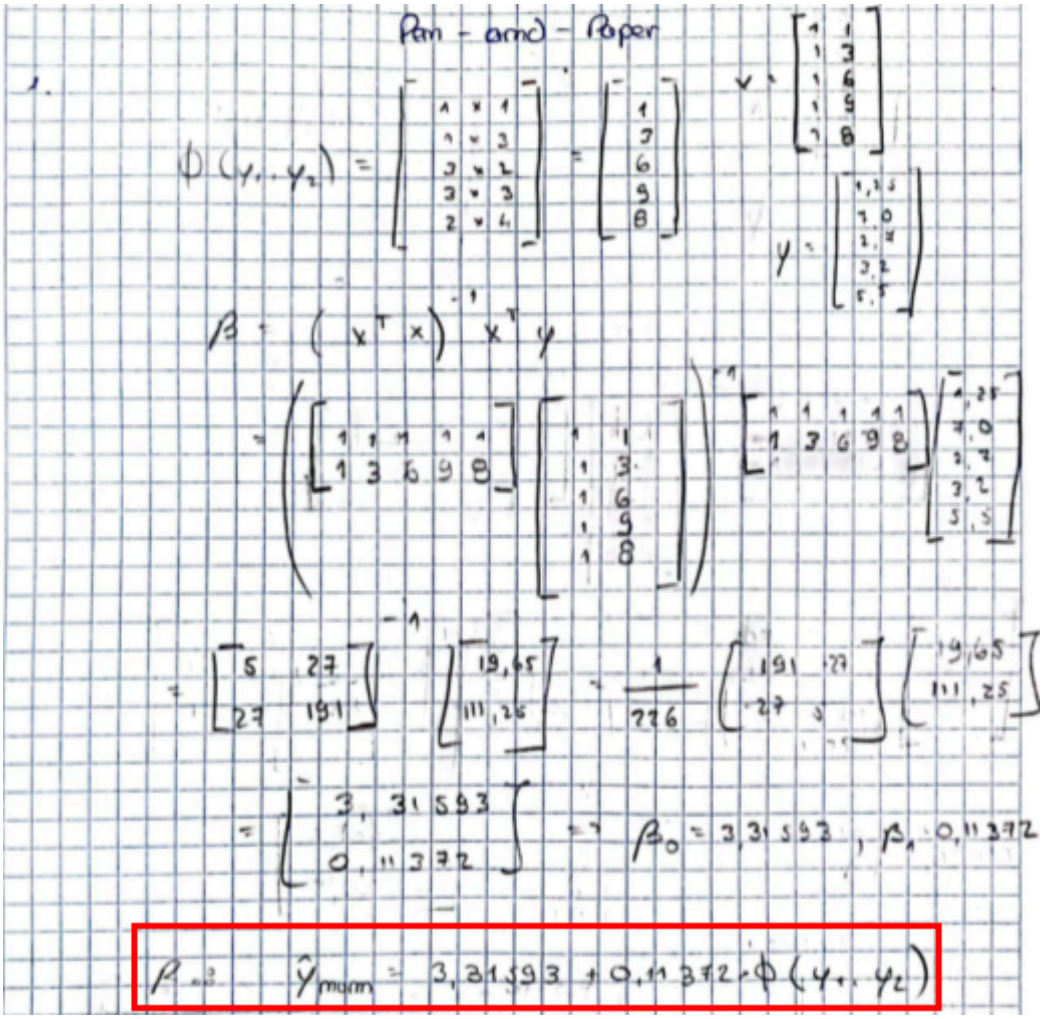
Aprendizagem 2024/25

Homework III – Group 066

(ist106794, ist1107301)

I. Pen and Paper

1)

