UNIVERSITY OF SUSSEX

Build MLP model that can be used for forecasting the export value of crop products

Module Code and Title: 934G5 Machine Learning

Module Convenor: Dr Temitayo Olugbade

Student Candidate Number: 283949

May 2024

Table of Contents

This report is for Machine Learning coursework at the University of Sussex. 13 datasets were provided for this machine learning project to implement a multilayer perceptron model (MLP) for forecasting the export value of crop products for a country three years into the future.

1. Performance

I used the MLP model as required in the specification for this project. Therefore, the performance of this model was evaluated by using three metrics which are Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Square Error (RMSE), and R^2Score. The main reason for using these metrics is to understand the accuracy and goodness of fit.

The Mean Absolute Error metric is for calculating the average magnitude of the errors in a set of predictions without considering their direction. The below equation is the formula for this metric:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n} | y_i - \hat{y}_i |$$

Equation 1: MAE formula

the y_i is the actual value and $\hat{y}_i$ is the predicted value in the above equation. Thus, in my project, y_i is the actual export value, and$\hat{y}_i$ is the predicted export value.

Another metric which is used in this project is Mean Squared Error. This metric is for measuring the average of the squares of the errors. Additionally, MSE is more sensitive to outliers than MAE. The equation 2 is the formula of MSE.

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

Equation 2: MSE formula

Root Mean Square Error was also used in this machine-learning project. RMSE metric is for the square root of the MSE, and it is for estimating the standard deviation of the prediction errors. The following equation indicates the RMSE calculation:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Equation 3: RMSE formula

The R^2 score used in this project to recognise how well the regression predictions approximate the real data points, and it is calculated as follows:

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

Equation 4: R Square formula

In the R^2 score equation, the $\bar{y}$ is the actual values.

The following figure indicates the performance metrics of this project after training and evaluating the model.

```
Mean Absolute Error: 354053.53580361645
Mean Squared Error: 1389012021748.4165
Root Mean Square Error: 1178563.541667744
R² Score: 0.9650922576603029
```

Figure 1: Performance of Metrics

As shown in Figure 1, the MAE value is 354,053, and the MSE value is around 1 billion, the RMSE is around 1 million and, R^2 score is 0.96. Thus, in my opinion, these values indicate that the model performs good in predicting the export values due to R^2 score which shows that approximately 96.51% of the variance in the export values is explained by the model.

The MAE value is a big number, but compared to the total export value, which is millions of Dollars in each country, then we can say that the number around 3 thousand is not a big number for error.

All of the utilised metrics of this project were obtained from the *sklearn.metrics* module. In other words, I did not create the explained metrics from scratch and used them from the *sklearn.metrics* module.

Additionally, the final merged data frame was split into training and testing sets. The split was done using an 80-20 ratio. This means that the training test comprises 80% of the total instances and is used to train the model. In addition, the training set in this project includes both features and labels. And test set is for comprising the remaining 20% of the instances and is used to test the performance of the model on unseen data.

The train-test split was achieved using the *train_test_split* function from the *sklearn.model_selection* module, as shown in the code below:

```
# Normalize the data
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=5)
```

Figure 2: code for train-test split

As shown in this code, the *random_state* parameter was set to 5 to ensure the reproducibility of the results.

2. MLP model

As mentioned, the MLP model is used to predict the export values. This section explains the architecture of this model and training details.

There is an input layer in the MLP model which has n unit. In fact, n is the number of features in the dataset. Moreover, there are hidden layers and an output layer as well. In my MLP model, the first hidden layer consists of 128 neurons with Rectified Linear Unit (ReLU) activation function. Then, the second hidden layer consists of 64 neurons with a ReLU activation function. And finally, the third hidden layer consists of 32 neurons with a ReLU activation function.

The output layer has 1 neuron because this is a regression problem where we predict a single continuous value. The activation function for the output layer is linear, which is implicit in the context of regression with Keras.

