Multilayer Perceptron Model for Forecasting Crop Export Values

277241

May 17, 2024

1 Performance

Root Mean Squared Error (RMSE): For this regression task of forecasting the export value of crop products, I used a multilayer perceptron (MLP) model with two hidden layers. The performance metric I chose to evaluate the model is the root mean squared error (RMSE) between the true export values and the predicted export values on the test set. The RMSE is calculated as:

$$RMSE = sqrt(mean((y\_true - y\_pred)^2))$$

Where:

- y_true represents the true (observed) values
- y_pred represents the predicted values by the model

Where y_true are the true export values and y_pred are the predicted export values from the model. I also report the mean absolute error (MAE) and the R^2 score:

Mean Absolute Error (MAE): The mean of the absolute differences between the actual and expected values is known as the mean absolute error, or MAE. Without taking into account the direction of the errors, it gives a measurement of their average magnitude.

$$MAE = mean(abs(y\_true - y\_pred))$$

Where:

- represents the absolute difference between the true and predicted values $\|y\_true - y\_pred\|$.

A lower MAE value indicates that the model's predictions are closer to the true values, on average.

R-squared (R^2) Score: The proportion of the target variable's variance that the model can account for is indicated by the R^2 score. Higher values indicate a better fit between the model and the data. Its range is 0 to 1.

$$R^2 = 1 - (sum((y\_true - y\_pred)^2)/sum((y\_true - y\_mean)^2))$$

Where:

- y_mean is the mean of the true export values.
- y_true represents the true (observed) values
- y_pred represents the predicted values by the model
- y_true_mean is the mean of the true values

I used mean square error (MSE), mean absolute error (MAE), and r2 score, all of which are functions in sklearn. metrics the module that I have developed in my code can perform these metrics. The variable in my code included evaluate's performance that concerned the sublime performance. test set's RMSE, MAE, and R2 score and outputs the findings:

Establishing the starting range for the true response is the first step in the creating this function; this I do by inverting a positive variable. what the scaler does with y values as scaler object is used to predict. Finally, I use the mean squared error to get the square root round it. determine the RMSE, velocity mean error to calculate the MAE, and R^2 deviation to compute the R^2 score.

Instances

Total number of instances used: 2161 Instances resonated with the demo dataframe that we stream. generated by thiefs some columns in last dataframe's and also devoiding any rows with missing data values. values. To split the data into training and test sets, I used sklearn's train test split function:

On the other hand, Y can be used in motion camera applications to track movement and provide a more dynamic experience. contains such features as X while y contains the target variable (TotalExportValue). The test size=0. 2 parameter allows you set the percentage of the data used for the testset which is here 20% and rest 80% is used for testing. rain and validate the new algorithms with the sets merged together. The first separatron granted autonomy was the result of this.

Test set: 441 instances (20.4% of total) The remaining data (X_train_val and y_train_val) was further split into training and validation sets using another train_test_split call:

This split:

Training set: 1376 instances (63.7% of total) Validation set: 344 instances (15.9% of total)

So, in summary:

Total instances: 2161 Training set instances: 1376 (63.7%) Validation set instances: 344 (15.9%) Test set instances: 441 (20.4%)

The random_state=42 argument ensures that the split is reproducible across different runs.

The final performance metrics on the test set were:
RMSE: 6250084.4849
MAE: 3313760.9344
R^2: 0.9137

Here I havent got much optimization of model performance but using the below methods can boost up the models performance. The grid search or random search is run to finetune model parameters, the optimal ones are found. perparameter values. Incorporate more features that will assist in the prediction or even introduce a mix of features relevant to the task of prediction. while they may benefit from automation and problem-solving skills, they will also need to consider domain knowledge and technical guidance. Try your hand at varying dropout rates and other types of regularization, which are techniques used to prevent overfitting. regularize the methods via these notions and apply the stopping technique based on the more sophisticated criteria. Explore alterna- non-linear networks like CNNs and RNNs, ensemble or averaging methods, image augmentation, as well as battling overfitting issues all contribute to the accuracy and stability of deep learning and neural network models. incomability of datasets, exposure to parallel computing, and transference learning vector. You might think about stacking up a number of