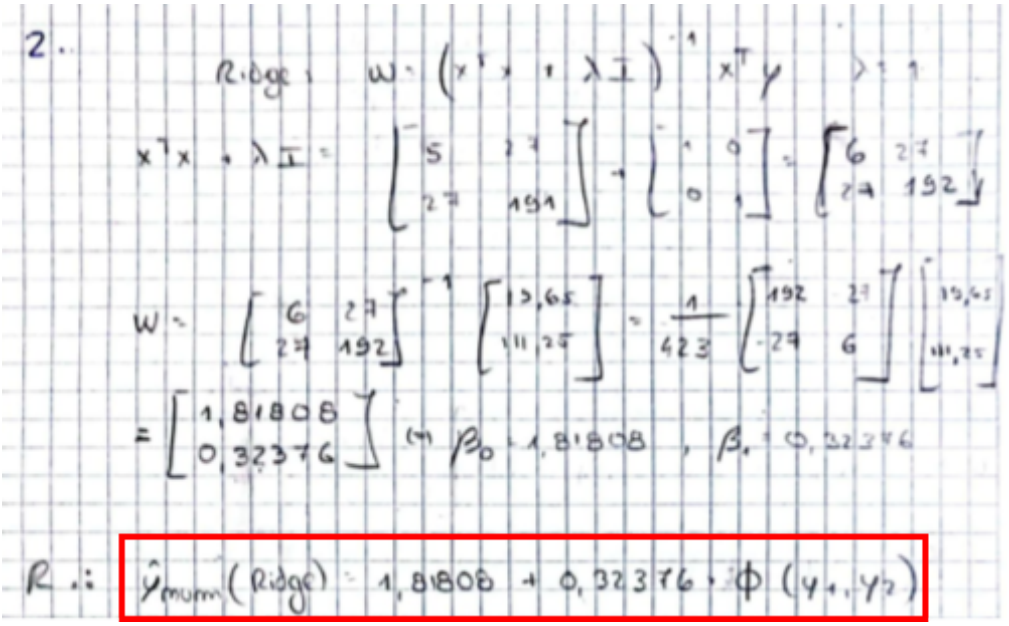


Resposta: $\hat{y}_{num} = 3.31593 + 0.11372 * \phi(y_1, y_2)$

Homework III – Group 066

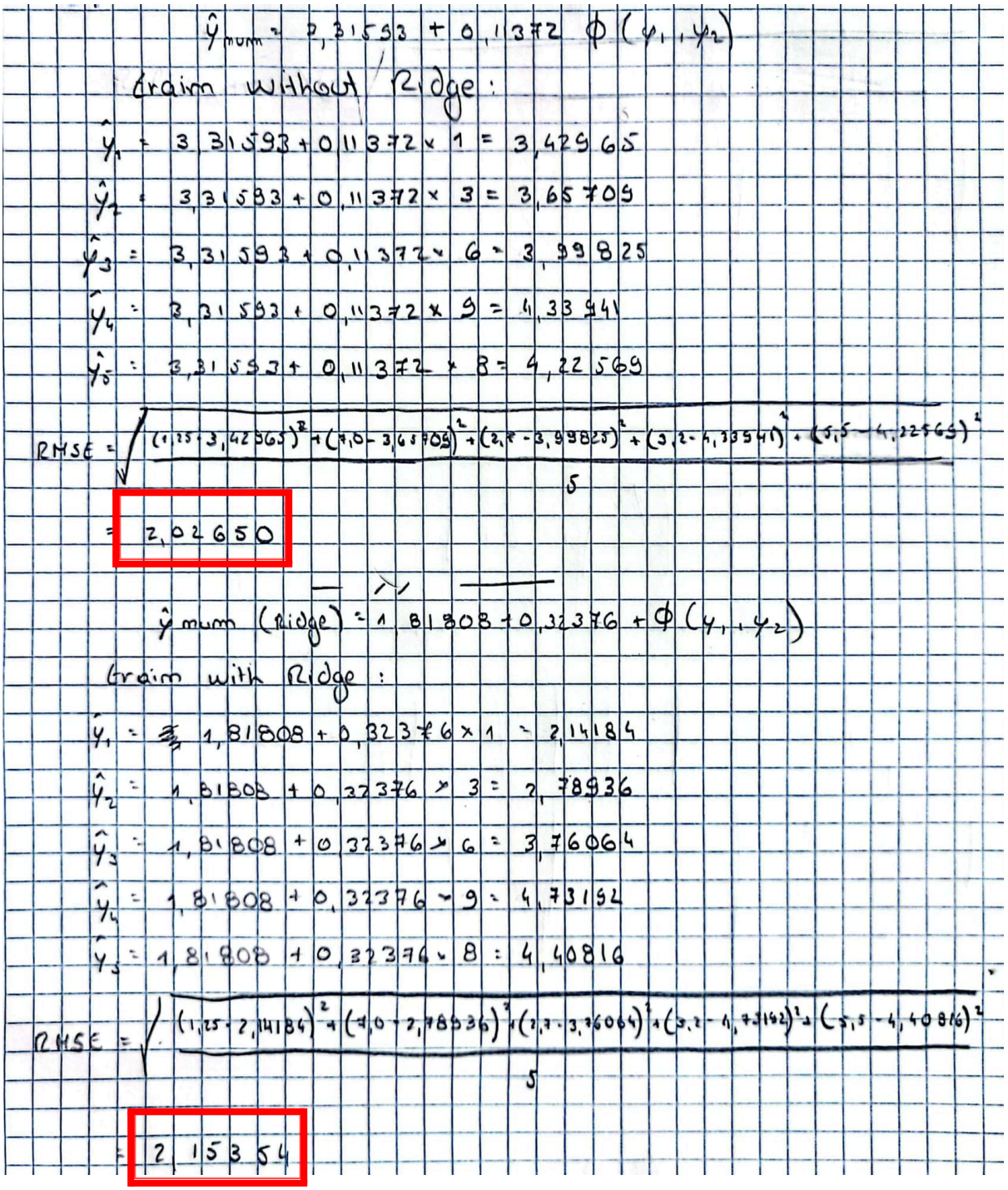
(ist106794, ist1107301)