In another experiment, I added one more hidden layer which consists of 256 neurons with ReLU. Then, as shown in Figure 3, the MAE increased to 451,437, and the R^2 score decreased to 0.94. Therefore, adding a new hidden layer had a negative impact on the prediction.

```
Mean Absolute Error: 451437.1820805132
Mean Squared Error: 2168802824326.9883
Root Mean Square Error: 1472685.5823043112
R² Score: 0.9454950648433434
```

Figure 3: Performance of Metrics after adding a hidden layer

In addition, I compiled the model with the MAE loss function as Equation 1 in the first section. Then, I used the Adam optimizer with a learning rate of 0.01. In another experiment, I used the learning rate of 0.1. And then, the MAE reduced from 354,053 to 338,999 as determined in the figure below.

```
Mean Absolute Error: 338999.1334699865
Mean Squared Error: 1124724515405.0264
Root Mean Square Error: 1060530.299145209
R² Score: 0.97173415854423
```

Figure 4: Performance of Metrics with the learning rate of 0.1

The following code shows how the MLP model was assigned and compiled:

```
# Define the MLP model
model = Sequential()
model.add(Dense(128, input_dim=X_train.shape[1], activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(32, activation='relu'))
model.add(Dense(1))  # Output layer for regression

# Compile the model
optimizer = Adam(learning_rate=0.1)
model.compile(optimizer=optimizer, loss='mean_absolute_error')
```

Figure 5: the code for assigning and compiling the model

One of the important things in the machine learning is to prevent the overfitting. For this project, I used some strategies such as dropout regularization and validation split.

The dropout layers were added after the first and second hidden layers in this coursework. The dropout is a technique where randomly selected neurons which are ignored during training. This technique is to prevent the network from becoming too reliant on specific neurons and forces it to generalise better. For example, as indicated in the code in Figure 5, the dropout rate was set to 0.2 which means 20% of the neurons were dropped during each update cycle in this machine-learning project.

For the validation split, I used 20% of the training data as a validation set. By using this technique, the performance of the model can be monitored on unseen data during training and helps in early stopping if overfitting is detected.

The graph below shows the actual value against the predicted value of this machine-learning project:

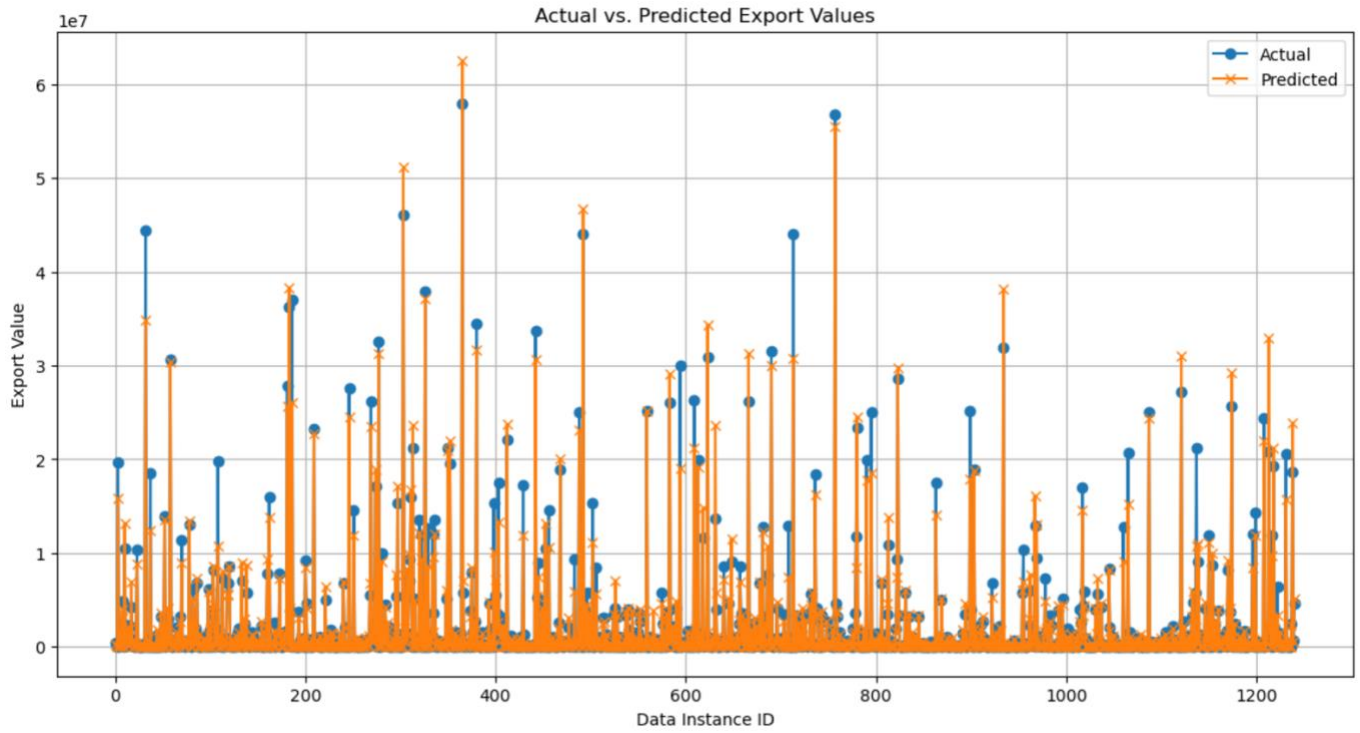

Figure 6: plot the actual value and predict the value

3. Features and Labels

The label or the target variable which is used for this machine learning model is a column in the final merged data frame and I called it *Export_Value_in_3_Years*. This label is for the value of exports forecasted three years into the future. This column was derived from the given dataset which is called *Food trade indicators*. The following data frame is *Food trade indicators:*

	Domain Code	Domain	Area Code (M49)	Area	Element Code	Element	Item Code (CPC)	Item	Year Code	Year	Unit	Value	Flag	Flag Description	Note
0	TCL	Crops and livestock products	4	Afghanistan	5622	Import Value	F1888	Cereals and Preparations	1991	1991	1000 USD	41600.00	A	Official figure	NaN
1	TCL	Crops and livestock products	4	Afghanistan	5622	Import Value	F1888	Cereals and Preparations	1992	1992	1000 USD	25600.00	E	Estimated value	NaN
2	TCL	Crops and livestock products	4	Afghanistan	5622	Import Value	F1888	Cereals and Preparations	1993	1993	1000 USD	40000.00	E	Estimated value	NaN
3	TCL	Crops and livestock products	4	Afghanistan	5622	Import Value	F1888	Cereals and Preparations	1994	1994	1000 USD	25700.00	E	Estimated value	NaN
4	TCL	Crops and livestock products	4	Afghanistan	5622	Import Value	F1888	Cereals and Preparations	1995	1995	1000 USD	37720.00	E	Estimated value	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1733	TCL	Crops and livestock products	716	Zimbabwe	5922	Export Value	F1896	Tobacco	2020	2020	1000 USD	794956.99	A	Official figure	NaN

Table 1: Food trade indicators data frame

There are import and export values in the element column of this data frame. Thus, I extracted the important columns ('Area', 'Element', 'Year', 'Unit', 'Value' and 'Item') from this data frame. Then, I filtered the dataset to only include export values by using df9[df9['Element'] == 'Export Value' where df9 is the *Food trade indicators* dataframe after extracting the main columns. Then, the result of this extraction is the following dataframe:

	Area	Year	Element	Value	Item
19	Afghanistan	2009	Export Value	15.00	Cereals and Preparations
21	Afghanistan	2010	Export Value	54.00	Cereals and Preparations
23	Afghanistan	2011	Export Value	0.00	Cereals and Preparations
25	Afghanistan	2012	Export Value	0.00	Cereals and Preparations
27	Afghanistan	2013	Export Value	0.00	Cereals and Preparations
...	...	...	...	...	...
141729	Zimbabwe	2018	Export Value	893113.05	Tobacco
141731	Zimbabwe	2019	Export Value	828488.44	Tobacco
141733	Zimbabwe	2020	Export Value	794956.99	Tobacco
141735	Zimbabwe	2021	Export Value	836533.69	Tobacco
141737	Zimbabwe	2022	Export Value	998057.60	Tobacco

