of all the repetitions. The metrics to evaluate the accuracy is the mean squared error.

We also tried to compute the best value of α by repeating the training process for M times with different values for α each time and record the best ones.

The results for this problem can be observed on Table 1.

Table 1: Results for all models considered for regression part 1

Model	Best α	MSE
Linear	-	0.015434868882439745
Lasso	0.03397	0.015656467874114982
Ridge	0.18	0.01447720775362391

2 Regression Part 2

The second problem was identical to the first one with the offset that there are some outliers on the data, less than 10% to be more exact. However, we don't know the exact number of outliers.

In order to best deal with this, we first tried to use Inter Quartile Range to detect the outliers. The Inter Quartile range is simply the difference of the 75_{th} percentile (Q3) and the 25_{th} percentile (Q1). This can be used to detect outliers on the dataset by determining a threshold, and claiming that every point bigger that (threshold*IQR)+Q3 or smaller than Q1-(threshold*IQR). This method did not perform very well finding outliers, possibly because the outliers were not points that were very distant from the inliers but were probably mixed with the data, as we can see from Figure 3 that the dataset is very concentrated.

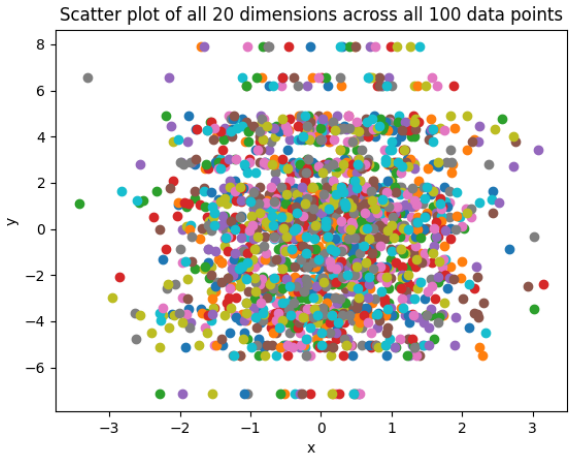

Figure 3: Scatter plot of all columns

Next we tried to use the Isolated Forests method to detect the outliers. It works by, as the name suggests, trying to isolate each point. After randomly selecting a data point, it tries recursively to isolate it from the nearer points by randomly selecting a split value. In doing this it is essentially creating a tree structure where the longer the path from the root node each point is, the harder it was to isolate and the more likely it is an inlier. This method then creates a tree-like structure from each point of the dataset. Since this method depends from the percentage of outliers on the dataset and that number was unknown to us, we iterated with several outliers between 0 and 10% to find the one that gave us the best results.

After the outliers selected with Isolated Forests were removed from the dataset, the method was the same as in the Part 1 of the Regression Problem. We iterated the same three models, Linear, Lasso and Ridge.

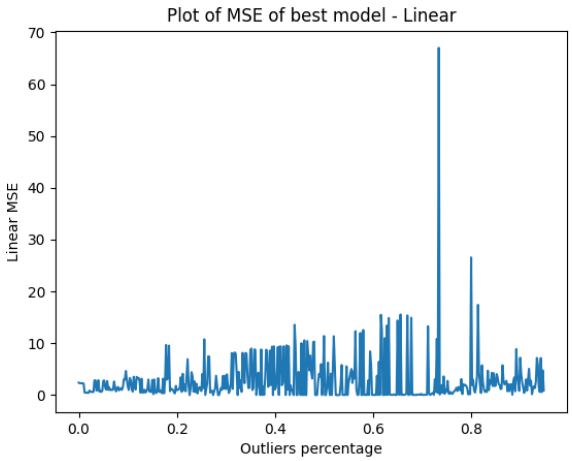

Figure 4: Linear MSE

In Figure 4 we can see that the Linear MSE oscillates a lot regarding the outliers percentage number.

The best model was the Linear one, and the percentage of outliers as a parameter of Isolation Forests on *sklearn* that gave us the best results was 0.526, with a Linear MSE of 0.007564418508519184 on the test data (20% of all data).

