Aprendizagem 2024

HomeWork 3 - Grupo 02

Joana Vaz (106078) e David Quintino (107095)

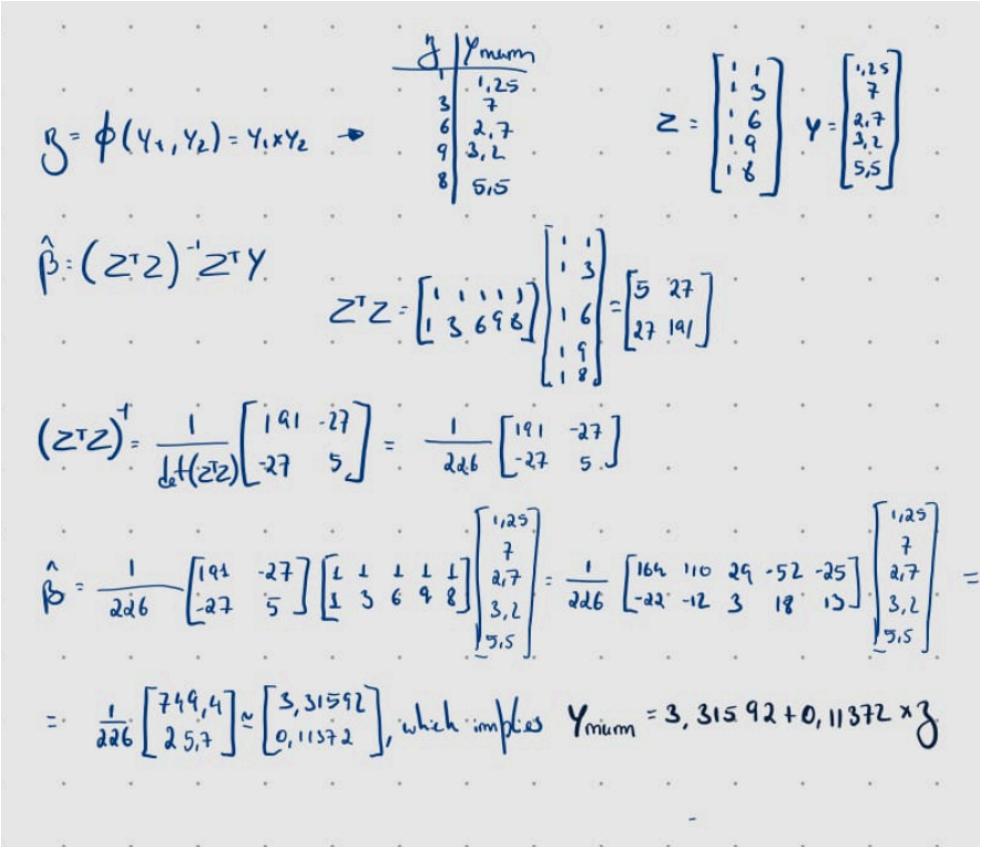
I. Pen-and-paper [12v]

For questions in this group, show your numerical results with 5 decimals or scientific notation.
Hint: we highly recommend the use of `numpy` (e.g., `linalg.pinv` for inverse) or other programmatic facilities to support the calculus involved in both questions (1) and (2).

Below is a training dataset D composed by two input variables and two output variables, one of which is numerical (y_{num}) and the other categorical (y_{class}). Consider a polynomial basis function $\phi(y_1, y_2) = y_1 \times y_2$ that transforms the original space into a new one-dimensional space.

D	y_1	y_2	y_{num}	y_{class}
x_1	1	1	1.25	B
x_2	1	3	7.0	A
x_3	3	2	2.7	C
x_4	3	3	3.2	A
x_5	2	4	5.5	B

1. [2v] Learn a regression model on the transformed feature space using the OLS closed form solution to predict the continuous output y_{num}



2. [2.5v] Repeat the previous exercise, but this time learn a Ridge regression with penalty factor $\lambda = 1$. Compare the learnt coefficients with the ones from the previous exercise and discuss how regularization affects them.

