Machine Learning Coursework Report

Candidate Number: 277229

MSc. Data Science

May 2024

1 Performance

For this module's coursework, the assignment requested the construction of an MLP (Multi-Layer Perceptron) model to process the given data. Considering that the main purpose was to forecast the price of exporting crops for the following 3 years, a regression model was chosen.

In this section, the performance of the developed model is going to be evaluated. After carefully preprocessing the given data, a process that will be explained in detail in its own section in this report, the final dataset consists of **70,151 instances**. This data has been divided into training, validation, and test sets to ensure a proper assessment of the model's predictive abilities and its capacity to generalise to new, unseen data.

Initially, the total number of instances was divided, using **train_test_split()** alongside the following parameters:

1. **test_size = 0.25:** this splits the total data and allocates 25% of it to the test set.
2. **train_size = 0.75:** this ensures that the remaining 75% of the data is included in the training data.
3. **random_state = 1:** with this parameter, the split can be reproduced in the same manner multiple times.

After this initial split, the training set is divided one more time, to create the validation set. For this case, the parameters used were:

1. **test_size = 0.2:** since this split is only handling the previously allocated training data, this parameter allocates 20% of the training data (15% of the total data) to the validation set.
2. **train_size = 0.8:** indicates that the remaining 80% of the training data will be used as the final training set.
3. **random_state = 1:** is used in the same way as the first split, to ensure that this split can be reproduced multiple times, if necessary.

Finally, after performing these two splits, the resulting instances are:

- The **training** set is made up of **42,090 instances**, approximately 60% of the total, which was used to build and initially train the model.
- The **validation** set is made up of **10,523 instances**, about 15% of the total, utilised for hyperparameter training and to prevent overfitting.
- The **test** set is made up of **17,538 instances**, roughly 25% of the total, which provides a substantial basis for final evaluation and confirms the model's effectiveness.

After building the model, and using it with the given data, the overall performance has been measured using R-square (R^2) while the Mean Squared Error (MSE) was used to measure losses:

- **R-Square** is a statistical tool that provides a measure of how well the observed outcomes are replicated by the model, based on the proportion of total variance of outcomes explained by the model. The formula for R-Square is:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

  Where:

  - y_i is the actual value of the target variable
  - $\hat{y}_i$ is the predicted value from the model.
  - $\bar{y}$ is the mean of the actual values.
  - n is the total number of observations.

For this project, the model consistently shows an improvement when measuring the testing R-squared, over 100 epochs. It achieves the highest value of 0.60435 at epoch 85, indicating that the model can explain about 60.43% of the variance in the target variable from the testing dataset. This indicates that the model has an acceptable level of predictive accuracy, considering the type of data that is handling, however, there's still room for improvement. Even though the R^2 is relatively low, it is important to point out that it increases as the epochs progress, having started at 0.53200 in the first epoch, which shows that the model learns and adapts to the dataset.

- **Mean Square Error or L2** measures the average squared difference between the estimated values and what is estimated. The formula for L2 is:

$$L_2(\mathbf{w}) = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2$$

where:

  - w is the vector containing model parameters (weights).
  - X stands for the features matrix, with each row representing an input instance and each column representing a feature.
  - y is the vector of actual values observed within the dataset.

MSE is being used to measure test loss. In epoch 85, which was flagged as the best improvement, the test loss presented a value of 7.50585, when it started at 8.9164. The training and testing loss values show a trend of general decrease, suggesting that the model is learning effectively over time without showing signs of overfitting, as the testing loss decreases along with the training loss.