2)



By using Ridge's formula, the value for the intercept decreased and the value for the slope increased. This is the result of the decrease of large coefficients, a characteristic of this regularization method . Since the data set is pretty small and therefore noisy, applying Ridge's regularization helps reduce variance in the data, preventing overfitting at the cost of introducing bias. It can have a stronger effect by increasing the λ parameter.

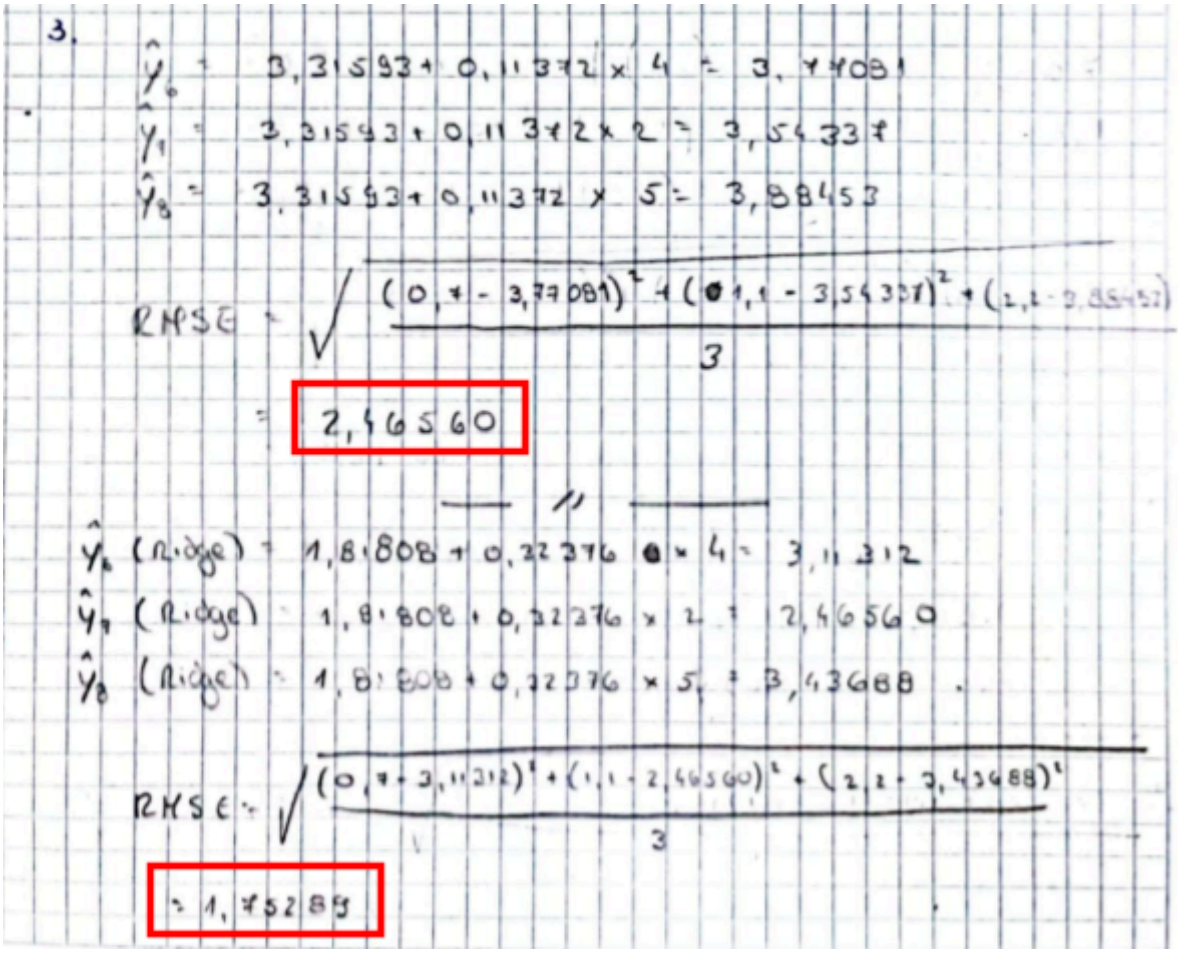
3)