$\hat{\beta}_{Ridge} = (Z^T Z + \lambda I)^{-1} Z^T Y =$ given our matrixes and $\lambda = 1$

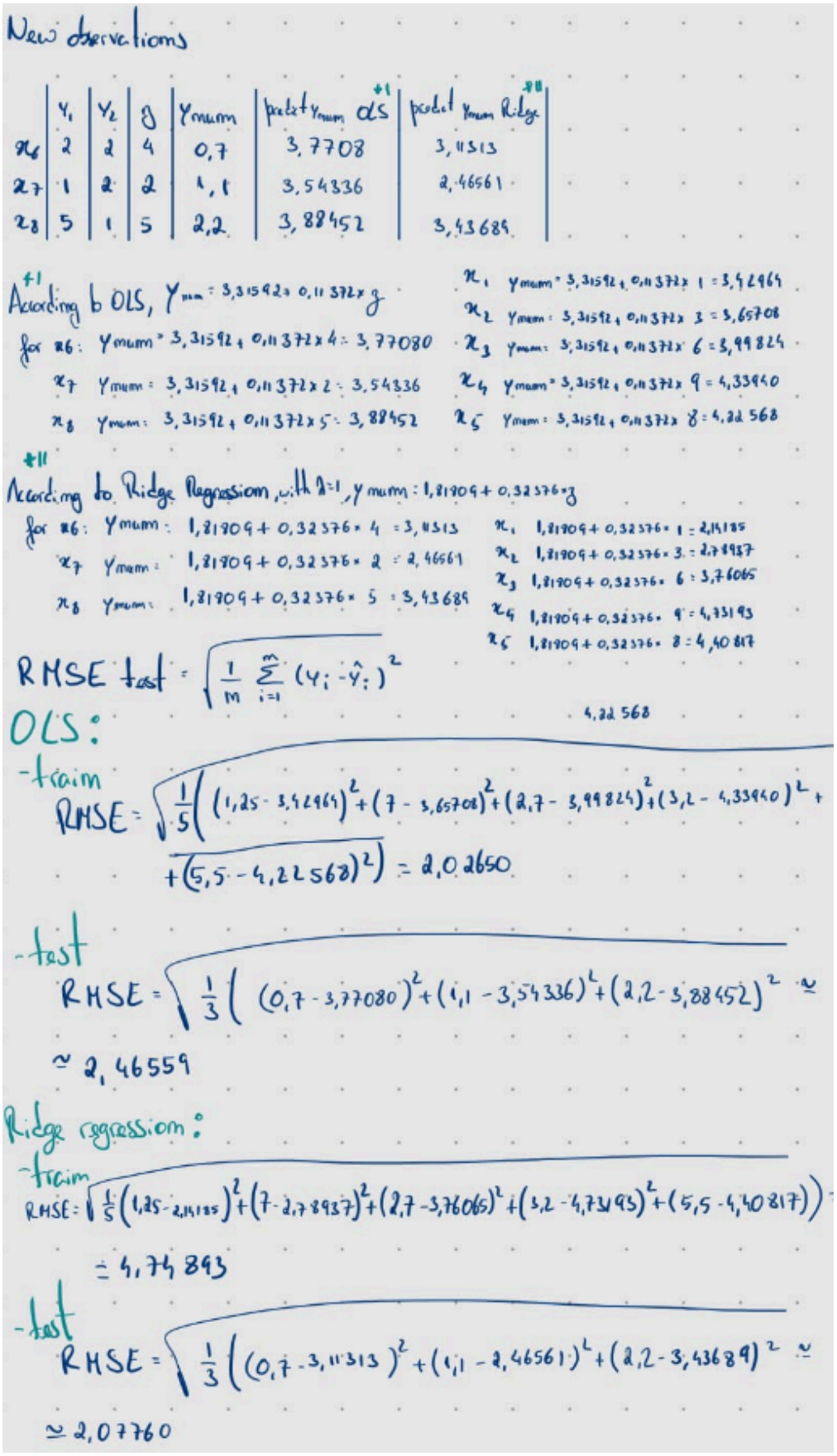
$= \begin{bmatrix} 6 & 27 \\ 27 & 192 \end{bmatrix}^{-1} Z^T Y = \frac{1}{423} \begin{bmatrix} 192 & -27 \\ -27 & 6 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 3 & 6 & 9 & 8 \end{bmatrix} \begin{bmatrix} 1,25 \\ 7 \\ 2,7 \\ 3,2 \\ 5,5 \end{bmatrix} = \frac{1}{423} \begin{bmatrix} 769,05 \\ 136,95 \end{bmatrix} =$