To provide a visual representation of the measurements taken regarding this model's performance, the following graph has been plotted:

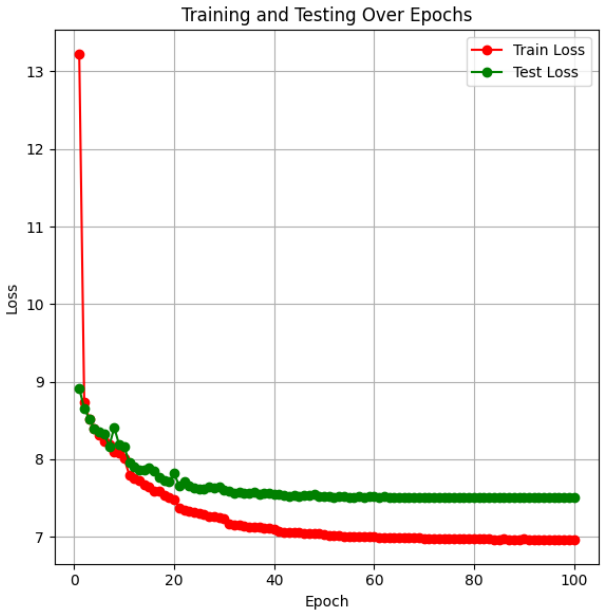

Figure 1: Training and Testing Over Epochs

The graph in Figure 1 shows the loss over epochs for both training and testing datasets. The training loss, represented in red, indicates how well the model fits the training data, while the test loss, represented in green, demonstrates how well the model performs on unseen data. The steep initial decrease in loss indicates that the majority of the learning happens during the earlier epochs before the model is stabilised.

Additionally, the stability of the test loss relative to the training loss, without significant increases or fluctuations, indicates effective generalisation by the model and suggests that it is not being overfitted. The continuous improvement across the training epochs shows the model's effective learning and adaptability to complex datasets.

2 MLP model

This section will explain in detail how the MLP model has been configured, its architecture, the loss function that is being used for training and the measures taken to prevent overfitting.

2.1 Model Architecture

2.1.1 Input Layer

The input layer is the first layer of the network and it receives the input data directly. In this model, the input layer has **76 units**, that correspond to the 76 features that are used to predict the target variable. This layer simply passes the normalized features to the first hidden layer, without any transformation

2.1.2 Output Layer

The output layer is the final layer of the model, responsible for producing the results. The objective of this layer is to ensure that the model's predictions are as close as possible to the target variable. The model aims to generalise by learning from the data, meaning that it could make accurate predictions on unseen data. For this project, the model's output layer consists of a **single unit**, where the output layer is designed to produce a single continuous value.

In MLPs, the **activation function for the output layer** shapes the form and range of the output the model can produce. For this layer there is no activation function, however, a linear combination of inputs and weights is used, which allows the model to use real continuous values. This linear combination is used through the **nn.Linear** method, and can be represented by:

$$y = w \cdot x + b$$

Where:

- **y** is the prediction of the model.
- **x** is the input to the output layer.
- **w** is the weight of the output neuron.
- **b** is the bias of the output neuron.

The output layer determines how the network uses the features processed by earlier layers to make predictions. During training, the predictions are regularly checked against the actual values, and the difference between them is measured with the loss function. This measurement is then used to adjust the network weights, through a process known as backpropagation. This adjustment helps the model get better at making predictions over time.

2.1.3 Hidden Layers

Another aspect to take into account when describing the architecture of an MLP model is the hidden layers, which are the layers between the input layer and the output layer. They are called "hidden" because they don't receive the external inputs or produce the final output; instead, they work internally to model the complexities in the data.

This model was designed with **two hidden layers**. The first one contains **128 units**, while the second one contains **64 units**. These units learn to recognise patterns or features in the input data, and then use them to predict the output.

The **activation function** that was used in both hidden layers for this project is ReLU (Rectified Linear Unit), which turns any negative input to zero and leaves any positive input unchanged. This helps avoid the issue of vanishing gradients, a problem where the updates to improve the network become so small that learning nearly stops. This means that the use of ReLU helps the model learn more steadily and quickly. ReLu's function is defined as:

$$f(x) = \max(0, x)$$

The configuration of hidden layers and the number of units impact the neural network's ability to learn and generalise. By selecting this number of layers and units, and by using this activation function, the model can effectively capture the patterns in the data, which results in accurate predictions.