ŷ num = 2,31593 + 0,11372 Φ(y1, y2)
train without Ridge:
ŷ1 = 3,31593 + 0,11372 × 1 = 3,42965
ŷ2 = 3,31593 + 0,11372 × 3 = 3,65709
ŷ3 = 3,31593 + 0,11372 × 6 = 3,99825
ŷ4 = 3,31593 + 0,11372 × 9 = 4,33941
ŷ5 = 3,31593 + 0,11372 × 8 = 4,22569
RMSE = √((1,25 − 3,42965)² + (4,0 − 3,65709)² + (2,8 − 3,99825)² + (3,2 − 4,33941)² + (5,5 − 4,22569)² / 5)
= 2,02650
ŷ num (Ridge) = 1,81808 + 0,32376 Φ(y1, y2)
train with Ridge:
ŷ1 = 1,81808 + 0,32376 × 1 = 2,14184
ŷ2 = 1,81808 + 0,32376 × 3 = 2,78936
ŷ3 = 1,81808 + 0,32376 × 6 = 3,76064
ŷ4 = 1,81808 + 0,32376 × 9 = 4,73192
ŷ5 = 1,81808 + 0,32376 × 8 = 4,40816
RMSE = √((1,25 − 2,14184)² + (4,0 − 2,78936)² + (2,8 − 3,76064)² + (3,2 − 4,73192)² + (5,5 − 4,40816)² / 5)
= 2,15354

Aprendizagem 2024/25

Homework III – Group 066

(ist106794, ist1107301)



For the model without Ridge we can see that the train RMSE score is lower than the test RMSE score. This indicates slight levels of overfitting.

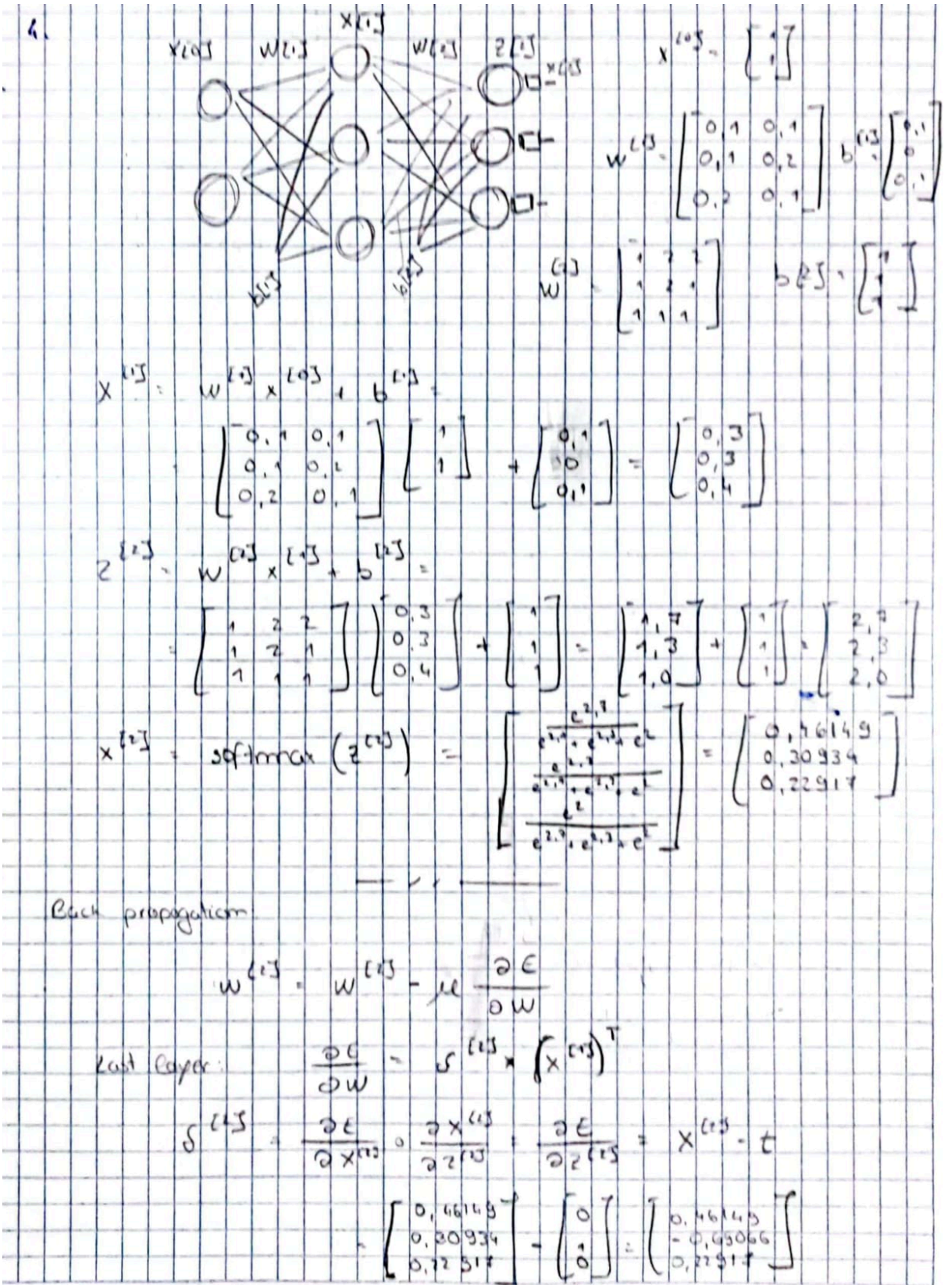
The train RMSE for the model after applying Ridge's regularization is higher when compared with the train RMSE without Ridge, as predicted. This happens because the model is more generalized and less fit to that training data. However, the RMSE score for the test with Ridge applied is significantly lower because of the effects of this regularization.