Table 2: Data frame to only include export values

After filtering the export value, I used the unique function, and I understand that there are some items in the Item column which are not crop products. Therefore, I decided to filter the items to only include crop products. Then, I summed export values for each area and year across all crop items. Figure 6 shows this processing:

```
# List of crop-related items to keep
crop_items = [
    'Cereals and Preparations',
    'Fats and Oils (excluding Butter)',
    'Sugar and Honey',
    'Fruit and Vegetables',
    'Non-edible Fats and Oils', 'Tobacco'
]
```

```
# Filter the DataFrame to include only the crop-related items
filtered_export_df9 = extracted_col_export_df9[extracted_col_export_df9['Item'].isin(crop_items)]
```

```
# Sum export values for each area and year across all crop items
area_yearly_export_sums = filtered_export_df9.groupby(['Area', 'Year'])['Value'].sum().reset_index()
# Rename columns for clarity
area_yearly_export_sums.columns = ['Area', 'Year', 'Total Export Value']
area_yearly_export_sums
```

	Area	Year	Total Export Value
0	Afghanistan	1991	51858.00
1	Afghanistan	1992	19062.00
2	Afghanistan	1993	21324.00
3	Afghanistan	1994	26907.00
4	Afghanistan	1995	24240.00
...	...	...	...
6096	Zimbabwe	2018	992306.64
6097	Zimbabwe	2019	949699.72
6098	Zimbabwe	2020	934734.08
6099	Zimbabwe	2021	919869.29
6100	Zimbabwe	2022	1090008.89

Figure 7: Filter the items and sum export values

A new column is created by shifting the *Export_Value* column three years forward with grouping by Area. This shift is for predicting the export value three years into the future. This means that for each country in the dataset, the *Export_Value_in_3_Years* corresponds to the export value recorded three years after the current year. Thus, after shifting, the last three years for all countries became NaN in *Export_Value_in_3_Years column*. Then, by using forward fill and backfill functions, I filled NaN values in *Export_Value_in_3_Years* with the last available value per country and used it as the label or target column.

The data frame below shows the shift three years forward in *Export_Value_in_3_Years column*:

	Area	Year	Total Import Value	Total Export Value	Export_Value_in_3_Years
0	Afghanistan	1991	87160.0	51858.0	26907.0
1	Afghanistan	1992	95105.0	19062.0	24240.0
2	Afghanistan	1993	86236.0	21324.0	39173.0
3	Afghanistan	1994	69697.0	26907.0	44556.0
4	Afghanistan	1995	143826.0	24240.0	44243.0
5	Afghanistan	1996	121390.0	39173.0	38255.0
6	Afghanistan	1997	79548.0	44556.0	31080.0
7	Afghanistan	1998	65058.0	44243.0	27110.0
8	Afghanistan	1999	94935.0	38255.0	31153.0
9	Afghanistan	2000	168495.0	31080.0	47612.0

Table 3:Shift forward the target column

This process ensures that each instance in the training data has a corresponding export value to be predicted three years in the future. Thus, it allows the model to learn and forecast accurately.

I did the correlation with the target column, Random Forest model, and Recursive Feature Elimination to find the most relevant column to the target column. Therefore, the features which I used for training the model include a select set of agricultural and economic indicators, chosen based on their correlation with the target variable. In other words, I chose the first six columns which have the highest correlation to the target column.

Hence, the following features are which I used for training the model:

1. **Total Export Value**:

Description: This column is the value of agricultural products exported from a country.

Derivation: This column is evaluated by sum for each area and year and derived from the given dataset which is called *Food trade indicators*.

Relevance: This column is related to the target variable, as past export values can help in predicting future export trends.

2. **Total Import Value**:

Description: This column is the total value of agricultural products imported into a country.

Derivation: This column is evaluated by sum for each area and year and derived from the given dataset which is called *Food trade indicators*.

Relevance: Import values can influence local market conditions and future export strategies.

3. **Cropland**:

Description: This column is the total area of cropland used for growing crops.

Derivation: Directly included from the provided dataset which is called *Land use*. It is in the item column, and then by using the pivot, I put it as a column with its value.

Relevance: we can say that more croplands indicate higher production capacity, impacting export potential.

4. **Arable Land**:

Description: This column shows the total area of Arable land suitable for growing crops.

Derivation: This column is included from the provided dataset which is called *Land use*. It is in the item column, and then by using the pivot, I put it as a column with its value.

Relevance: The availability of arable land is a key factor in agricultural productivity and export potential.