2.2 Loss Function

As it has already been mentioned in the **Performance** section of this report, this model uses the **Mean Square Error** as its loss function; however, the first mention of this function was in reference to its use when measuring the model's performance. In this section, it will be explained in its use during the training process.

The most important role of a loss function is to guide the training process. By calculating the error between the model's predictions and the actual data, it provides a metric that the training algorithm can use to update the model parameters to reduce the error. The loss function is what the training process tries to minimise, so it affects how the weights of the network are updated during the training process.

$$L_2(\mathbf{w}) = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2$$

In particular, MSE captures the average squared difference between the estimated values and the actual values. Since the differences are squared, it generates bigger penalties for larger errors, because larger errors have a disproportionately large effect on the total loss.

2.3 Model Configuration

The **Adam optimizer**, which stands for Adaptive Moment Estimation, is used to adjust the neural network's attributes, like weights and learning rate, to minimise losses. To further enhance training, this project uses the **StepLR scheduler** to adjust the learning rate, reducing it by a factor of 0.5 every 10 epochs. As a visual representation of this configuration, a plot has been created, showing learning rate changes over epochs.

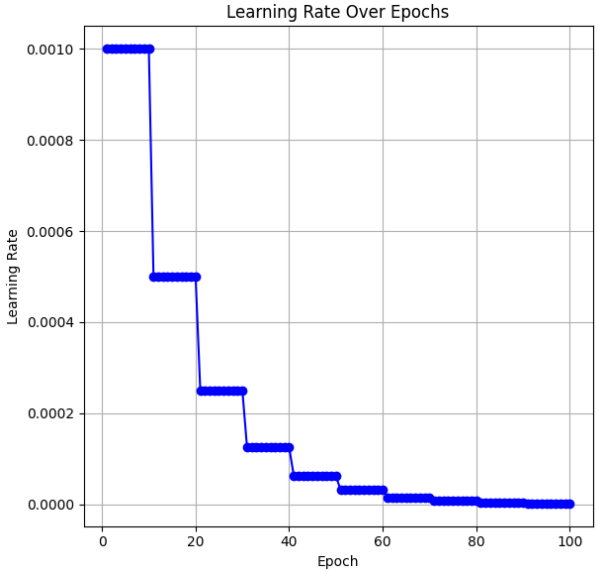

Figure 2: Learning Rate Over Epochs

The graph displays how the learning rate changes while the model is being trained. It is visible by the step-like drops in the graph, which show that the learning rate is reduced at certain times, proving the use of the learning rate scheduler. This helps the model start by learning quickly and then slow down as it gets closer to learning everything it needs.

2.4 Overfitting Prevention

Overfitting means that a model has learned the training data too well, and it's not capable of generalising and correctly predicting the behaviour of unseen data, because it hasn't actually learned the patterns the data is presenting.

The strategies that have been used to prevent this from happening in this project are:

1. **Data Splitting:** As it has already been explained, the data has been split into training, validation and testing sets. This allows the model to be trained on the training set, while it's improved with the validating set, and its performance is measured on unseen data.

2. **Learning Rate Scheduling:** As it has already been described, the model uses **StepLR** scheduler from PyTorch. This helps prevent overfitting by making smaller updates to the weights as the model is being trained, reducing the risk of overfitting by refining the learning process.

3. **Early Stopping:** This last measurement taken is not directly visible on the code, however it's something that has been done. The number of epochs used for this model has been chosen carefully, to avoid overfitting. This was done by monitoring the validation loss and stopping the training when this loss no longer improved.

3 Features & Labels

3.1 Label

This model was built to forecast the export value of crop products using a dataset that was divided into 13 different files. Those files contained information regarding various agricultural variables, such as crop production, land temperatures, land use, fertilisers, and pesticides. One particular file, named **Food Trade Indicators**, was of noticeable relevance for this project because it included explicit import and export values, denoted in its **Element** column. That's why this file was used as a primary reference for the target of this model since it showed a comprehensive list of crops and their export values throughout the year for all listed geographical regions.