In conclusion, the results are expected. By using Ridge's formula, we were able to reduce test variance in the model.

Homework III – Group 066

(ist106794, ist1107301)

4)

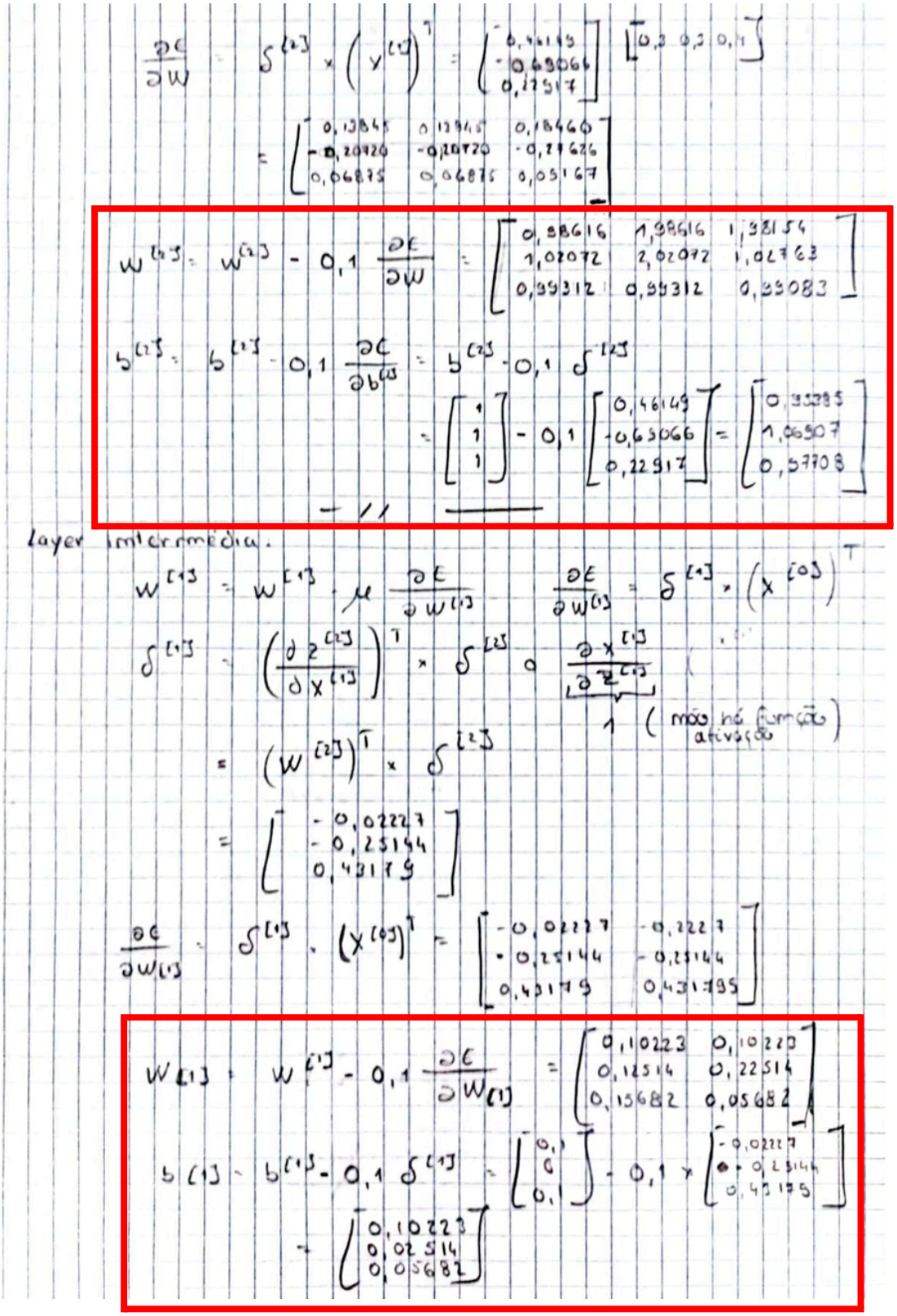


TÉCNICO LISBOA

Aprendizagem 2024/25

Homework III – Group 066

(ist106794, ist1107301)



Homework III – Group 066

(ist106794, ist1107301)

II. Programming and critical analysis

Here we have an overview of all the imports used and how we imported the dataset, which are both common to all questions. Therefore, we will be omitting this code from the beginning of each task to avoid redundancy.

```python
import pandas as pd, matplotlib.pyplot as plt, numpy as np
import seaborn as sns
import warnings

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_absolute_error
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

warnings.filterwarnings("ignore")

# Load the data
df = pd.read_csv('parkinsons.csv')
X = df.drop('target', axis=1)
y = df['target']
```

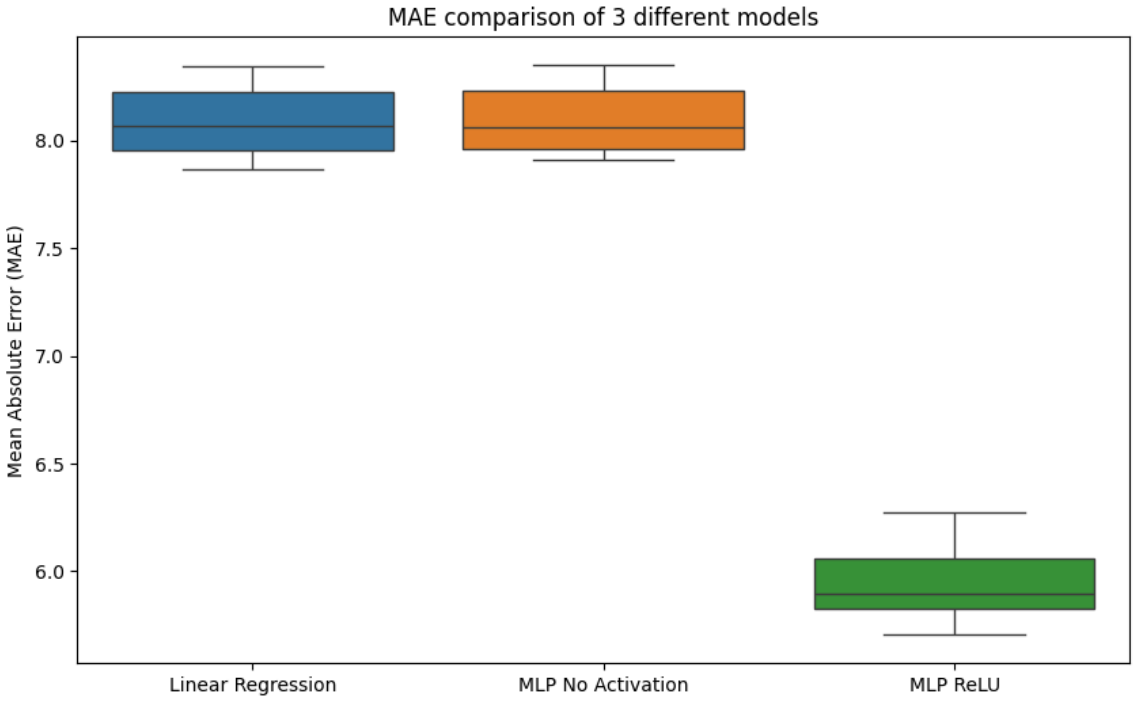
5) Performing 10 runs for each model and calculating the MAE for each one

```python
mae_linear = []
mae_mlp_no_activation = []
mae_mlp = []

for i in range(1, 11):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=i)

    # Linear Regression Model
    linear = LinearRegression()
    linear.fit(X_train, y_train)
    y_pred_linear = linear.predict(X_test)
    mae_linear.append(mean_absolute_error(y_test, y_pred_linear))

    # MLP Model no activation
    mlp_no_activation = MLPRegressor(hidden_layer_sizes=(10,10), activation='identity', random_state=0)
    mlp_no_activation.fit(X_train, y_train)
    y_pred_mlp_no_activation = mlp_no_activation.predict(X_test)
    mae_mlp_no_activation.append(mean_absolute_error(y_test, y_pred_mlp_no_activation))

    # MLP Model ReLU
    mlp = MLPRegressor(hidden_layer_sizes=(10,10), activation='relu', random_state=0)
    mlp.fit(X_train, y_train)
    y_pred_mlp = mlp.predict(X_test)
    mae_mlp.append(mean_absolute_error(y_test, y_pred_mlp))
```

