

Aprendizagem

Instituto Superior Técnico

outubro de 2023

Homework 3 - Report

Joana Pimenta (103730), Rodrigo Laia (102674)

Pen and Paper

1. a) Uma função radial basis permite mapear observações para um novo espaço baseando-se na distância entre as observações e os centróides.

$$\phi_j(x) = \exp\left(-\frac{\|\vec{x} - c_j\|^2}{2}\right) \quad (1)$$

Cálculo dos vetores transformados:

$$\vec{\phi}_i = \left(\exp\left(-\frac{\|\vec{x}_i - c_1\|^2}{2}\right), \exp\left(-\frac{\|\vec{x}_i - c_2\|^2}{2}\right), \exp\left(-\frac{\|\vec{x}_i - c_3\|^2}{2}\right) \right) \quad (2)$$

Assim os vetores transformados obtidos foram:

$$\phi_1 = (0.74826, 0.74826, 0.10127)$$

$$\phi_2 = (0.81465, 0.27117, 0.33121)$$

$$\phi_3 = (0.71177, 0.09633, 0.71177)$$

Para fazer regressão de Ridge é necessário minimizar a função de erro:

$$E(\vec{w}) = \frac{1}{2} \sum_{i=1}^n (z_i - \vec{w}^T \cdot x_i)^2 + \frac{\lambda}{2} \|\vec{w}\|^2 \quad (3)$$

Sendo que isso é equivalente a calcular \vec{w} através da seguinte fórmula:

$$\vec{w} = (X^T \cdot X + \lambda \cdot I)^{-1} \cdot X^T \cdot \vec{z} \quad (4)$$

Uma vez que estamos a trabalhar com uma transformação de espaços, é necessário calcular a matriz transformada Φ colocando para cada linha um 1 na primeira

coluna e depois o vetor transformado de cada observação. Utilizamos então as fórmulas acima com Φ no lugar de X , assumindo que após a transformação a relação entre as variáveis e o target é linear.

Cálculos intermédios:

$$\Phi = \begin{bmatrix} 1 & 0.74826 & 0.74826 & 0.10127 \\ 1 & 0.81465 & 0.27117 & 0.33121 \\ 1 & 0.71177 & 0.09633 & 0.71177 \\ 1 & 0.88250 & 0.16122 & 0.65377 \end{bmatrix}$$

$$\Phi^T = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0.74826 & 0.81465 & 0.71177 & 0.88250 \\ 0.74826 & 0.27117 & 0.09633 & 0.16122 \\ 0.10127 & 0.33121 & 0.71177 & 0.65377 \end{bmatrix}$$

$$(\Phi^T \cdot \Phi + \lambda \cdot I)^{-1} \cdot \Phi^T = \begin{bmatrix} 0.14105 & 0.35022 & 0.35575 & -0.30185 \\ -0.09064 & 0.43823 & -0.50361 & 0.53370 \\ 0.99394 & -0.50615 & -0.13690 & -0.16477 \\ -0.31222 & -0.65246 & 0.72647 & 0.42436 \end{bmatrix}$$

$$\vec{w} = \begin{bmatrix} 0.33914 \\ 0.19945 \\ 0.40096 \\ -0.29600 \end{bmatrix}$$

Assim, a regressão de Ridge obtida foi:

$$\hat{z} = 0.33914 + 0.19945 \cdot \phi_1 + 0.40096 \cdot \phi_2 - 0.29600 \cdot \phi_3$$

b) Para calcular o RMSE (root mean square error) foi utilizada a seguinte fórmula:

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (z_i - \hat{z}_i)^2} \quad (5)$$

$$\hat{z}_i = \vec{w}^T \cdot \vec{\phi}_i \quad (6)$$

Targets estimados:

$$\hat{z}_1 = 0.75843$$

$$\hat{z}_2 = 0.51231$$

$$\hat{z}_3 = 0.30905$$

$$\hat{z}_4 = 0.38629$$

Assim, o RMSE obtido foi:

$$RMSE = 0.06508$$

2. É importante referir que para este exercício se utilizou a seguinte notação: $L_{observation}^{label}$
As fórmulas utilizadas foram:

$$\vec{x}^{[p]} = \phi(\vec{W}^{[p]} \cdot \vec{x}^{[p-1]} + \vec{b}^{[p]})$$

$$\vec{\delta}^{[p]} = \frac{\partial E}{\partial \vec{x}^{[p]}} \circ \frac{\partial \vec{x}^{[p]}}{\partial \vec{z}^{[p]}} = (\vec{x}^{[p]} - \vec{t}) \circ \phi'(\vec{z}^{[p]}), \text{ para a última layer}$$

$$\vec{\delta}^{[p]} = \left(\frac{\partial \vec{z}^{[p+1]}}{\partial \vec{x}^{[p]}} \right)^T \cdot \vec{\delta}^{[p+1]} \circ \frac{\partial \vec{x}^{[p]}}{\partial \vec{z}^{[p]}} = (\vec{W}^{[p+1]})^T \cdot \vec{\delta}^{[p+1]} \circ \phi'(\vec{z}^{[p]}), \text{ para as outras layers}$$

$$\vec{W}^{[p]} = \vec{W}^{[p]} - \eta \cdot \frac{\partial E}{\partial \vec{W}^{[p]}} = \vec{W}^{[p]} - \eta \cdot \vec{\delta}^{[p]} \cdot (\vec{x}^{[p-1]})^T$$

$$\vec{b}^{[p]} = \vec{b}^{[p]} - \eta \cdot \frac{\partial E}{\partial \vec{b}^{[p]}} = \vec{b}^{[p]} - \eta \cdot \vec{\delta}^{[p]}$$

Estas expressões são válidas para squared error loss function.

Dados necessários para começar o algoritmo:

$$\vec{x}_1^{[0]} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, t_1 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \vec{x}_2^{[0]} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ -1 \end{bmatrix}, t_2 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

$$W^{[1]} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}, b^{[1]} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, W^{[2]} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 4 & 1 \end{bmatrix}, b^{[2]} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, W^{[3]} = \begin{bmatrix} 1 & 1 \\ 3 & 1 \\ 1 & 1 \end{bmatrix}, b^{[3]} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

$$\phi = \tanh(0.5x - 2), \phi' = 0.5 * (1 - \tanh^2(0.5x - 2))$$

POR GUIA DE REDE!!!!!!!

Primeiro é necessário realizar (forward) propagation para obter os valores das observações:

- Para a primeira observação:

$$z_1^{[1]} = W^{[1]} \cdot x_1^{[0]} + b^{[1]} = \begin{bmatrix} 5 \\ 6 \\ 5 \end{bmatrix} \Rightarrow x_1^{[1]} = \phi(z_1^{[1]}) = \begin{bmatrix} 0.46212 \\ 0.76159 \\ 0.46212 \end{bmatrix}$$

$$z_1^{[2]} = W^{[2]} \cdot x_1^{[1]} + b^{[2]} = \begin{bmatrix} 2.68583 \\ 4.97061 \end{bmatrix} \Rightarrow x_1^{[2]} = \phi(z_1^{[2]}) = \begin{bmatrix} -0.57642 \\ 0.45048 \end{bmatrix}$$

$$z_1^{[3]} = W^{[3]} \cdot x_1^{[2]} + b^{[3]} = \begin{bmatrix} 0.87406 \\ -0.27878 \\ 0.87406 \end{bmatrix} \Rightarrow x_1^{[3]} = \phi(z_1^{[3]}) = \begin{bmatrix} -0.9159 \\ -0.97266 \\ -0.9159 \end{bmatrix}$$

- Para a segunda observação:

$$z_2^{[1]} = W^{[1]} \cdot x_2^{[0]} + b^{[1]} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \Rightarrow x_2^{[1]} = \phi(z_2^{[1]}) = \begin{bmatrix} -0.90515 \\ -0.90515 \\ -0.90515 \end{bmatrix}$$

$$z_2^{[2]} = W^{[2]} \cdot x_2^{[1]} + b^{[2]} = \begin{bmatrix} -1.71545 \\ -4.43089 \end{bmatrix} \Rightarrow x_2^{[2]} = \phi(z_2^{[2]}) = \begin{bmatrix} -0.99343 \\ -0.99956 \end{bmatrix}$$

$$z_2^{[3]} = W^{[3]} \cdot x_2^{[2]} + b^{[3]} = \begin{bmatrix} -0.992996 \\ -2.979861 \\ -0.992996 \end{bmatrix} \Rightarrow x_2^{[3]} = \phi(z_2^{[3]}) = \begin{bmatrix} -0.98652 \\ -0.99814 \\ -0.98652 \end{bmatrix}$$