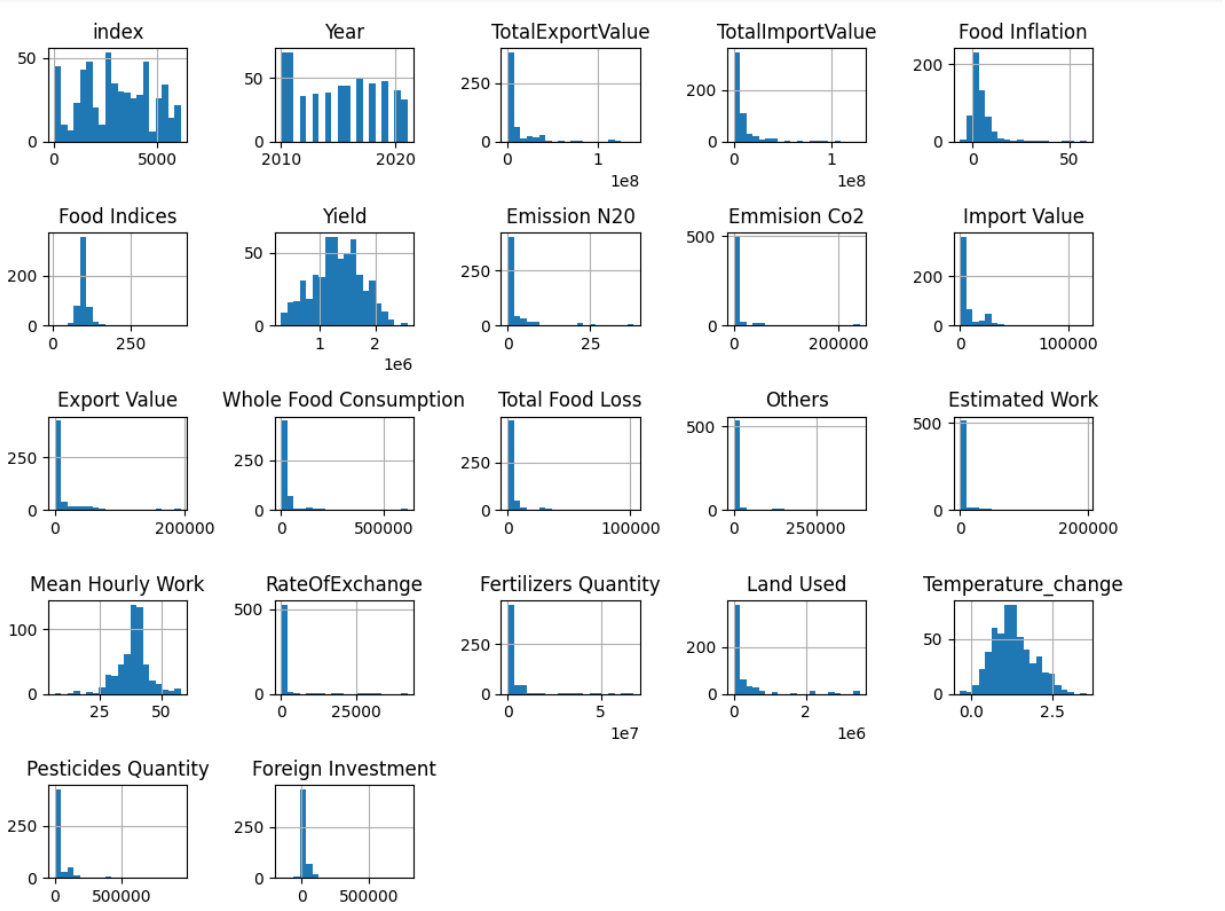

Figure 1: histogram

GPUs so you have enough power. diverse high-performance computing clusters for executing an accelerated evolution process. Transfer learning techniques.