Plotting the data

```
boxplot_data = [mae_linear, mae_mlp_no_activation, mae_mlp]

plt.figure(figsize=(10, 6))
sns.boxplot(data=boxplot_data)
plt.xticks([0, 1, 2], ['Linear Regression', 'MLP No Activation', 'MLP ReLU'])
plt.ylabel('Mean Absolute Error (MAE)')
plt.title('MAE comparison of 3 different models')
plt.show()
```

Output:



6) If we analyze both the Linear Regression and the MLP with no activation boxplots, we notice them to be very similar to each other.

This similarity makes sense if we consider the nature and calculus behind both models. The Linear Regression model is a simple model that captures linear relationships between the features and the target variable. On the other hand, the MLP model with no activation essentially is a more complex linear regression, despite having two hidden layers. Since there are no non-linear activation functions in the process, the model will not be able to catch other patterns or relationships in the data other than the linear ones, and so, the result will end up very similar, except with a lot of extra steps. Because of this, both models create two very similar MAE boxplots, as indicated in the first paragraph.

We can finally understand the activation functions' importance in introducing non-linearity to the model so that it can capture non-linear and more complex relationships between the features and, therefore, achieve a higher generalization capacity and better test results. Without them, MLP's would just be linear transformations of the input.

7)

Splitting the data, creating the MLP regressor, setting up the Grid Search and assessing the best combination of parameters

```
# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

# Grid Search parameters
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
}

# Create the MLP Regressor
mlp_regressor = MLPRegressor(hidden_layer_sizes=(10,10), random_state=0)

# Grid Search
grid_search = GridSearchCV(mlp_regressor, param_grid=param_grid,
                           scoring='neg_mean_absolute_error', cv = 5)
grid_search.fit(X_train, y_train)


# Evaluate the best model
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("Best hyperparameters found:", "\nL2_penalty:",grid_search.best_params_['alpha'], "\nLearning rate:", grid_search.best_params_['learning_rate_init']
      , "\nBatch size: ", grid_search.best_params_['batch_size'])
print("Test MAE of the best model: ", test_mae)
```

