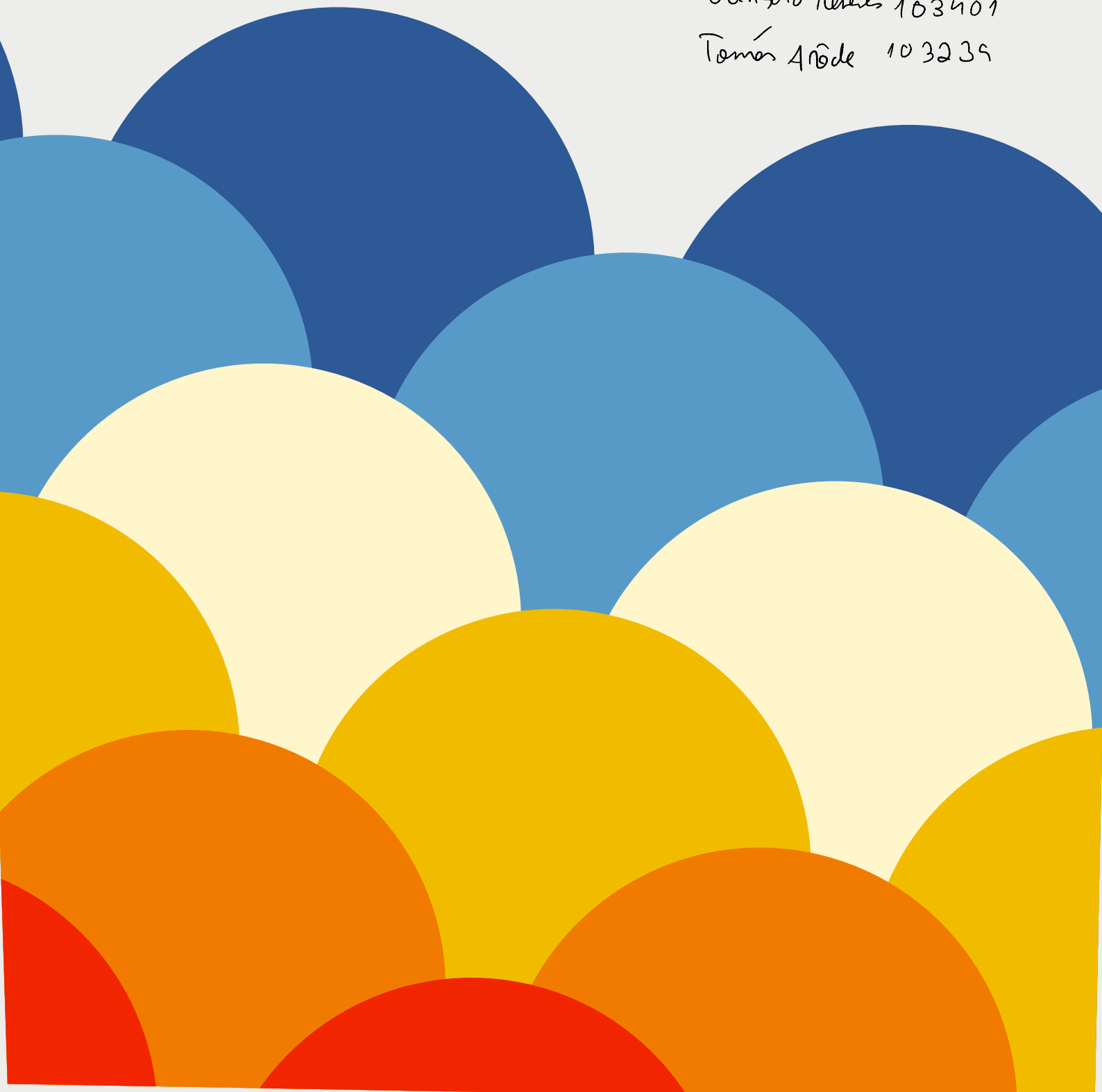
Homework 3

Gonçalo Neves 103401
Tomás Abade 103239



① a)

	y_1	y_2	t
x_1	0,7	-0,3	0,8
x_2	0,4	0,5	0,6
x_3	-0,2	0,8	0,3
x_4	-0,4	0,3	0,3

$$C = \left[\begin{pmatrix} 0 \\ 0 \end{pmatrix}; \begin{pmatrix} 1 \\ -1 \end{pmatrix}; \begin{pmatrix} -1 \\ 1 \end{pmatrix} \right]$$