Depois é necessário realizar (backward) propagation dos erros da última camada para a primeira:

- Para a primeira observação:

$$\delta_1^{[3]} = (x_1^{[3]} - t_1) \circ \phi'(z_1^{[3]}) = \begin{bmatrix} -0.07379 \\ -0.05320 \\ -0.07379 \end{bmatrix}$$

$$\delta_1^{[2]} = (W^{[3]})^T \cdot \delta_1^{[3]} \circ \phi'(z_1^{[2]}) = \begin{bmatrix} -0.10255 \\ -0.08001 \end{bmatrix}$$

$$\delta_1^{[1]} = (W^{[2]})^T \cdot \delta_1^{[2]} \circ \phi'(z_1^{[1]}) = \begin{bmatrix} -0.07179 \\ -0.08874 \\ -0.07179 \end{bmatrix}$$

- Para a segunda observação:

$$\delta_2^{[3]} = (x_2^{[3]} - t_2) \circ \phi'(z_2^{[3]}) = \begin{bmatrix} -0.026596 \\ -0.001854 \\ -0.01321 \end{bmatrix}$$

$$\delta_2^{[2]} = (W^{[3]})^T \cdot \delta_2^{[3]} \circ \phi'(z_2^{[2]}) = \begin{bmatrix} -2.9697 \cdot 10^{-4} \\ -1.8157 \cdot 10^{-5} \end{bmatrix}$$

$$\delta_2^{[1]} = (W^{[2]})^T \cdot \delta_2^{[2]} \circ \phi'(z_2^{[1]}) = \begin{bmatrix} -2.8473 \cdot 10^{-5} \\ -3.3395 \cdot 10^{-5} \\ -2.8473 \cdot 10^{-5} \end{bmatrix}$$

Por fim é necessário atualizar os pesos e os bias. Como estamos a realizar um batch gradient descent update (with learning rate 0.1), a expressão para os pesos atualizados é dada por:

$$W^{[p]} = W^{[p]} - \eta \cdot (\delta_1^{[p]} \cdot (x_1^{[p-1]})^T + \delta_2^{[p]} \cdot (x_2^{[p-1]})^T)$$

$$W^{[1]} = W^{[1]} - 0.1 \cdot (\delta_1^{[1]} \cdot (x_1^{[0]})^T + \delta_2^{[1]} \cdot (x_2^{[0]})^T) = \begin{bmatrix} 1.0071818 & 1.00717895 & 1.00717895 & 1.00717611 \\ 1.00887759 & 1.00887425 & 2.00887425 & 1.00887091 \\ 1.0071818 & 1.00717895 & 1.00717895 & 1.00717611 \end{bmatrix}$$

$$b^{[1]} = b^{[1]} - 0.1 \cdot (\delta_1^{[1]} + \delta_2^{[1]}) = \begin{bmatrix} 1.0071818 \\ 1.00887759 \\ 1.0071818 \end{bmatrix}$$

$$W^{[2]} = W^{[2]} - 0.1 \cdot (\delta_1^{[2]} \cdot (x_1^{[1]})^T + \delta_2^{[2]} \cdot (x_2^{[1]})^T) = \begin{bmatrix} 1.00471224 & 1.00778345 & 1.00471224 \\ 1.00369595 & 4.00609219 & 1.00369595 \end{bmatrix}$$

$$b^{[2]} = b^{[2]} - 0.1 \cdot (\delta_1^{[2]} + \delta_2^{[2]}) = \begin{bmatrix} 1.01028494 \\ 1.00800323 \end{bmatrix}$$

$$W^{[3]} = W^{[3]} - 0.1 \cdot (\delta_1^{[3]} \cdot (x_1^{[2]})^T + \delta_2^{[3]} \cdot (x_2^{[2]})^T) = \begin{bmatrix} 0.99310456 & 1.00066557 \\ 2.99674951 & 1.00221106 \\ 0.99443459 & 1.00200382 \end{bmatrix}$$

$$b^{[3]} = b^{[3]} - 0.1 \cdot (\delta_1^{[3]} + \delta_2^{[3]}) = \begin{bmatrix} 1.01003842 \\ 1.00550496 \\ 1.00869959 \end{bmatrix}$$