5. **Agricultural Use of Pesticides (Total)**:

Description: The total quantity of pesticides used in agriculture.

Derivation: This column is included in the given data and by pivoting, put it in a separate column.

Relevance: we can say that pesticide use can affect crop yields and quality, influencing export volumes.

6. **Temporary Crops**:

Description: The area used for growing temporary crops.

Derivation: This column is included in the given data and by pivoting, put it in a separate column.

Relevance: Indicates the flexibility and short-term planning in agricultural production, affecting export planning.

4. Preprocessing

Preprocessing is a very important step in preparing the data for training a machine learning model.

1. **Filtering the Dataset**

The first thing which I did for preprocessing was data filtering. For instance, first, in each of the given datasets, I extracted the main important columns. The year and columns were common columns in all 13 given datasets. Therefore, these two columns were useful for merging my data frame after extracting the main columns and cleaning the data.

In most of the datasets, I filtered the most important elements or items. For instance, in the *Food balances indicators* dataset, I filtered all of the elements which are import Quantity, Export Quantity, Losses, Food, and Other uses (non-food).

```
# Filter the dataset to only include import quantity
import_quantity_df7 = filtered_crop_item_df7[filtered_crop_item_df7['Element'] == 'Import Quantity']
```

Figure 8: Example of filtering the data

2. **Summing Values:**

I did the sum of the values in some of the datasets. For example, I summed the quantities for each element across all crop items in each area and year in the *Food balances indicators* and *Food trade indicators* datasets. And, for mathematically we can say the following formula:

$$Total\ Quantity = \sum_{i=0}^{n} Quantity_i$$

Figure 9: Mathematically summing the quantity

The code below shows how I did the filtering and summing of the value for each area and year across all crop items:

```
# Filter the dataset to only include food quantity
food_quantity_df7 = filtered_crop_item_df7[filtered_crop_item_df7['Element'] == 'Food']
# Sum food quantity values for each area and year across all crop items
food_quantity_sums_df7 = food_quantity_df7.groupby(['Area', 'Year'])['Value'].sum().reset_index()
```

Figure 10: an example of filtering and summing the values

3. **Merging Data Frames:**

After making the separate the values of elements or items in each given dataset, then at the end, I merged all valued to the new data frame. And then, in the end, I merged all of the final data merged to the data frame which is called *final_df_merged*. The following code indicates how I did the merge all of the data frames:

```
# merge all of the data frames which I extracted from all of the datastes
important_df = [df9_merged, df1_merged, summed_yield_values_df2, pivot_df3, yearly_exchange_rate, pivot_df6, df7_merged, pivot_df8,
                df10_merged, df11_merged, pivot_df12, pivot_df13]

final_df_merged = reduce(lambda left, right: pd.merge(left, right, on=['Area', 'Year'], how='outer'), important_df)
```

Figure 11: merging all of the data frames

4. **Handling Missing Values:**

After merging all of the data frames, then I checked the missing values. Therefore, to handle the missing values, I checked which column has more than 50% NaN values, and then I decided to drop these columns.

```
# Calculate the percentage of NaN values for each column
missed_value_percentage = final_df_merged_cleaned.isna().mean()
# Filtering out columns where NaN percentages are greater than 50%
columns_to_drop = missed_value_percentage[missed_value_percentage > 0.5].index

# Dropping these columns from the DataFrame
final_df_merged_cleanedd = final_df_merged_cleaned.drop(columns=columns_to_drop)
```

Figure 12: dropping the column where NaN values are greater than 50%

After dropping those columns, then I decided to fill the NaN values with forward fill and then backfill.

```
# Fill NaN values with forward fill and then back fill
final_df_merged_cleanedd.fillna(method='ffill', inplace=True)
final_df_merged_cleanedd.fillna(method='bfill', inplace=True)
final_df_merged_cleanedd.isna().sum()
```

Figure 13: filling the NaN values

5. **Correlation**

After filling in the NaN values, then I did a correlation of all columns of my data frame with my target column which is *Export_Value_in_3_Years* to understand the relevance to the target variable.