C_1 C_2 C_3

0,1

$\eta = 1$

$$\phi_j(x) = \exp\left(- \frac{\| x - C_j \|^2}{2} \right)$$

$$X = \begin{bmatrix} 1 & \phi_1(x_1) & \phi_2(x_1) & \phi_3(x_1) \\ 1 & \phi_1(x_2) & \phi_2(x_2) & \phi_3(x_2) \\ 1 & \phi_1(x_3) & \phi_2(x_3) & \phi_3(x_3) \\ 1 & \phi_1(x_4) & \phi_2(x_4) & \phi_3(x_4) \end{bmatrix} = \begin{bmatrix} 1 & 0,74826 & 0,74826 & 0,10127 \\ 1 & 0,81465 & 0,27117 & 0,33121 \\ 1 & 0,71177 & 0,09633 & 0,71177 \\ 1 & 0,88250 & 0,16122 & 0,65377 \end{bmatrix}$$

$$W = (X^T \cdot X + \lambda \cdot I)^{-1} \cdot X^T \cdot t = \begin{bmatrix} 0,33914 \\ 0,19945 \\ 0,40096 \\ -0,29600 \end{bmatrix}$$

equação do hiperplano: $w_0 + w_1 \cdot y_1 + w_2 y_2 + w_3 \cdot y_3 = \hat{z}$

↳ $0,33914 + 0,19945 \cdot y_1 + 0,40096 \cdot y_2 - 0,29600 \cdot y_3 = \hat{z}$

b)

$$\hat{t} = X \cdot W = \begin{bmatrix} 0,75844 \\ 0,51232 \\ 0,30905 \\ 0,38629 \end{bmatrix}$$

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^{n} \left(t_i - \hat{t}_i \right)^2} = 0,06508$$

2)

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

$$b^{[1]} = [1 \ 1 \ 1]^T \qquad b^{[2]} = [1 \ 1]^T \qquad b^{[3]} = [1 \ 1 \ 1]^T$$

$f(x) = \tanh(0,5 \cdot x - 2)$ $\qquad \eta = 0,1$

$E(w) = \frac{1}{2} \| z - \hat{z} \|_2^2$

Nota: Todos os cálculos foram realizados com auxílio da biblioteca numpy.

→ $x_1 = [1 \ 1 \ 1 \ 1]^T$, $t_1 = B = [-1 \ 1 \ -1]^T$

→ $x_2 = [1 \ 0 \ 0 \ -1]^T$, $t_2 = A = [1 \ 0 \ 0 \ -1]^T$

Forward Propagation → $x^{[p]} = \phi^{[p]} \cdot (w^{[p]} x^{(p-1)} + b^{[p]})$

Para x_1:

$$z_1^{[1]} = w^{[1]} x_1^{[0]} + b^{[1]} = [5 \ 6 \ 5]^T$$

$$x_1^{[1]} = \phi^{[1]} \cdot (z_1^{[1]}) = f(z_1^{[1]}) = [0,46212 \ \ 0,76159 \ \ 0,46212]^T$$

Nota: Neste caso: $\phi^{(1)}(x) = \phi(x)^{[2]} = \phi(x)^{[3]} = f(x)$

Da mesma forma:

$$z_1^{[2]} = [4,97061 \ \ 2,68583]^T \qquad z_1^{[3]} = [0,87406 \ \ 1,77503 \ \ 0,87406]^T$$

$$x_1^{[2]} = [0,45048 \ \ -0,57642]^T \qquad x_1^{[3]} = [-0,91590 \ \ -0,80494 \ \ -0,91590]^T$$

De seguida calculamos os deltas para x_1:

δ da última camada: $\delta^{[P]} = (x_i^{[P]} - t_i) \circ \phi'^{[P]}(z^{[1]})$