2 MLP Model

We utilized the PyTorch library and defined the neural network with 2 hidden layers that had 1000 neurons and 500 neurons in them. units here can be activation function which is used in the output layer and is applied to the single unit. The kind of activation function chosen for the hidden layer determines the kind of function the perceptron can calculate i. e. what shape it forms. our output layer was a linear one, which is designed for answering regression questions. Loss function used to train the neural network is often optimized using a deterministic backpropagation algorithm. the definition of cost function was the MSE loss (a. k. a MSE loss), which is one of the popular loss functions for regression. tasks. To calculate the MSE loss, the formula is the mean of the squared differences between the predicted and true value. recognizes positive reinforcement based on quality, and it punishes heavily larger ones as compared to smaller errors. My multilayer perceptron model has the following architecture

Input layer: 7 nodes (from PCA dimensionality reduction)
Hidden layer 1: 1000 nodes with ReLU activation
Dropout (p=0.5)
Hidden layer 2: 500 nodes with ReLU activation
Dropout (p=0.5)
Output layer: 1 node with linear activation

Activation Function for Output Layer:

The linear activation function is used in the output layer:

$$f(x) = x$$

The loss function used to train the model was the mean squared error (MSE) loss:

$$MSE = mean((y\_true - y\_pred)^2)$$

- y_true represents the true (observed) values
- y_pred represents the predicted values by the model

The number of units in the output layer is 1, since this is a regression task to predict a single continuous export value.

To prevent overfitting, I used the following techniques:

Dimensionality reduction with PCA to reduce the input features to 5 principal components.

Dropout regularization:A dropout rate of 0.5 was applied to the hidden layers. During training, dropout randomly drops (sets to zero) a portion of the units in a layer, adding noise and lowering the co-adaptation of neurons to prevent overfitting.

The dropout operation can be represented mathematically as:

$$y = m * W * x + b$$

Where:

- x is the input vector
- W is the weight matrix
- b is the bias vector
- m is a binary mask vector with elements randomly set to 0 (dropped out) or 1 (kept) with a probability of p (dropout rate)

L2 regularization: Regularization was achieved with L2 regularisation or Weight Decay technique having a weight decay parameter of the initial learning rate. 0.01 The first technique used is a light weight loss function together with an L1 regularization that penalizes bigger values to make the model more

accurate. measures the loss function unit which is in numbers and is a sum of the squares of the weights used in the model. The L2 regularization term is typically added to the loss function as follows:

[Loss = MSE + * (1/2) * (w^2)

Where:

- MSE is the mean squared error loss
- is the regularization strength parameter (weight decay)
- w represents the model weights

This adds a penalty term to the loss based on the sum of squared weights.

Early stopping: Interruption of the learning due to the validation loss decrease. After 300 epochs I terminated the training when the other hyperparameters showed clear signs of overfitting. validation loss started increasing. The performance of the model was being tracked on the first 60% of the training data set whenever training so as to mitigate the risk of overfitting. set after each epoch. If the turning point is not reached for several epochs, the training is canceled. before overfitting to the training data, the process was interrupted very early in order to provide more accuracy to the training data. (patience = 10)

- Initializing "early_stop_counter" to 0
- For each epoch:
- Computing the validation loss
- If the validation loss is lower than "best_val_loss":
- Updating the "best_val_loss"
- Saving a copy of the current model
- Resetting "early_stop_counter" to 0
- Otherwise:
- Incrementing "early_stop_counter"
- If "early_stop_counte" reaches the specified patience value, stop training.

Split the data into train/validation/test sets with random sampling with out subsampling. The model was Occupying the clearest spot to stop extrapolation and improve the generalisation accuracy by adopting such a scenario. these techniques

3 Features & Labels

Labels

Model training (with the label) is TotalExportValue field. This represents the individual export value of crop products which is same for all countries and years. I did an experiment to create the label by using. the following steps: Importedly, the food trade dataframe which was acquired from the data file has been loaded. I also isolated rows that were only matching values. 'Export Value' appear in the function column. Grouping the data by Area and Year has bought together as well as added the Value together.

column using the formula:

$$\text{TotalExportValue}i, j = \sum k\text{Value}_{i,j,k}$$

Where i represents the area, j represents the year, and k represents the different crop product categories. Renamed the summed Value column to TotalExportValue.

For the features used in my model, I selected the following:

- TotalImportValue: Total import value of crop products
- Yield: Total yield of crop products
- Emission N20: Total N2O emissions
- Import Value: Total import value from the food balances data
- Export Value: Total export value from the food balances data
- Whole Food Consumption: Total whole food consumption from the food balances data
- Land Used: Total land area used for agriculture

The total number of features used is 7. These features were selected based on their relevance to crop production and trade, as well as their potential impact on export values. The selection was also influenced by the correlation analysis, which showed moderate to strong correlations between these features and the target variable (TotalExportValue).