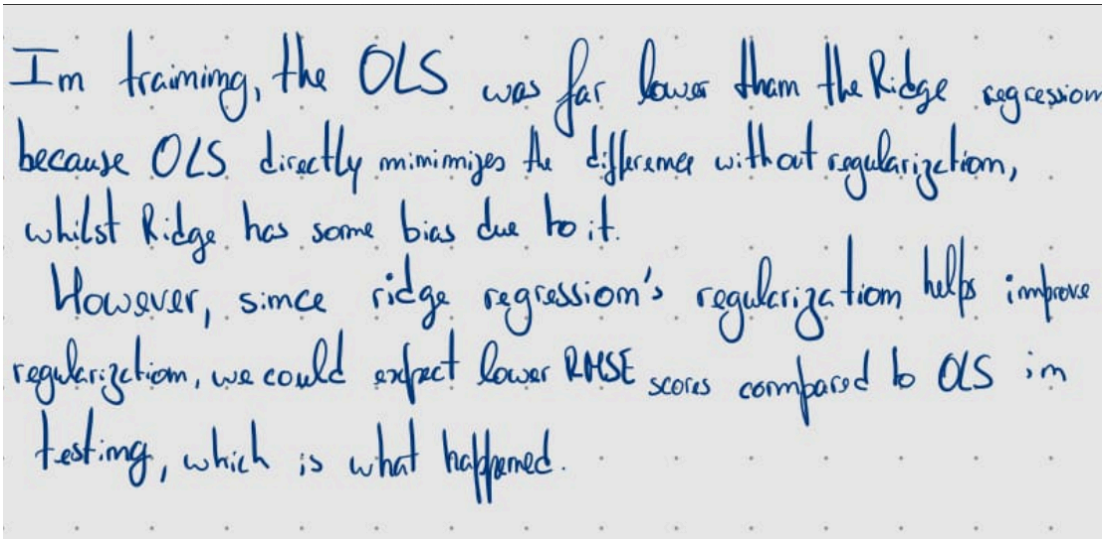
$= \begin{bmatrix} 1,81809 \\ 0,32376 \end{bmatrix}$, which implies $Y_{num} = 1,81809 + 0,32376 \times 3$

The ridge regression shows lower coefficients than OLS, because the former has regularization, which penalizes large coefficients and stabilizes the solution, as opposed to the latter.

In conclusion, regularization reduces the coefficients, often leading to a more precise expression.

3. [2.5v] Given three new test observations and their corresponding y_{num} output $x_6 = (2,\ 2,\ 0.7)$, $x_7 = (1,\ 2,\ 1.1)$ and $x_8 = (5,\ 1,\ 2.2)$, compare the train and test RMSE of the two models obtained in 1) and 2). Explain if the results go according to what is expected.





4. [5v] Consider an MLP to predict the output y_{class} characterized by the weights

$$W^{[1]} = \begin{bmatrix} 0.1 & 0.1 \\ 0.1 & 0.2 \\ 0.2 & 0.1 \end{bmatrix} \quad b^{[1]} = \begin{bmatrix} 0.1 \\ 0 \\ 0.1 \end{bmatrix} \quad W^{[2]} = \begin{bmatrix} 1 & 2 & 2 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad b^{[2]} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

the output activation function $softmax(Z_c^{[out]}) = \frac{e^{Z_c^{[out]}}}{\sum_{l=1}^{|C|} e^{Z_l^{[out]}}}$, no activations on the hidden layer(s) and the cross-entropy loss: $-\sum_{i=1}^{N}\sum_{l=1}^{|C|} t_l^{(i)} log(Z_l^{[out](i)})$. Consider also that the output layer of the MLP gives the predictions for the classes A, B and C in this order. Perform one stochastic gradient descent update to all the weights and biases with learning rate $\eta = 0.1$ using the training observation x_1.