δ das restantes camadas: $\delta^{[p]} = w^{[p+1]^T} \cdot \delta_i^{[p+1]} \circ \phi'^{[p]}(z_i^{[p]})$

$$\frac{d}{dx}\tanh(x) = 1 - \tanh^2(x) \Rightarrow \frac{d}{dx}\tanh(0{,}5x-2) = \frac{1}{2}\left(1 - \tanh^2(0{,}5\cdot x - 2)\right)$$

Posteriormente $\phi'^{[P]} = f'(x) = \frac{1}{2}[1 - \tanh^2(0{,}5x-2)]$

Assim, $\delta_1^{[3]} = (x_1^{[3]} - t_1) \circ f'(z_1^{[3]}) = [0{,}00678 \quad -0{,}31773 \quad 0{,}00678]^T$

$\delta^{[2]} = w^{[3]^T} \cdot \delta_1^{[3]} \circ f'(z_1^{[2]}) = [-0{,}37448 \quad -0{,}10156]^T$

Deste modo, $\delta^{[1]} = [-0{,}18719 \quad -0{,}33587 \quad -0{,}18719]^T$

Para x_2 executamos exatamente o mesmo algoritmo e obtenos os seguintes resultados:

Forward Propagation:

$z_2^{[1]} = [1 \quad 1 \quad 1]^T$

$x_2^{[1]} = [-0{,}90515 \quad -0{,}90515 \quad -0{,}90515]^T$

$z_2^{[2]} = [-4{,}43089 \quad -1{,}71544]^T$

$x_2^{[2]} = [-0{,}99956 \quad -0{,}99343]^T$

$z_2^{[3]} = [-0{,}99299 \quad -2{,}99212 \quad -0{,}99299]^T$

$x_2^{[3]} = [-0{,}98652 \quad -0{,}99816 \quad -0{,}98652]^T$

Cálculo de deltas:

$\delta_2^{[3]} = [-2{,}65961\cdot10^{-2} \quad 3{,}36962\cdot10^{-6} \quad 1{,}80463\cdot10^{-5}]^T$

$\delta_2^{[2]} = [-1{,}15093\cdot10^{-5} \quad -1{,}72899\cdot10^{-4}]^T$

$\delta_2^{[1]} = [-1{,}66619\cdot10^{-5} \quad -1{,}97816\cdot10^{-5} \quad -1{,}66619\cdot10^{-5}]$

Backpropagation: $w^{[p]} = w^{[p]} - \eta \left(\frac{\partial E^{(1)}}{\partial w^{(p)}} + \frac{\partial E^{(2)}}{\partial w^{(p)}} \right) =$

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

$$w^{[3]} = \begin{bmatrix} 0,99704 & 0,99775 \\ 3,01431 & 0,98169 \\ 0,99971 & 1,00041 \end{bmatrix}$$

$$w^{[2]} = \begin{bmatrix} 1,01430 & 1,02852 & 1,01730 \\ 1,00468 & 1,00972 & 1,00468 \end{bmatrix}$$

$$w^{[1]} = \begin{bmatrix} 1,01872 & 1,01872 & 1,01872 & 1,01872 \\ 1,03359 & 1,03359 & 2,03359 & 1,03359 \\ 1,01872 & 1,01872 & 1,01872 & 1,01872 \end{bmatrix}$$

$$b^{[p]} = b^{[p]} - \eta \cdot \left[\frac{\partial E^{(1)}}{\partial b^{(p)}} + \frac{\partial E^{(2)}}{\partial b^{(p)}} \right] = b^{[p]} - \eta \cdot \left[\delta_1^{[p]} + \delta_2^{[p]} \right]$$

$$b^{[3]} = [1,00198 \quad 1,03177 \quad 0,99930]^T$$

$$b^{[2]} = [1,03745 \quad 1,01017]^T$$

$$b^{[1]} = [1,01872 \quad 1,03359 \quad 1,01872]^T$$

G11_notebook

October 20, 2023

1 Homework 2

Gonçalo Meneses, 103401 e Tomás Arêde, 103239

Antes de começar, vamos remover os avisos que surgem ao longo deste notebook.