The process to obtain the actual label that was used in the model involved several steps:

1. The **First File - Food Trade Indicator** was uploaded in the notebook as a data frame.

2. The rows in the file were filtered, and only the ones marked as **Export Value** were kept.

3. After merging this file with other relevant data into a single data frame called **files_combinados**, the column Value from this file was specifically named **Export Value**.

4. To align with the assignment's requirement to predict values three years into the future, an additional **Target** column was created using the **.shift(-3)** method.

5. As part of the preprocessing, which will be detailed in its own section of this report, all the numerical values went through a logarithmic transformation, to help normalise the distribution of variables that might be heavily skewed. For this task, the **np.log1p** method was used, which stands for "logarithm plus one":

$$\log(1 + x)$$

6. This transformation created a column in the data frame, called **Target_log**, which is the **only label** that was used in this model. This label represents the transformed target value shifted to 3 years in the future, as requested by the assignment.

3.2 Features

From the 13 files in which the data was divided, 7 were selected, including **Food Trade Indicators**, which has already been named as the source of the target column.

All the files that made up the dataset have some columns in common, which allow for the union of the data. These columns, that were present in all files, had information regarding the region, in the columns **Area** and **Area Code**; information regarding **Year**; information regarding a specific type of **Item**, corresponding to the subject that the file dealt with, and a **Value** for said item.

Each file presented information about a specific subject related to crop values, and within each file, there were different types of items. All these items were considered features since they directly or indirectly affect the export value of crops. To arrange the data in a usable manner for the model, the **.pivot_table** method was used. The result of this method, for each file, was the creation of a new data frame that consisted of **Area**, **Area Code**, **Year**, and as many columns as there were **Items** in that file, with their corresponding **Values**. This implementation created the additional issue of multiplying the number of NaNs, but the way this problem was dealt with will be explained in the next section of this report.

After all the information was retrieved from each file selected, and merged into a single data frame, there was an additional step that was required before the final features were selected. The numerical columns, with the exception of **Area Code**, needed to go through a logarithmic transformation, like the **Target** column, which resulted in a new column for each one.

The final result for feature selection shows a total of **76 features**, derived from:

- **Second File - Crops Production Indicators:** This file was selected for its detailed metrics on crops and livestock products:

  1. Feature_Cereals, primary_log
  2. Feature_Citrus Fruit, Total_log
  3. Feature_Citrus Fruit, Total_log
  4. Feature_Fibre Crops, Fibre Equivalent_log
  5. Feature_Fruit Primary_log
  6. Feature_Oilcrops, Cake Equivalent_log
  7. Feature_Oilcrops, Oil Equivalent_log
  8. Feature_Pulses, Total_log
  9. Feature_Roots and Tubers, Total_log
  10. Feature_Sugar Crops Primary_log
  11. Feature_Treenuts, Total_log
  12. Feature_Vegetables Primary_log

- **Third File - Consumer prices indicators:** This file was selected because it reflects consumer price indices and inflation data directly affecting food costs:

  13. Feature_Consumer Prices, Food Indices (2015 = 100)_log
  14. Feature_Food price inflation_log

- **Fourth File - Land use:** This file was selected because it presents insights into land utilisation, which affects agricultural output capacities:

  15. Feature_Agricultural land
  16. Feature_Agriculture_log
  17. Feature_Agriculture area actually irrigated_log
  18. Feature_Arable land_log
  19. Feature_Country area_log
  20. Feature_Cropland_log
  21. Feature_Cropland area actually irrigated_log
  22. Feature_Farm buildings and Farmyards_log
  23. Feature_Forestry area actually irrigated_log
  24. Feature_Land area_log
  25. Feature_Land area actually irrigated_log
  26. Feature_Land area equipped for irrigation_log
  27. Feature_Perm. meadows & pastures - Cultivated_log
  28. Feature_Perm. meadows & pastures - Nat. growing_log
  29. Feature_Perm. meadows & pastures area actually irrig._log
  30. Feature_Permanent crops_log
  31. Feature_Permanent meadows and pastures_log
  32. Feature_Temporary crops_log
  33. Feature_Temporary fallow_log
  34. Feature_Temporary meadows and pastures_log

- **Fifth File - Food balances indicators:** This file was selected because it provides information on food availability, losses, imports, and exports. Some of the items from this file were not included ('Alcoholic Beverages', 'Eggs', 'Meat', 'Milk', 'Fish', etc), since they were not related to crop food products:

    35. Feature_Cereals - Excluding Beer_log
    36. Feature_Fruits - Excluding Wine_log
    37. Feature_Oilcrops_log
    38. Feature_Pulses_log
    39. Feature_Spices_log
    40. Feature_Starchy Roots_log
    41. Feature_Sugar Crops_log
    42. Feature_Treenuts_log
    43. Feature_Vegetable Oils_log
    44. Feature_Vegetables_log

- **Sixth File - Fertilizers use:** This file was selected because it has information on fertilisers and it directly affects crops, in quantity and quality:

    45. Feature_Ammonia, anhydrous_log
    46. Feature_Ammonium nitrate (AN)_log
    47. Feature_Ammonium sulphate_log
    48. Feature_Calcium ammonium nitrate (CAN) and other mixtures with calcium carbonate_log
    49. Feature_Diammonium phosphate (DAP)_log
    50. Feature_Fertilizers n.e.c._log
    51. Feature_Monoammonium phosphate (MAP)_log
    52. Feature_NPK fertilizers_log
    53. Feature_Other NK compounds_log
    54. Feature_Other NP compounds_log
    55. Feature_Other nitrogenous fertilizers, n.e.c._log
    56. Feature_Other phosphatic fertilizers, n.e.c._log
    57. Feature_Other potassic fertilizers, n.e.c._log
    58. Feature_PK compounds_log
    59. Feature_Phosphate rock_log
    60. Feature_Potassium chloride (muriate of potash) (MOP)_log
    61. Feature_Potassium nitrate_log
    62. Feature_Potassium sulphate (sulphate of potash) (SOP)_log
    63. Feature_Sodium nitrate_log
    64. Feature_Superphosphates above 35%_log
    65. Feature_Superphosphates, other_log
    66. Feature_Urea_log

- **Seventh File - Pesticides use:** This file was selected because it has information on pesticides and crop health, which impacts viable export quantities:

    67. Feature_Fungicides and Bactericides_log
    68. Feature_Fungicides – Seed treatments_log
    69. Feature_Herbicides_log
    70. Feature_Insecticides_log

71. Feature_Insecticides – Seed Treatments_log
72. Feature_Pesticides (total)_log
73. Feature_Rodenticides_log

- **Common Features:** This is the data was present across all files:

74. Year_log
75. Area Code

- **Food Trade Indicator:** This represents the target variable's original values, before the 3-year shift.

76. Exchange Value_log

As a whole, the features were selected to capture a list of factors that influence crop export values, including production outputs, consumer prices, land usage, fertilisers and pesticides.

4 Preprocessing

There were several steps that needed to be taken before the model could be run, in order to prepare the given data.

4.1 File Selection

The first step taken for this project has been the selection of the appropriate files from the list of 13 files that make up the entire data frame. The final 7 files selected have already been mentioned in the previous section of this report, and they were selected taking into consideration the following characteristics:

1. **Agricultural Production:** The files that provided information regarding production indicators, of any type of crop, and their value, where included in the project.
2. **Market Prices:** The files that provided information regarding market prices, particularly the ones showing information regarding the prices of food, were included in the project.
3. **Land Use:** The files that provided information regarding the use of land in a region, regarding the amount of land that is used for agriculture, regarding the type of treatment the land was given, like pesticides and fertilisers, were included in the project.