$x_1 \rightarrow [1,1] \rightarrow X_0$

Forward pass

First layer

$$z^1 = W^1 \times X_0^T + b^1 = \begin{bmatrix} 0,1 & 0,1 \\ 0,1 & 0,2 \\ 0,2 & 0,1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 0,1 \\ 0 \\ 0,1 \end{bmatrix} = \begin{bmatrix} 0,3 \\ 0,3 \\ 0,4 \end{bmatrix}$$

Since there is no activations on the hidden layers $X_1 = z_1$

Second layer

$$z^2 = W^2 \times X_1 + b^2 = \begin{bmatrix} 1 & 2 & 2 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 0,3 \\ 0,3 \\ 0,4 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2,7 \\ 2,3 \\ 2 \end{bmatrix}$$

Output activation $\text{softmax}(z_c^{[out]}) = \dfrac{e^{z_c^{[out]}}}{\sum_{l=1}^{|C|} e^{z_l^{[out]}}}$

$$X_2 = \begin{bmatrix} \frac{e^{2,7}}{e^{2,7} + e^{2,3} + e^{2}} \\ \frac{e^{2,3}}{e^{2,7} + e^{2,3} + e^{2}} \\ \frac{e^{2}}{e^{2,7} + e^{2,3} + e^{2}} \end{bmatrix} = \begin{bmatrix} 0,46149 \\ 0,30934 \\ 0,22917 \end{bmatrix}$$

Cross entropy loss $= -\sum_{i=1}^{N} \sum_{l=1}^{|C|} t_l^{(i)} \log(X_l^{[out](i)})$

$N = 1$ because there is only one observation (x_1)

$t^T = [0 \ 1 \ 0]$ because x_1's class is B, so the B is set to one and others to zero.

$$-\sum_{i=1}^{N} \sum_{l=1}^{|C|} t_l^{(i)} \log(X_l^{[out](i)}) = -1 \times \log(0,30934) = 1,17331$$

Back propagation: $W^{[i]} \leftarrow W^{[i]} - \eta \frac{\partial E}{\partial W^{[i]}}$ $\quad b^{[i]} \leftarrow b^{[i]} - \eta \frac{\partial E}{\partial b^{[i]}}$

$$\frac{\partial E}{\partial W^{[2]}} = \frac{\partial E}{\partial X^{[2]}} \circ \frac{\partial X^{[2]}}{\partial Z^{[2]}} \cdot \frac{\partial Z^{[2]}}{\partial W^{[2]}} = \underbrace{\frac{\partial E}{\partial Z^{[2]}}}_{\delta^{[2]}} \cdot \left(\frac{\partial Z^{[2]}}{\partial W^{[2]}} \right)^T$$

$$\delta^{[2]} = \frac{\partial E}{\partial Z^{[2]}} = \frac{\partial \left(-\sum_{l=1}^{|C|} t_l \times \log(X_l^{[2]}) \right)}{\partial Z^{[2]}} = X^{[2]} - t$$

$$= \begin{bmatrix} 0,46149 \\ 0,30934 \\ 0,22917 \end{bmatrix} - \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0,46149 \\ -0,69066 \\ 0,22917 \end{bmatrix}$$

$$W_2 = W_2 - 0,1\left(\delta^{[2]}(X^{[1]})^T\right) = \begin{bmatrix} 1 & 2 & 2 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix} - 0,1\left(\begin{bmatrix} 0,46149 \\ -0,69066 \\ 0,22917 \end{bmatrix} \times [0,5 \;\; 0,5 \;\; 0,4]\right)$$

$$= \begin{bmatrix} 0,98616 & 1,98616 & 1,98154 \\ 1,02072 & 2,02072 & 1,02763 \\ 0,99312 & 0,99312 & 0,99083 \end{bmatrix}$$

$$b^{[2]} = b^{[2]} - 0,1 \times \frac{\partial E}{\partial b^2} = b^{[2]} - 0,1 \times \delta^{[2]} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - 0,1 \times \begin{bmatrix} 0,46149 \\ -0,69066 \\ 0,22917 \end{bmatrix} =$$