[ ]:
```
import warnings
warnings.filterwarnings('ignore')
```

Começamos, então, por carregar o nosso ficheiro *csv* e criar o data frame que irá armazenar os nossos dados.

[ ]:
```
import pandas as pd
from sklearn.model_selection import train_test_split

data = pd.read_csv("winequality-red.csv",delimiter=';')

df = pd.DataFrame(data)

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

df.head()
```

[ ]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides
0	7.4	0.70	0.00	1.9	0.076
1	7.8	0.88	0.00	2.6	0.098
2	7.8	0.76	0.04	2.3	0.092
3	11.2	0.28	0.56	1.9	0.075
4	7.4	0.70	0.00	1.9	0.076

	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates
0	11.0	34.0	0.9978	3.51	0.56
1	25.0	67.0	0.9968	3.20	0.68
2	15.0	54.0	0.9970	3.26	0.65
3	17.0	60.0	0.9980	3.16	0.58
4	11.0	34.0	0.9978	3.51	0.56

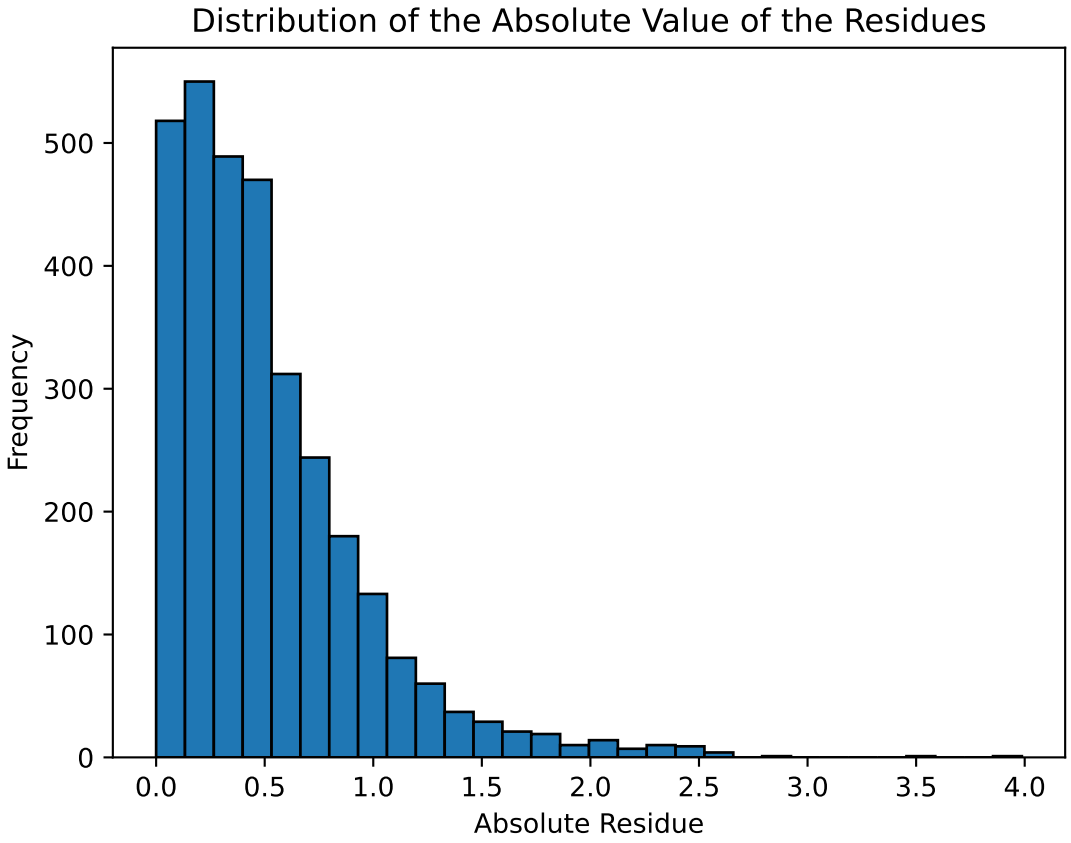
	alcohol	quality

```
0      9.4        5
1      9.8        5
2      9.8        5
3      9.8        6
4      9.4        5
```