4.2 Data Loading

All the selected files were loaded into the notebook, each on its own data frame. Alongside this initial step, some filtering was also performed, to align the data used in this project with the expected features for the model.

- **First File - Food Trade Indicators:** as it has been mentioned previously, this file contains the information regarding the Target Variable. It was loaded into the data frame **file_FoodTrade**. The information belonging to this file that was kept was:

    1. **Export Values:** the rows were filtered, and only the rows where the column **Element** had the value **Export Value**.
    2. **Columns Kept:** the only columns kept from this file were **Area**, **Area Code**, **Year**, and Value, which was renamed to **Export Value**.

- **Remaining Files:** All the remaining files were treated in a fairly similar manner. As has already been explained in the previous section, these files went through a **pivot** method, to divide the different values present in the column **Item** into different columns each. The columns used as indices for this process were **Area**, **Area Code** and **Year**, since they represent the merging columns for the entire data set. Two of the files needed extra filtering:

    1. **Food Balance Indicators**: some of the items present in this file did not interact with crops, such as eggs, meat and alcoholic beverages. It was decided that this information was not needed for the model, and so the rows were removed before the pivot table was created.

2. **Consumer prices indicators**: this file presented the extra complexity of presenting its information divided in monthly intervals instead of yearly. In order to unify the data input for the model, the **mean** of each region per year, was taken by item and was used to reduce the number of rows. The resulting data frame that was used for the creation of this pivot table only contains one row per year per type of item.

4.3 Data Merging

After all the pivot tables were created, and the export value data frame filtered, they were all merged into one single data frame, called **files_combinados**. The merging method used **Area**, **Area Code** and **Year** as indices since they represent the common data by which all the information can be combined. The merging method implemented an outer join, to ensure all relevant data was kept.

4.4 Missing Data

The creation of pivot tables included an additional challenge in this project. By creating new columns, aligning elements into the same rows, and merging different files, a lot of empty values, NaNs, were inserted into the data. In order for the model to be able to work, these values needed to be handled.

The methods chosen for this process were **ffill** and a secondary **bfill**. These methods propagate the last valid observation forward (or backwards) to the next valid one. An important consideration during this method was the need to group by the **Area** column since it would not be correct to utilise data from one region to fill the information of another.

4.5 Data Alignment

As part of the preprocessing methods used for feature and label selection, data alignment was performed, to ensure that each feature vector corresponded to the correct target value. This is essential to build an accurate and reliable model. An inner join was used, as it retains only the rows that have matching indices in both the feature and target datasets. This method guarantees that the data used for training and evaluation is aligned, preventing any issues that could impact the model's performance.

4.6 Normalisation and Scaling

To enhance the model's performance by minimising the influence of outliers in the model, the data was transformed, stabilising the variance and normalising the distribution of the features. To accomplish this, a logarithmic transformation **(np.log1p)** was applied to numerical columns.

$$\log(1 + x)$$

This transformation created mirroring columns of all numerical values, with the exception of **Area Code** which was left out because it's a categorical identifier. These new columns were identified with **_log** at the end of their names, and they were the ones that were introduced in the model as features.

For scaling, the **RobustScaler** was applied to the features. This type of scaling is less sensitive to outliers, making it appropriate for this data frame, with possible outliers and varying scales.

4.7 Additional Preprocessing Processes

Two additional processes that were involved in this process were **feature and label preparation**, and **data splitting**. Both of these processes have already been explained in previous sections of this report.

Alongside these processes, there were some methods used to retrieve the information in the original format, which dealt particularly with area code and the logarithmic transformation. Dictionaries were created, keeping track of indices and area codes, and the logarithmic transformation was undone, for both the test data and the predictions. This was done to produce a readable CSV file containing all the important outcomes of the model.