$$= \begin{bmatrix} 0,95385 \\ 1,06907 \\ 0,97708 \end{bmatrix}$$

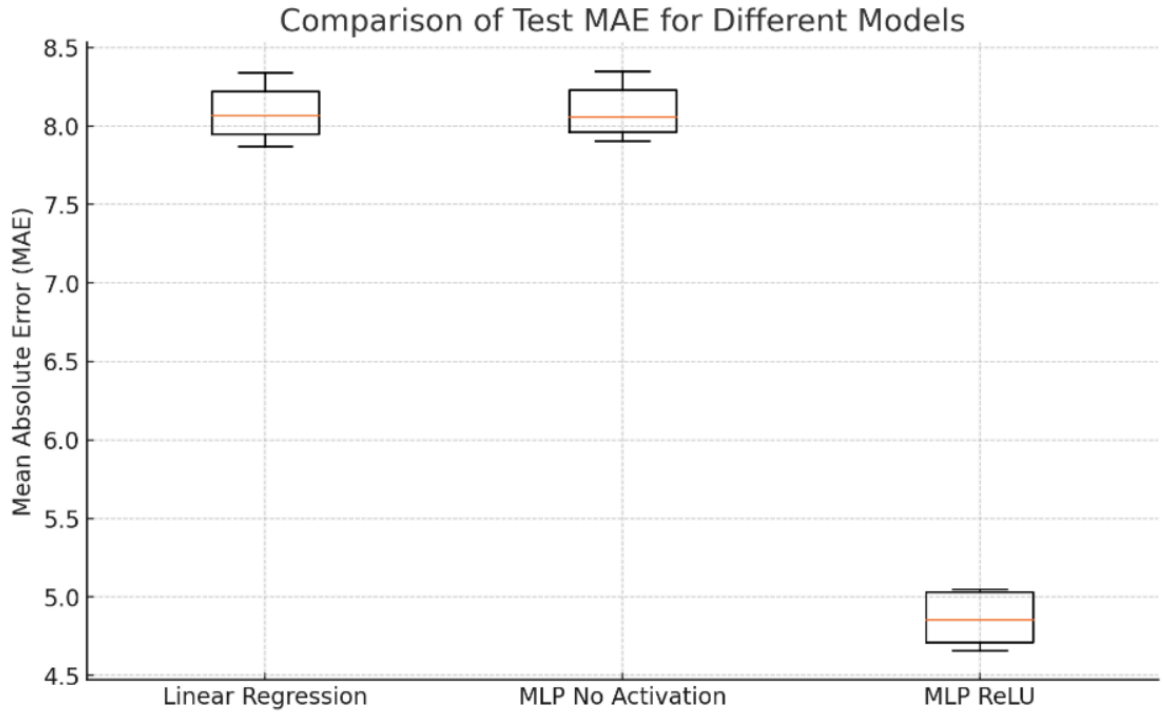
II. Programming and critical analysis [8v]

Consider the `parkinsons.csv` dataset (available at the course's webpage), where the goal is to predict a patient's score on the Unified Parkinson's Disease Rating Scale based on various biomedical measurements.

To answer question 5), average the performance of the models over 10 separate runs. In each run, use a different 80-20 train-test split by setting a `random_state=i`, with `i=1..10`.

5. [2v] Train a Linear Regression model, an MLP Regressor with 2 hidden layers of 10 neurons each and no activation functions, and another MLP Regressor with 2 hidden layers of 10 neurons each using ReLU activation functions. (Use `random_state=0` on the MLPs, regardless of the run). Plot a boxplot of the test MAE of each model.

The Mean Absolute Error (MAE) results for the three models (across 10 different train-test splits) are as follows: Linear Regression MAE [8.23, 7.98, 8.06, 8.34, 8.23, 7.91, 7.94, 8.20, 8.08, 7.87], MLP Regressor with no activation function (identity) MAE [8.25, 7.99, 8.06, 8.35, 8.25, 7.91, 7.95, 8.18, 8.07, 7.91] and MLP Regressor with ReLU activation MAE [5.02, 4.66, 5.05, 5.04, 5.05, 4.73, 4.68, 4.91, 4.71, 4.81].



The boxplot shows a comparison of the Mean Absolute Error (MAE) across the three models. Both Linear Regression and the MLP with no activation function (identity) demonstrate very similar performance, with an MAE of around 8.0. In contrast, the MLP with ReLU activation significantly outperforms the other two models, achieving a much lower MAE, around 5.0. This suggests that the MLP with ReLU activation provides better results, likely because the non-linear ReLU activation allows the model to capture more complex relationships in the data.