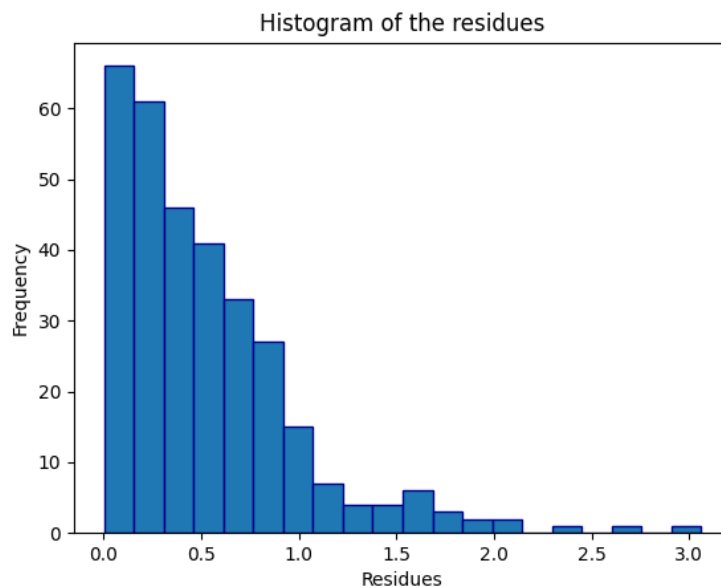
Programming - Código Python e Resultados Obtidos

1. Utilizamos um training-test split de 80-20, ou seja 80% dos dados foram utilizados para treinar o modelo e 20% para testá-lo.

De modo a minimizar o erro devido à aleatoriedade do algoritmo, o output final considerado foi a média de 10 outputs de 10 regressores MLP estocásticos com sementes de 0 a 9. Os resíduos foram calculados através da seguinte fórmula:

$$\text{residual} = |z - \hat{z}| \quad (7)$$

O histograma dos resíduos encontra-se representado na seguinte figura:



Observando o histograma, conclui-se que a frequência diminui significativamente com o crescimento do valor absoluto dos resíduos. Isto significa que a maior parte da diferença entre o valor real e o valor previsto pelo MLP é pequena.

Código Utilizado:

```
1 import pandas as pd, numpy as np
2 import matplotlib.pyplot as plt
3 from sklearn.model_selection import train_test_split
4 from sklearn.neural_network import MLPRegressor
5
6 # Reading the csv file
7 df = pd.read_csv('Homework3/winequality-red.csv')
8 # Separating the variables from the target
9 variables = df.drop("quality", axis= 1)
10 target = df['quality']
11
12 # Training Test Split
13 variables_train, variables_test, target_train, target_test =
    train_test_split(variables, target,
```

```

14         train_size=0.8, stratify=target, random_state=0)
15
16 y_pred = np.zeros(len(target_test))
17
18 # Average the mlp regressor
19 for i in range(1, 11):
20     # Learn the MLP regressor
21     mlp = MLPRegressor(hidden_layer_sizes=(10,10), activation='relu',
22                         early_stopping=True, validation_fraction=0.2, random_state=i)
23     #Predict output
24     y_pred += mlp.fit(variables_train, target_train).predict(
25                     variables_test)
26
27 y_pred = y_pred/10
28 first_rmse = np.sqrt(np.mean((target_test - y_pred)**2))
29
30 ##### Exercise 1 #####
31
32 # Calculate the residues
33 residues = abs(target_test - y_pred)
34 # Plot the residues
35
36 plt.hist(residues, edgecolor='darkblue', bins=20)
37 plt.title('Histogram of the residues')
38 plt.xlabel('Residues')
39 plt.ylabel('Frequency')
40 plt.savefig('ex1_histogram.png')
41 plt.show()

```

2. Uma vez que sabemos que a qualidade do vinho tem de ser um inteiro entre 1 e 10 podemos arredondar os valores previstos pelo MLP para o inteiro mais próximo e transformar os valores menores que 1 em 1 e os maiores que 10 em 10. Comparando os valores do erro médio absoluto (MAE) antes e depois deste processo conclui-se que o MAE antes (0.51167) é maior do que o novo (0.45625). Assim, podemos dizer que este processo melhora o desempenho do MLP.