1.0.1 Exercício 1

[ ]:
```
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from IPython.display import set_matplotlib_formats
set_matplotlib_formats('svg', 'pdf')

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,␣
 ↪random_state=0)

residue_list = []

mae_scores = []
mae_rounded_scores = []

rmse_early_scores = []

for state in range(1, 11):

    mlp_regressor = MLPRegressor(hidden_layer_sizes=(10, 10),␣
 ↪activation='relu', early_stopping=True, validation_fraction=0.2,␣
 ↪random_state=state)
    mlp_regressor.fit(X_train, y_train)

    y_pred = mlp_regressor.predict(X_test)

    residues = np.abs(y_pred - y_test)
    residue_list.extend(residues)

    mae = mean_absolute_error(y_true=y_test, y_pred=y_pred)
    mae_scores.append(mae)

    y_pred_rounded = np.round(y_pred)
    y_pred_rounded_bounded = np.clip(y_pred_rounded, 1, 10)

    mae_rounded = mean_absolute_error(y_true=y_test,␣
 ↪y_pred=y_pred_rounded_bounded)
```

```
    mae_rounded_scores.append(mae_rounded)

    rmse_early = np.sqrt(mean_squared_error(y_true=y_test, y_pred=y_pred))
    rmse_early_scores.append(rmse_early)

plt.hist(residue_list, bins=30, edgecolor='k')
plt.xlabel('Absolute Residue')
plt.ylabel('Frequency')
plt.title('Distribution of the Absolute Value of the Residues')
plt.show()
```



1.0.2 Exercício 2

[ ]:
```
avg_mae_scores = np.average(mae_scores)
avg_rounded_mae_scores = np.average(mae_rounded_scores)

print(f'Average of the Mean Absolute Errors: {avg_mae_scores:.5f}')
print(f'Average of the Mean Absolute Erros with rounding and bounding:␣
  ↪{avg_rounded_mae_scores:.5f}')
```

```
if avg_rounded_mae_scores < avg_mae_scores:
    print('Rounding and bounding estimates is beneficial to the performance of␣
 ↪our MLP classifier.')
else:
    print("Our MLP classifier doesn't benefit from rounding and bounding␣
 ↪estimates.")
```

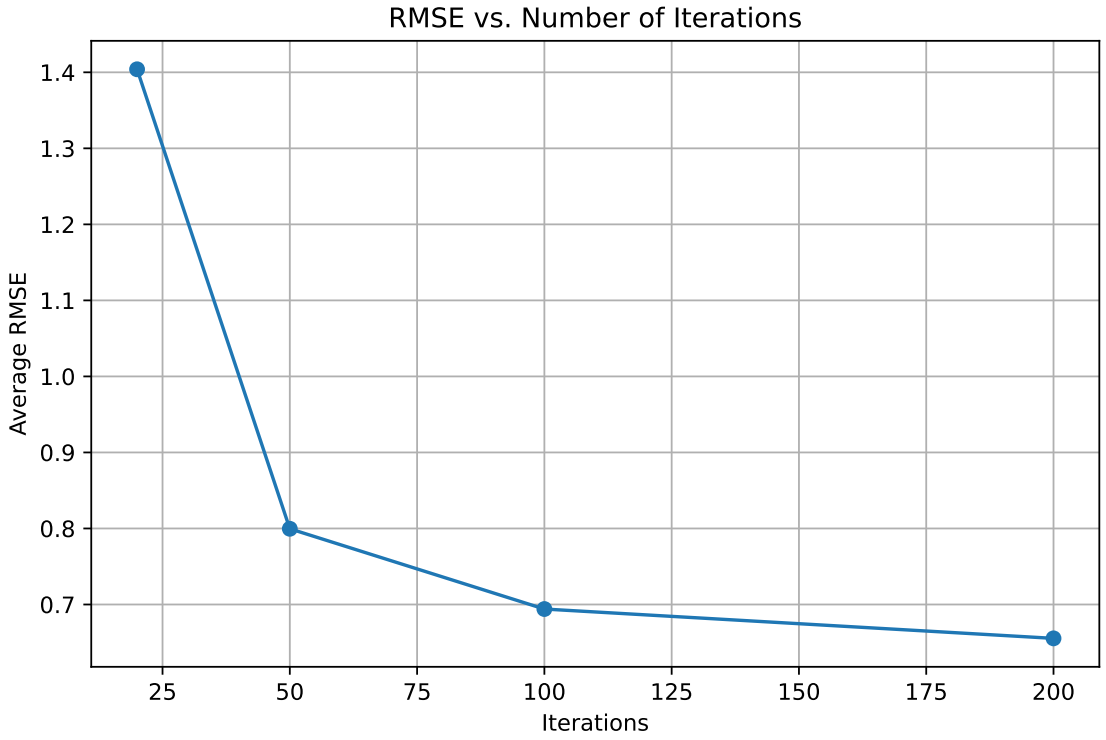
```
Average of the Mean Absolute Errors: 0.50972
Average of the Mean Absolute Erros with rounding and bounding: 0.43875
Rounding and bounding estimates is beneficial to the performance of our MLP
classifier.
```

