Aprendizagem 2022

Homework III – Group 019

Diogo Gaspar 99207, Rafael Oliveira 99311

Part I: Pen and paper

1. **Consider the basis function, $\phi_j(x) = x^j$, for performing a 3-order polynomial regression,**

$$\hat{z}(x, w) = \sum_{j=0}^{3} w_j \phi_j(x) = w_0 + w_1 x + w_2 x^2 + w_3 x^3.$$

Learn the Ridge regression (l_2 regularization) on the transformed data space using the closed-form solution with $\lambda = 2$.

We have in hands a **supervised learning** problem, with a given training dataset as shown below:

	y_1	z
x_1	0.8	24
x_2	1	20
x_3	1.2	10
x_4	1.4	13
x_5	1.6	12

Table 1: Training dataset: y_1 as the input's (only) variable, z as the target variable

We can note that in the statement's estimation function, $\hat{z}(x, w)$, x is a single-element vector (with its only entry being each sample's y_1 value). Therefore, it makes sense to "expand" the table above as follows, in order to have a broader representation of the values we'll end up using in the estimation function:

	y_1	y_1^2	y_1^3	z
x_1	0.8	0.64	0.512	24
x_2	1	1	1	20
x_3	1.2	1.44	1.728	10
x_4	1.4	1.96	2.744	13
x_5	1.6	2.56	4.096	12

Table 2: Training dataset with additional information

The equation below shows the closed-form solution for the Ridge regression problem, with $\lambda = 2$:

$$w = (\Phi^T \Phi + \lambda I)^{-1} \Phi^T z = (\Phi^T \Phi + 2I)^{-1} \Phi^T z$$

Here, Φ is the result of applying the basis function to our training dataset's inputs, such that:

$$\Phi = \begin{bmatrix} 1 & \phi_1(x_1) & \phi_2(x_1) & \phi_3(x_1) \\ 1 & \phi_1(x_2) & \phi_2(x_2) & \phi_3(x_2) \\ \vdots & \vdots & \vdots & \vdots \\ 1 & \phi_1(x_5) & \phi_2(x_5) & \phi_3(x_5) \end{bmatrix} = \begin{bmatrix} 1 & 0.8 & 0.64 & 0.512 \\ 1 & 1 & 1 & 1 \\ 1 & 1.2 & 1.44 & 1.728 \\ 1 & 1.4 & 1.96 & 2.744 \\ 1 & 1.6 & 2.56 & 4.096 \end{bmatrix}$$

We are now able to learn the given polynomial regression model, with $\lambda = 2$:

$$(\Phi^T\Phi + \lambda I)^{-1} = \left(\begin{bmatrix} 1 & 0.8 & 0.64 & 0.512 \\ 1 & 1 & 1 & 1 \\ 1 & 1.2 & 1.44 & 1.728 \\ 1 & 1.4 & 1.96 & 2.744 \\ 1 & 1.6 & 2.56 & 4.096 \end{bmatrix}^T \begin{bmatrix} 1 & 0.8 & 0.64 & 0.512 \\ 1 & 1 & 1 & 1 \\ 1 & 1.2 & 1.44 & 1.728 \\ 1 & 1.4 & 1.96 & 2.744 \\ 1 & 1.6 & 2.56 & 4.096 \end{bmatrix} + \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix} \right)^{-1}$$

$$= \begin{bmatrix} 0.34168753 & -0.1214259 & -0.07490231 & -0.00932537 \\ -0.1214259 & 0.3892078 & -0.09667718 & -0.07445624 \\ -0.07490231 & -0.09667718 & 0.37257788 & -0.17135047 \\ -0.00932537 & -0.07445624 & -0.17135047 & 0.17998796 \end{bmatrix}$$