6. [3v] Compare a Linear Regression with a MLP with no activations, and explain the impact and the importance of using activation functions in a MLP. Support your reasoning with the results from the boxplots.

Linear Regression is a simple linear model where the relationship between inputs and the output is assumed to be linear. Each input feature is assigned a weight, and the output is the weighted sum of the inputs. An MLP without activation functions is essentially equivalent to multiple layers of linear transformations, as the identity activation function keeps each layer as a linear transformation. Since a composition of linear transformations remains linear, an MLP without activation functions is functionally similar to a linear model like Linear Regression.

In terms of performance, both the Linear Regression model and the MLP without activation functions perform almost identically, with Mean Absolute Error (MAE) values hovering around 8.0, as shown in the boxplots. This demonstrates that without activation functions, the MLP does not offer any advantage over Linear Regression, as both are constrained to modeling linear relationships.

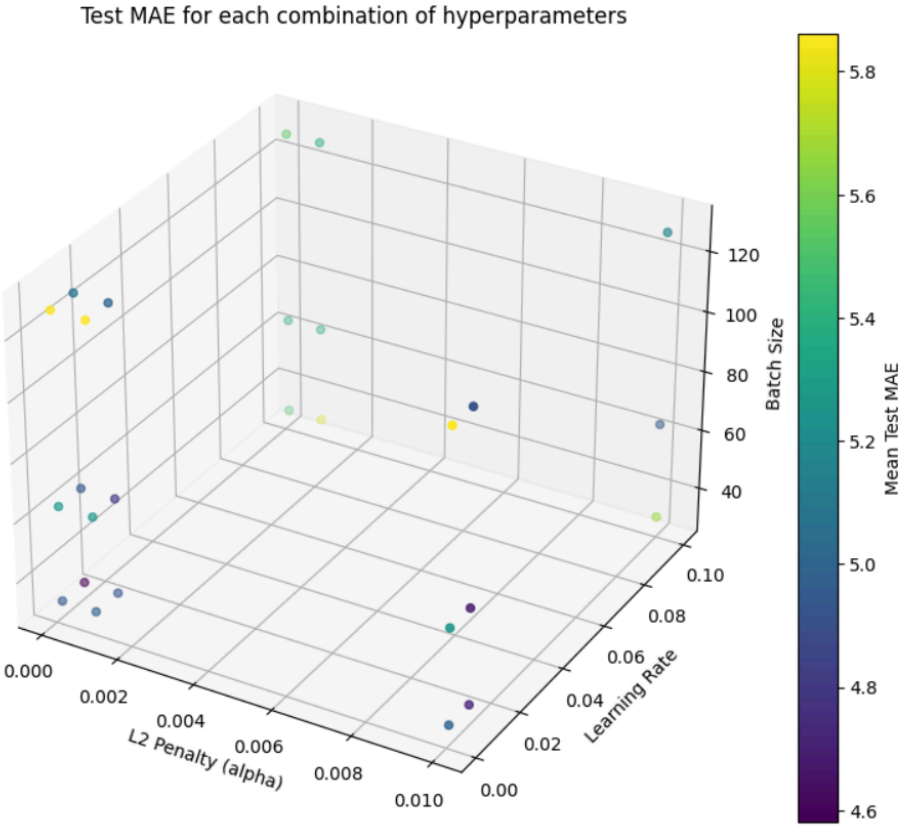
Activation functions play a critical role in MLPs by introducing non-linearity, which is necessary for the network to capture complex patterns in data. Without activation functions, an MLP can only model linear relationships, as seen in the MLP without activations. In contrast, the MLP with ReLU activation significantly outperformed both Linear Regression and the MLP without activations in the experiment, achieving an MAE of around 5.0 compared to the others' 8.0. This highlights the importance of non-linear activation functions, with ReLU enabling the MLP to capture more complex, non-linear relationships, leading to better performance in predicting Parkinson's Disease scores.

In conclusion, the use of activation functions in MLPs is essential for capturing non-linear relationships in data, resulting in improved performance. Without activation functions, the MLP is reduced to a series of linear transformations, offering no benefit over a simple linear model like Linear Regression. This is evident from the boxplot comparison, where the MLP with ReLU activation outperforms both Linear Regression and the MLP without activations.