1.0.3 Exercício 3

[ ]:
```
num_iterations = [20, 50, 100, 200]

rmse_scores = []

for random_state in range(1, 11):
    for iter in num_iterations:

        mlp_regressor = MLPRegressor(hidden_layer_sizes=(10, 10),␣
 ↪activation='relu', max_iter=iter, random_state=random_state)

        mlp_regressor.fit(X_train, y_train)

        y_pred = mlp_regressor.predict(X_test)

        rmse = np.sqrt(mean_squared_error(y_true=y_test, y_pred=y_pred))
        rmse_scores.append((iter, rmse))

avg_rmse_early_score = np.average(rmse_early_scores)

print(f'The average RMSE score for Early Stopping is: {avg_rmse_early_score:.
 ↪5f}')

average_rmse_scores = {}

for iter, rmse in rmse_scores:
    average_rmse_scores.setdefault(iter, []).append(rmse)

iters, avg_rmse_values = zip(*[(iter, np.mean(rmse_list)) for iter, rmse_list␣
 ↪in average_rmse_scores.items()])
[print(f'The average RMSE for {iter} iterations is: {avg_rmse:.5f}') for iter,␣
 ↪avg_rmse in zip(iters, avg_rmse_values)]

plt.figure(figsize=(8, 5))
```

```
plt.plot(iters, avg_rmse_values, marker='o')
plt.title('RMSE vs. Number of Iterations')
plt.xlabel('Iterations')
plt.ylabel('Average RMSE')
plt.grid(True)
plt.show()
```

```
The average RMSE score for Early Stopping is: 0.67065
The average RMSE for 20 iterations is: 1.40398
The average RMSE for 50 iterations is: 0.79961
The average RMSE for 100 iterations is: 0.69404
The average RMSE for 200 iterations is: 0.65545
```



1.0.4 Exercício 4

Apesar de, como se pode ver acima, o valor do *RMSE* ter diminuído com o número de iterações, é importante destacar alguns pontos.

Primeiro, o valor do erro associado a 20 iterações é de longe o mais alto, quando comparado aos das restantes iterações. Isso significa que, neste caso, o nosso modelo requer um grande número de iterações para treinar o modelo, ou seja, se utilizarmos *Early Stopping*, este poderá causar um término precoce do nosso treino, o que levará a um *underfitting* e, consequentemente, a uma performance subóptima.

Segundo, à medida que o número de iterações aumenta (neste caso, 20, 50, 100, 200), a performance

do nosso modelo também, e, em troca, o valor do *RMSE* diminui, porém, em incrementos cada vez mais pequenos. Assim, para um número suficientemente grande de iterações poderá haver um *overfitting* para os dados de treino, pelo que a utilização de *Early Stopping*, face ao número máximo de iterações, impediria que tal acontecesse. Isto é possível, uma vez que este método de regularização para o treino quando a performance do modelo para um conjunto de validação diminui e mantém a sua capacidade de generalização.

Finalmente, podemos concluir que um bom balanço entre *Early Stopping* e um número adequado de iterações é essencial para uma performance óptima do modelo, impedindo tanto um *underfit* como um *overfit*.