$$\Phi^T z = \begin{bmatrix} 1 & 0.8 & 0.64 & 0.512 \\ 1 & 1 & 1 & 1 \\ 1 & 1.2 & 1.44 & 1.728 \\ 1 & 1.4 & 1.96 & 2.744 \\ 1 & 1.6 & 2.56 & 4.096 \end{bmatrix}^T \begin{bmatrix} 24 \\ 20 \\ 10 \\ 13 \\ 12 \end{bmatrix} = \begin{bmatrix} 79 \\ 88.6 \\ 105.96 \\ 134.392 \end{bmatrix}$$

$$w = (\Phi^T\Phi + \lambda I)^{-1}\Phi^T z = \begin{bmatrix} 7.0450759 \\ 4.64092765 \\ 1.96734046 \\ -1.30088142 \end{bmatrix}$$

Having learned the regression model, we can now use it to predict labels z for new samples!

2. **Compute the training RMSE for the learnt regression model.**

3. **Consider a multi-layer perceptron characterized by one hidden layer with 2 nodes. Using the activation function** $f(x) = e^{0.1x}$ **on all units, all weights initialized as 1 (including biases), and the half squared error loss, perform one batch gradient descent update (with learning rate** $\eta = 0.1$**) for the first three observations (0.8), (1) and (1.2).**

Part II: Programming and critical analysis

The code utilized to answer the following questions is available in this report's appendix.

4. **Compute the MAE of the three regressors: linear regression, MLP_1 and MLP_2.**

   We opted to utilize `sklearn`'s `mean_absolute_error` function to compute the MAE of the three regressors. The regressors were created as shown in the appendix (using `Ridge` and `MLPRegressor` with the respective parameters).

   We gathered the following results:

Regressor	MAE
Linear Regression (Ridge)	0.16283
MLP_1	0.06804
MLP_2	0.09781

Table 3: Gathered Mean Absolute Errors for each specified regressor

5. **Plot the residues (in absolute value) using two visualizations: boxplots and histograms.**

   Each regressor's residues, calculated as the absolute difference between the predicted and actual values, were plotted using both boxplots and histograms (using, respectively, `seaborn`'s `boxplot` and `histplot` functions), as shown in this report's appendix (figures after the code).

6. **How many iterations were required for MLP_1 and MLP_2 to converge?**

   Calling the `print_regressor` method for each regressor shows us not only the MAE, but also the number of iterations required for each of the MLP regressors to converge. In this case, the number of iterations required for MLP_1 (MLP with early stopping) to converge was 452, while MLP_2 (MLP *without* early stopping) required only 77.

7. **What can be motivating the unexpected differences on the number of iterations? Hypothetize one reason underlying the observed performance differences between the MLPs.**

Appendix

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.io.arff import loadarff
from sklearn.linear_model import Ridge
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error
from sklearn.neural_network import MLPRegressor
sns.set_style('darkgrid')

# Load data
data = loadarff('data/kin8nm.arff')
df = pd.DataFrame(data[0])

X = df.drop('y', axis=1).values
y = df['y'].values

X_train, X_test, y_train, y_test = train_test_split(
  X, y,
  test_size=0.3,
  random_state=0
)

def predict(regressor):
  regressor.fit(X_train, y_train)
  return regressor.predict(X_test)

def plot_regressor_residues(regressor, description, y_pred):
  """Utilized for answering question 2."""
  residues = np.abs(y_test - y_pred)

  fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))
  sns.histplot(data=residues, ax=ax1)
  ax1.set_title('Residues histogram')
  ax1.set_xlabel('Residues')
  sns.boxplot(data=residues, ax=ax2, orient='h')
  ax2.set_title('Residues boxplot')
  ax2.set_xlabel('Residues')
  plt.suptitle(description)
  plt.show()

def print_regressor(regressor, description, y_pred):
  """Utilized for answering questions 1. and 3."""
  print(description)
  print('MAE: {:.5f}'.format(mean_absolute_error(y_test, y_pred)))
  if "MLP" in description:
    print('Iterations: {}'.format(regressor.n_iter_))

regressors = {
  "Ridge Regression": Ridge(alpha=0.1),
  "MLP 1": MLPRegressor(
    hidden_layer_sizes=(10, 10),
    activation='tanh',
```