The graph below shows the correlation of all columns with the target:

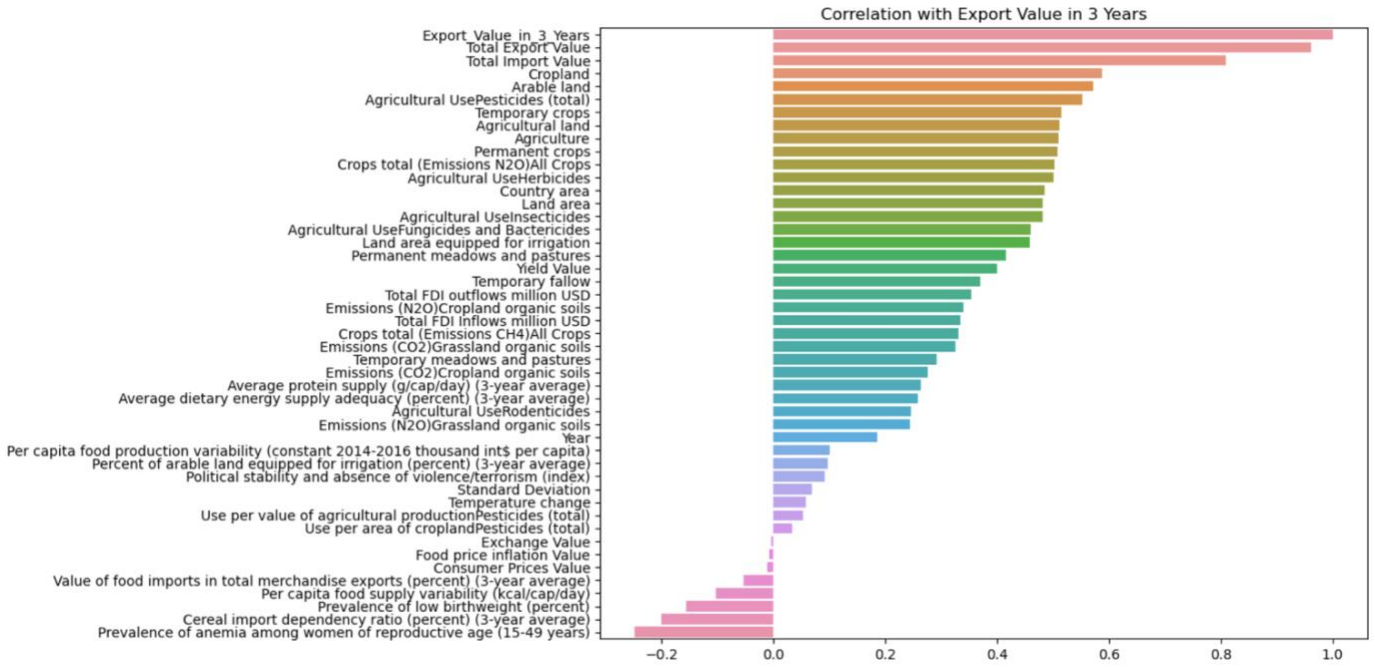

Figure 14: correlation with target

Then I printed out the first greatest correlation:

```
Export_Value_in_3_Years                    1.000000
Total Export Value                         0.961823
Total Import Value                         0.809837
Cropland                                   0.587885
Arable land                                0.571540
Agricultural UsePesticides (total)         0.553149
Temporary crops                            0.514624
Agricultural land                          0.511269
```

Figure 15: greatest correlation columns

6. **Feature Selection:**

I trained and tested the model with two types of feature selection. As explained, the first type takes only the first six columns which have the highest correlation with the target column.

Another method was Encoding Categorical Variables. It means that the categorical variables which is "Area" and "Year" were encoded using one-hot encoding to convert them into a numerical format suitable for the MLP model.

7. **Normalisation:**

I normalised all features by using *MinMaxScaler* to make all features on a comparable scale. In addition, normalisation is very important for neural networks as it helps in faster convergence and improves performance. The mathematical formula for *MinMaxScaler* which I used for this machine learning coursework is the following equation:

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Equation 5: MinMaxScaler formula

```
# Normalize the data
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)
```

Figure 16: normalisation

8. **Train-Test Split:**

As explained in section 1 of this report, The dataset was split into training and testing sets using an 80-20 ratio in this project. Thus, by this technique model can be evaluated on unseen data.