7. [4v] Using a 80-20 train-test split with `random_state=0`, use a Grid Search to tune the hyperparameters of an MLP regressor with two hidden layers (size `10` each). The parameters to search over are: (i) L2 penalty, with the values `{0.0001, 0.001, 0.01}`; (ii) learning rate, with the values `{0.001, 0.01, 0.1}`; and (iii) batch size, with the values `{32, 64, 128}`. Plot the test MAE for each combination of hyperparameters, report the best combination, and discuss the trade-offs between the combinations.

```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_absolute_error
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

data = pd.read_csv('parkinsons.csv')


X = data.drop(columns=['target'])
y = data['target']


mlp = MLPRegressor(hidden_layer_sizes=(10, 10), random_state=0,)


param_grid = {
    'alpha': [0.0001, 0.001, 0.01],  # L2 penalty
    'learning_rate_init': [0.001, 0.01, 0.1],  # Learning rate
    'batch_size': [32, 64, 128]  # Batch size
}

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=0)

grid_search = GridSearchCV(mlp, param_grid,
scoring='neg_mean_absolute_error', cv=5, n_jobs=-1, verbose=1)
grid_search.fit(X_train, y_train)

best_params = grid_search.best_params_
results = pd.DataFrame(grid_search.cv_results_)

best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)
```

```
test_mae = mean_absolute_error(y_test, y_pred)

print(f"Best hyperparameters: {best_params}")
print(f"Test MAE: {test_mae}")

mean_test_scores = -results['mean_test_score']
alphas = results['param_alpha'].astype(float)
learning_rates = results['param_learning_rate_init'].astype(float)
batch_sizes = results['param_batch_size'].astype(float)

fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

sc = ax.scatter(alphas, learning_rates, batch_sizes,
c=mean_test_scores, cmap='viridis')
ax.set_xlabel('L2 Penalty (alpha)')
ax.set_ylabel('Learning Rate')
ax.set_zlabel('Batch Size')
ax.set_title('Test MAE for each combination of hyperparameters')

plt.colorbar(sc, ax=ax, label='Mean Test MAE')

plt.show()
```



The code you provided performs a Grid Search to optimize the hyperparameters of the MLPRegressor (L2 penalty, learning rate, and batch size) and generates a 3D plot showing the Mean Absolute Error (MAE) for each combination of hyperparameters.

The L2 penalty, controlled by the alpha parameter, helps prevent overfitting by penalizing large weights in the model. Lower values of alpha allow the model to fit the data better, while higher values impose stronger regularization. In the plot, you can observe how different values of alpha impact the MAE. The ideal alpha should balance model complexity and its ability to generalize. If the MAE is lower for smaller alpha values, this suggests that the model benefits from less regularization.

The learning_rate_init parameter defines how fast the model adjusts its weights during training. Very high learning rates can cause the model to overshoot local minima, while very low rates may result in slow convergence. If the MAE is lower with medium learning rates, it indicates good convergence. A common trade-off is that very high learning rates may lead to worse performance, while very low rates are more stable but slower to converge.

The batch size affects how often the model's weights are updated during training. Smaller batch sizes lead to more frequent, noisier updates, while larger batch sizes result in smoother, less frequent updates. The 3D plot should show how batch size influences the combination of alpha and learning rate. If the MAE is lower with intermediate batch sizes, it suggests that these batch sizes strike a balance between frequent updates and reduced variability.

At the end of the Grid Search, the best hyperparameter combination is selected based on the lowest MAE, meaning that the optimized combination maximizes the model's accuracy in predicting the score on the Unified Parkinson's Disease Rating Scale.

There are several trade-offs to consider. Very high L2 penalties can lead to underfitting, while very low penalties might allow the model to memorize the training data (overfitting). The goal is to find a balance. A high learning rate may reduce training time but can prevent the model from finding an optimal point, while low learning rates are safer but slower. Smaller batch sizes introduce more noise into the weight updates but allow for quicker convergence, whereas larger batch sizes provide stability but may slow down the process.

If the generated plot shows points with lower MAE values for intermediate levels of alpha, learning_rate_init, and batch_size, it indicates that a balance between moderate regularization, intermediate learning rates, and batch sizes leads to the best performance.