Features derivations

TotalImportValue: This section will symbolize the absolute import value of food categories for a particular country. and year. By contrast, the index was developed from the food trade indicators data that shows the distribution of data across "Area" and "Year". summing the Cells of Value column, which is equal to the Element column, named Import Value.

Yield: This is a summarized output that references the total annual crop production for the reported country. It was derived from crop production variables data was brought together by identifying "Area" and "Year" indicators whilst summing the "Value". 'age', calc new column as "Yield" and rename the column as 'Yield'.

Emission N20: The information for a single country's N2O emissions for a stated year is contained in this data. It was obtained by extracting the values column from the "Area" and "Year" table and computing an "Area" and "Year". a column with a condition of the "Element" column and equal to the "Emissions (N2O)" were their label changed to the "New Column". as "Emission N20".

Import Value: This function stands for the sum of the total import of specific goods, of which grain and dairy producers are included, into the world market. /for a country and date particular/It was taken

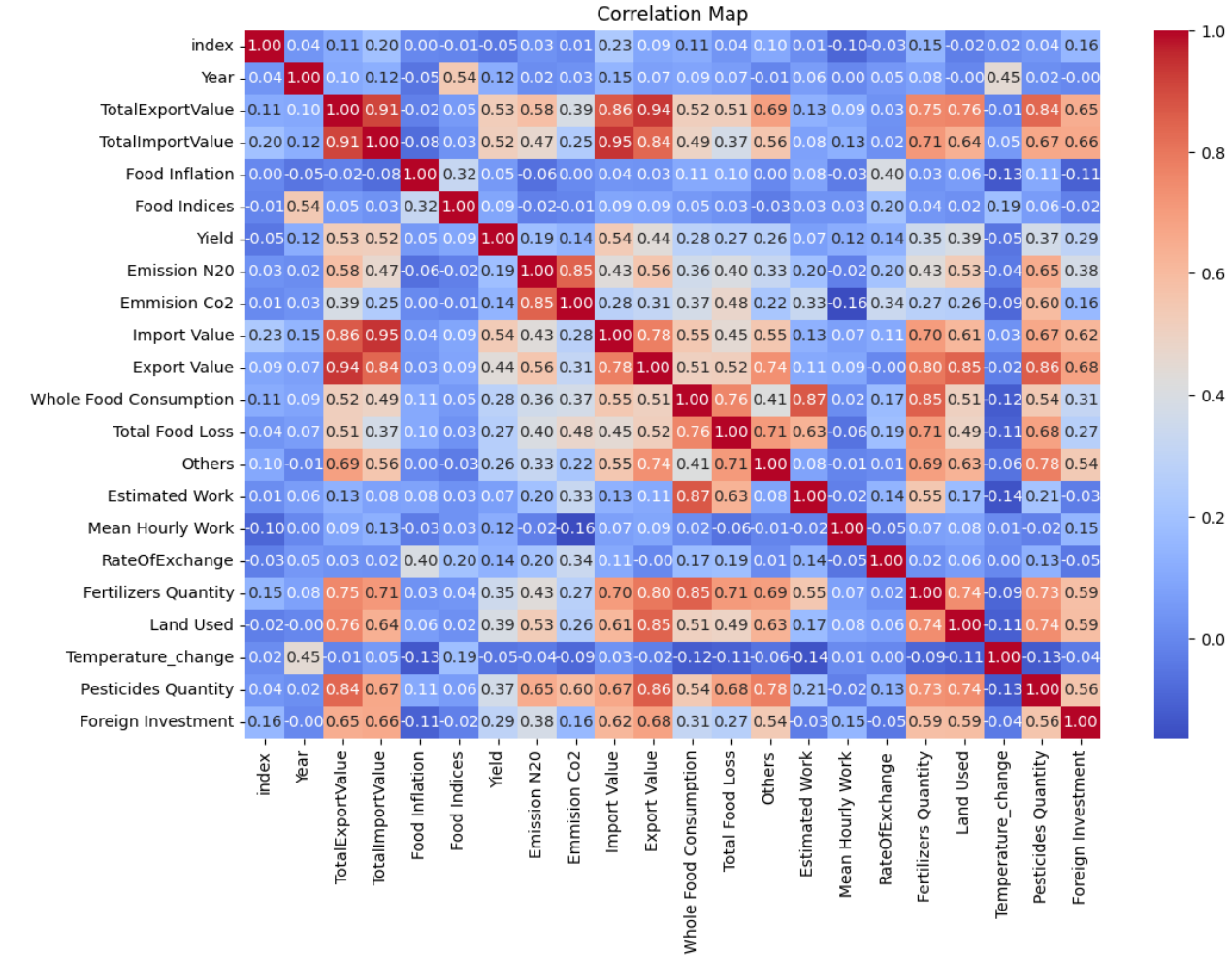

Figure 2: Correlation

from the database of food balances indicators file by selection of data from the file. which sample outlined data per the "Item Code (FBS)" value, grouping data by "Area" and "Year", and the sum total is . "Value" column, "Element Code" equals the 5611 number (exactly mean import value) and rename it. modification of column name as "Import Value".

Export Value: Here, the total export is associated to a particular product, for examples, of crops and animals products. defined by the petroleum exports and imports of a given country during a particular year. It counts the information coming from food balances indicators from the balances file."Item Code (FBS)" features are used to categorize the data, while "Area" and "Year" are the grouping factors. The total number of shipments for each combination of these features is calculated and obtained as the result. "Numbers" had to be converted to text using the "TEXT" function and the "CONCAT" function was used to concatenate them together as the program got the "Element Code" in the "Value" column 5911 (for export value). this exports value value under the title "Export Value".