3 Classification Part 1

As previously said, the second part of the project was on classification. For the first problem, the goal was to predict the gender of the person given an unseen image. The dataset used was from UTKFace. The training set consists of 7300 grayscale images of 50x50 pixels.

In order to perform such a task, we decided to use a Deep Learning Model. We tried with two types of DL models: a Multilayer Perceptron and a Convolution Neural Network. On further testing, the Convolution Neural Network proved to be the best model.

Convolutional Neural Networks are models know to be very effective to detect features on images because they are very powerful at encoding complex features and discover patterns hidden on the images.

In order to deal with the relatively low quantity of data, we did data augmentation by 2x. The data augmentation was done using a random rotation and an horizontal flip.

Even after the data augmentation, the model was still overfitting, so we removed some layers from the model until we were left with just three layers: two convolutional layers and a fully connected layer to transform all features into a single output, which we called ClassifyNet. This model is shown in greater detail on Table 2.

Table 2: ClassifyNet

Layer	Parameters
Conv1	Conv2d(1, 10, kernel_size=(5, 5), stride=(1, 1))
Conv2	Conv2d(10, 20, kernel_size=(5, 5), stride=(1, 1))
Dropout	Dropout(p=0.2, inplace=False)
Fc1	(fc1): Linear(in$_features = 1620, out_features = 1, bias = True$)

On Table 3 there are the hyperparameters chosen for our model.

Table 3: Hyperparameters for our model

Batch size	Learning rate	SGD momentum
64	0.001	0.9

This model was able to generalize well the training data and didn't overfit much, which was the main problem we encountered on this specific problem, probably due to low quantity of training data.

4 Classification Part 2

In this problem, we trained 3 different models. We used a Multi-Layer Perceptron, a Convolutional Neural Network and a Random Forest.

Before training the models we could observe that the data was imbalanced. There were many more samples with label '0' than the others. To tackle this issue, a set of weights for the labels was calculated to compensate for this lack of balance. The class '0' was penalized with a smaller weight and the other classes, in special class '1', were offered a superior weight.

The first model we aproached was the multi-layer perceptron (MLP). A MLP is a class of feed-forward artificial neural network. It consists of at least three layers of nodes (an input layer, a hidden layer and an output layer), in our case we have 2 hidden layers making a total of 4 layers. The summary of the model we trained can be seen in Figure 5. We trained the model with and without early stopping and the training and validation plots can be seen in Figs. 6 and 7.

Layer (type)	Output Shape	Param #
dense_41 (Dense)	(None, 625)	1563125
dense_42 (Dense)	(None, 500)	313000
dense_43 (Dense)	(None, 4)	2004

Total params: 1,878,129
Trainable params: 1,878,129
Non-trainable params: 0

Figure 5: Graph plot of the training and test losses for the **Summary of the MLP model trained**

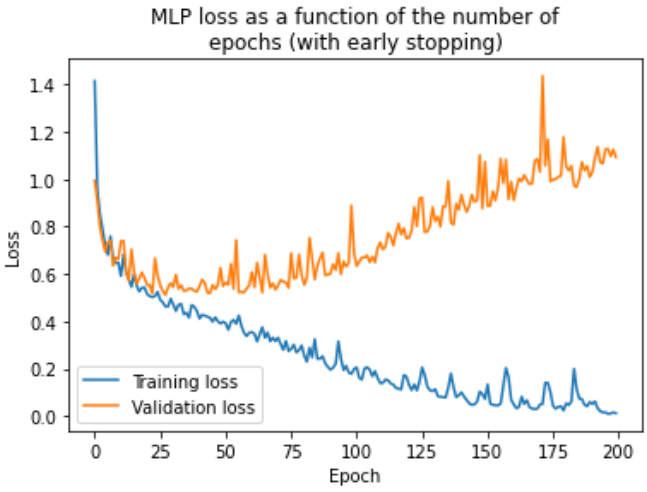

Figure 6: Graph plot of the training and test losses for the **MLP model without early stopping**