```
    max_iter=500,
    random_state=0,
    early_stopping=True
  ),
  "MLP 2": MLPRegressor(
    hidden_layer_sizes=(10, 10),
    activation='tanh',
    max_iter=500,
    random_state=0
  )
}

for description, regressor in regressors.items():
  y_pred = predict(regressor)
  print_regressor(regressor, description, y_pred)
  plot_regressor_residues(regressor, description, y_pred)
```

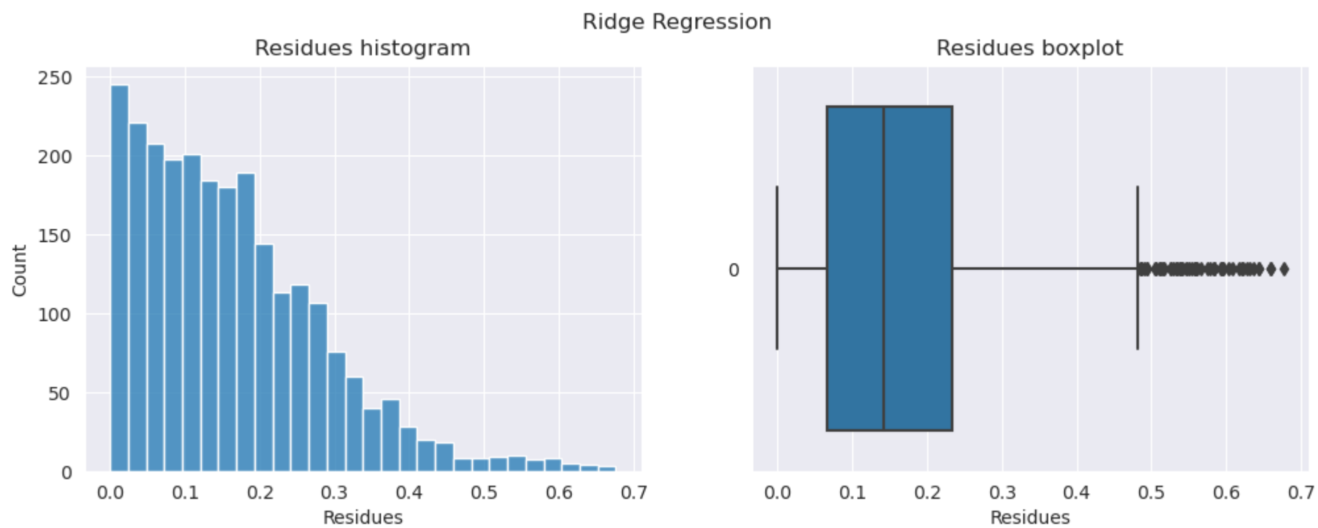

Figure 1: Ridge regression's residue plotting

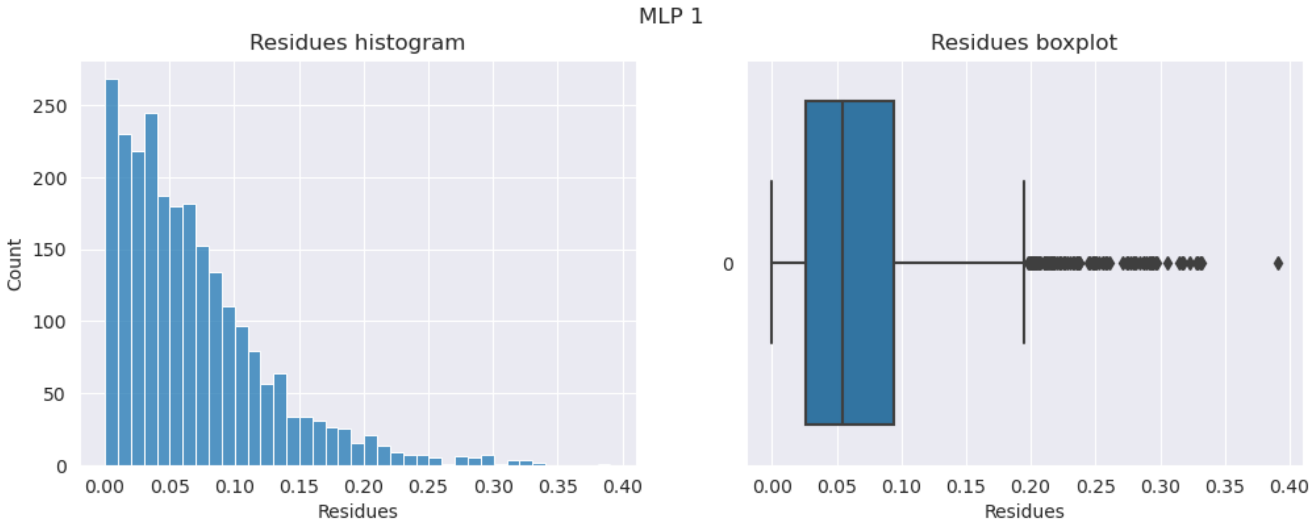

Figure 2: MLP_1 regression's residue plotting

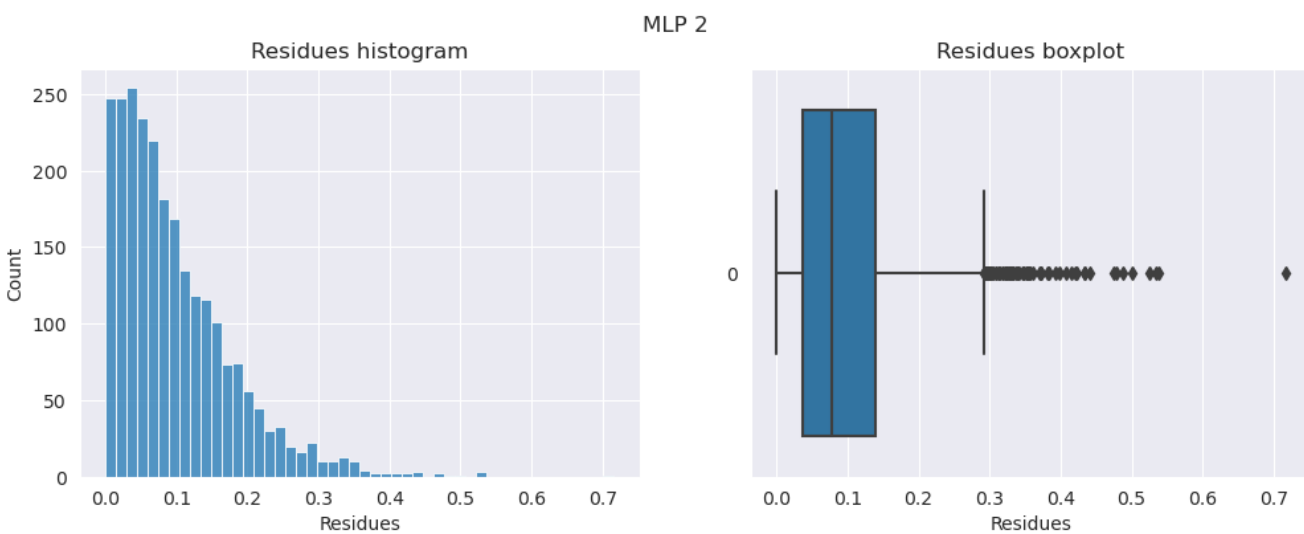

Figure 3: MLP_2 regression's residue plotting