Código Utilizado:

```

1
2 ##### Exercise 2 #####
3
4 # Round and bound the predictions
5 rounded_predictions = np.round(y_pred)
6 min_value = 1
7 max_value = 10
8 rounded_and_bounded_predictions = np.clip(rounded_predictions,
9                                           min_value, max_value)
10
11 # Calculate previous MAE
12 mae = np.mean(abs(target_test - y_pred))
13 #Calculate new MAE
14 mae_new = np.mean(abs(target_test - rounded_and_bounded_predictions))

```

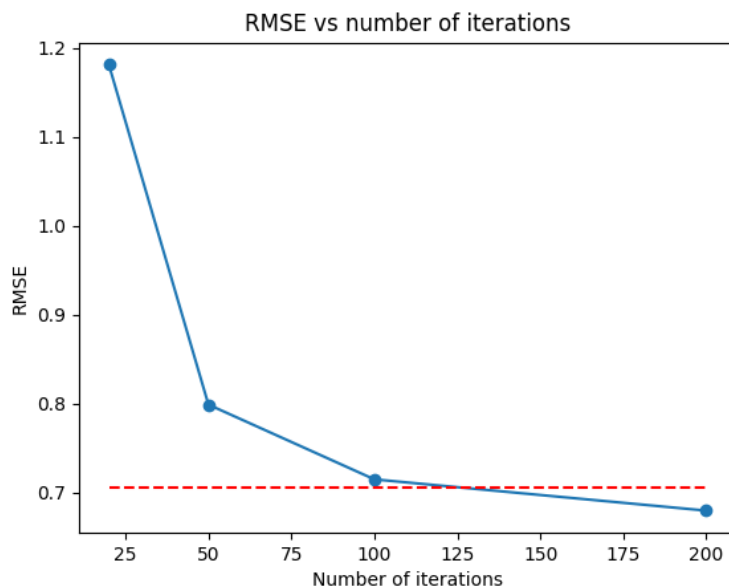


```

15 print('The previous MAE is: ', mae)
16 print('The new MAE is: ', mae_new)
17
18 if mae_new < mae:
19     print('The new MAE is lower than the previous one')
20 elif mae_new > mae:
21     print('The new MAE is higher than the previous one')
22 else:
23     print('The new MAE is equal to the previous one')

```

3. Até agora foram considerados MLP's com early stopping. Neste exercício vamos ver qual é o efeito do número máximo de iterações no desempenho do MLP, avaliado através do RMSE. Assim foram considerados 4 MLP's com números máximos de iterações iguais a 20, 50, 100 e 200. Da mesma maneira que antes, o modelo final considerado foi a média de 10 modelos com sementes de 0 a 9. No gráfico seguinte encontra-se representado o RMSE em função do número máximo de iterações.



Observando o gráfico, conclui-se que o RMSE diminui significativamente com o aumento do número máximo de iterações. No entanto, essa diminuição abranda à medida que se aumenta o número máximo de iterações.

Código utilizado:

```

1 # Calculate the RMSE for each number of iterations
2 rmse_final = []
3 iter_array = [20,50,100,200]
4 for iter in iter_array:
5     y_pred = np.zeros(len(target_test))
6     for i in range(1, 11):
7         # Learn the MLP regressor

```

```
8     mlp = MLPRegressor(hidden_layer_sizes=(10,10), activation='
relu', solver='adam', max_iter = iter, random_state=i)
9     #Predict output
10    y_pred += mlp.fit(variables_train,target_train).predict(
variables_test)
11    y_pred = y_pred/10
12    rmse_final.append(np.sqrt(np.mean((y_pred-target_test)**2)))
13
14 def const(x):
15     return first_rmse
16
17 # Plot the RMSE
18 plt.plot(iter_array, rmse_final, '-o', label='RMSE')
19 plt.hlines(first_rmse, xmin=min(iter_array), xmax=max(iter_array),
colors='r', linestyle='dashed')
20 plt.xlabel('Number of iterations')
21 plt.ylabel('RMSE')
22 plt.title('RMSE vs number of iterations')
23 plt.savefig('ex3_rmse.png')
24 plt.show()
```

4. Early stopping é uma técnica utilizada para combater o overfitting. O gráfico acima assemelha-se aos gráficos de overfitting mas não chega a subir ???????????