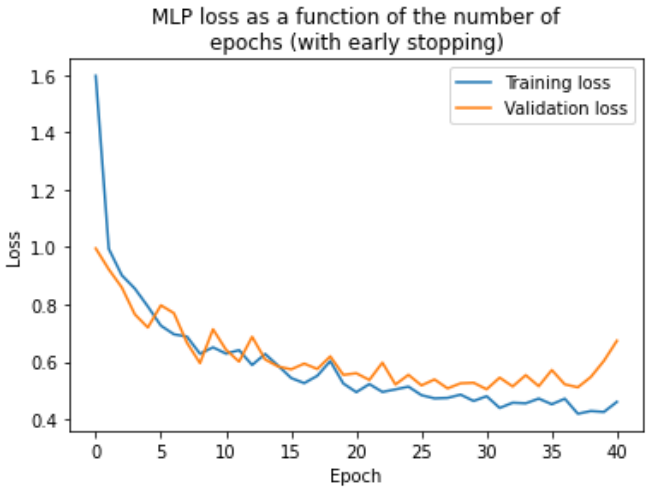

Figure 7: Graph plot of the training and test losses for the **MLP model with early stopping**

The model trained with Early Stopping achieved an accuracy score of 85,16%.

We also trained a Random Forest with 200 estimators using the Gini Index impurity measure which revealed not to be the best option for this problem since it achieved only 66,7% accuracy

score in its predictions.

The model that was chosen was the Convolutional Neural Network (CNN). A CNN can account for local connectivity. Its weights are smaller and shared, it's easier and less wasteful to train when compared to the MLP. It's often more effective, its layers are sparsely connected and can take matrices as well as vectors as inputs.

The network was comprised of a 2D convolutional layer with output size 10 and kernel size 5, then a MaxPool 2D layer of size 2, then a 2D convolutional layer with output size 20 and kernel size 5, then a MaxPool 2D layer of size 2, then a dense layer of size 1620 and an output layer of size 4 with softmax activation. All the other layers had Relu activation. The model summary can be seen in Figure 8. Early stopping was used in this problem and the training and validation loss plots can be seen in Figure 9.

Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 46, 46, 10)	260
max_pooling2d_6 (MaxPooling2	(None, 23, 23, 10)	0
conv2d_7 (Conv2D)	(None, 19, 19, 20)	5020
max_pooling2d_7 (MaxPooling2	(None, 9, 9, 20)	0
flatten_3 (Flatten)	(None, 1620)	0
dense_44 (Dense)	(None, 1620)	2626020
dense_45 (Dense)	(None, 4)	6484

Total params: 2,637,784
Trainable params: 2,637,784
Non-trainable params: 0

Figure 8: **Summary of the CNN model trained**

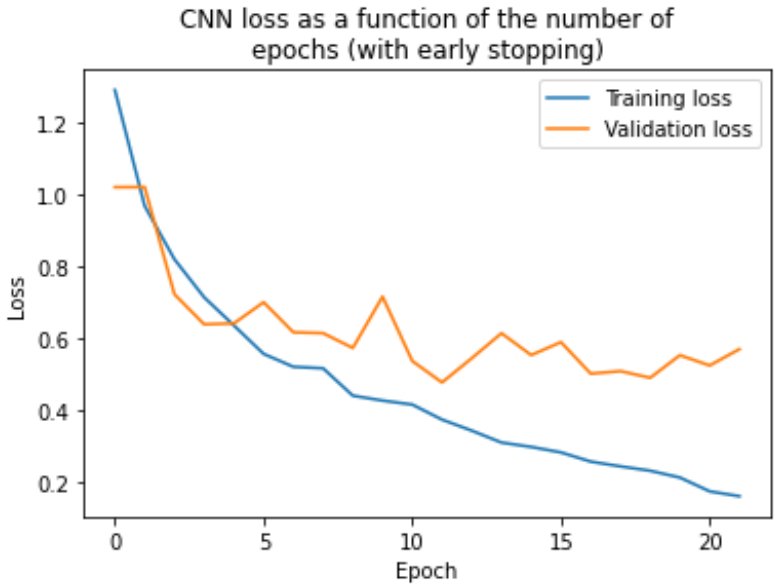

Figure 9: Graph plot of the training and test losses for the **CNN model with early stopping**

The model trained achieved an accuracy score of 88,7% and thus was the best model and the one chosen to solve this classification problem.