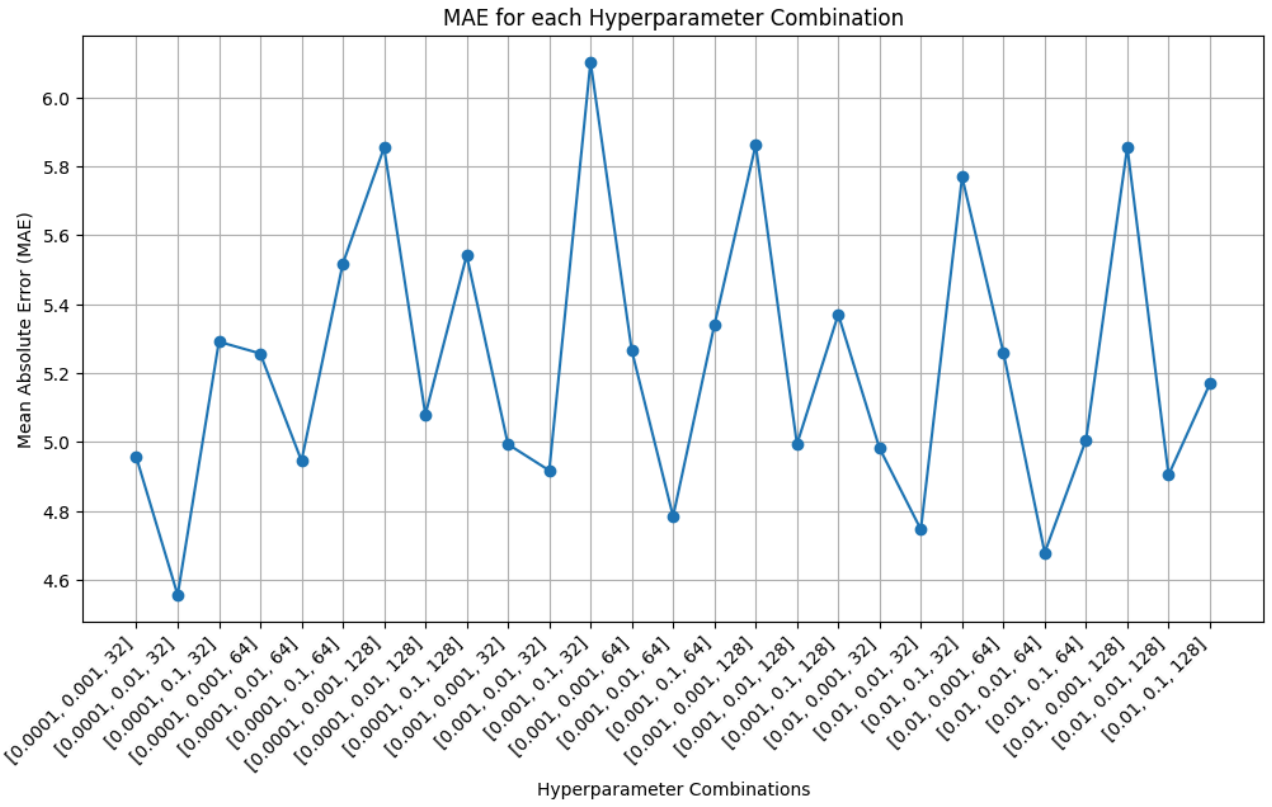
Plotting the MAE for every parameter combination

```
# Extract relevant columns
params = results['params']
mean_mae = -results['mean_test_score'] # MAE(since neg_mean_absolute_error is used)

# Labels
param_labels = [
    f"[{param['alpha']}, {param['learning_rate_init']}, {param['batch_size']}]"
    for param in params
]

# Plot the results
plt.figure(figsize=(12, 6))
plt.plot(range(len(mean_mae)), mean_mae, marker='o')
plt.xticks(range(len(mean_mae)), param_labels, rotation=45, ha = 'right')
plt.ylabel('Mean Absolute Error (MAE)')
plt.xlabel('Hyperparameter Combinations')
plt.title('MAE for each Hyperparameter Combination')
plt.grid()
plt.show()
```

Homework III – Group 066

(ist106794, ist1107301)

Output:



The best combination of hyperparameters observed after this experience is:

$$\{\text{L2 penalty} = 0.0001; \text{Learning Rate} = 0.01; \text{Batch} = 32\}$$

As for the trade-offs of each hyperparameter:

- L2 penalty (alpha):

  This parameter corresponds to the level of penalization to large coefficients we desire by applying Ridge's regularization in our model. Higher values of alpha correspond to more regularization.

  Therefore, we can associate overly lower values of this parameter with overfitting and low MAE scores in the test compared to the training, as low regularization is being applied, and excessively high levels with underfitting. The balance was found at alpha = 0.0001 in this model.

- Learning Rate

  This parameter represents how big are the updates the model does to its weights and biases on each iteration.

  Values too low might cause the model to take too much time to converge to the optimal solution while values too high might cause overshooting and divergence in the values.

  This was the hyperparameter that had the most impact on the MAE scores with Learning Rate = 0.01 being the optimal value for this model.

- Batch

  This parameter is the number of instances the model has in account before making the next update to the weights and biases.

  Generally lower values will be always better for the model's performance, since it means it is updating values more frequently. However they cause the training to take more time for the same reason, so the trade-off here is in balancing training time and effectiveness of the MLP.

  As expected, the model had better MAE scores for the lower value of batch = 32.

END