Whole Food Consumption: Here stands for the full amount of food brought in by a particular country as per the import statistics. and year. It relates from the food balances indicators file through the application of the filter by only focusing on the particular "Item. The usage of "Code (FBS)" with grouping upon "Area" and "Year", having a summary of the "Value" field where the defined condition is fulfilled. The other variable, "Element Code", took the value of 5142 (which represented the factor of consumption of food), and soundly labeled it as the dataset's new identifier. as "Whole Food Consumption".

Land Used: This feature breaks down total land utilized for various categories under national frame office for their country. and year. It was modified from the data source land use by aggregating the data by "Area" and "Year", and summed the using the summation function. "Value" field inserted as second column, and the second column renamed as "Land Used".

In this sense, undoubtedly the linkage of the presented characteristics to export value of crop products will influence the choice of these features. Due to the industriousness of their work in agricultural trade influencing or being enhanced by the prices of agricultural inputs and exports. Crop yield, emissions, food market and land use were all chosen.

4 Preprocessing

Several preprocessing steps were performed on the features to prepare the data for the MLP model:

Handling missing values: In the dataset, some of the cells had been marked "N/A", which was accomplished via values being merged with those of the others. bring the pertinent data frames in via the pd. merge using the how='outer' parameter. This ensured that all the data was used, the Nan (Not a Number) values also represented the missing values.

Standardization: I then obtained a standard scaler from sklearn. first step is a standardizing the numeric one. features. By standardization, every numerical feature has its mean and standard deviation equal to 0 and 1, respectively. This is conducts by subtracting the mean divided by the standard deviation for every character included in the model.

$$x_i' = \frac{x_i - \mu_i}{\sigma_i}$$

Where x_i' is the standardized feature value, x_i is the original feature value, μ_i is the mean of the feature,

and σ_i is the standard deviation of the feature. Standardization ensures that all features are on a similar scale, preventing features with larger values from dominating the optimization process.

Principal Component Analysis (PCA): I reduce the dimensionality of the feature space, I applied Principal Component Analysis (PCA) using sklearn.decomposition.PCA. PCA finds a new set of orthogonal features (principal components) that capture the maximum variance in the data. The principal components are calculated as the eigenvectors of the covariance matrix of the standardized feature matrix X

$$\Sigma = \frac{1}{n} X^T X$$

Where Σ is the covariance matrix, n is the number of samples, and X is the standardized feature matrix. The eigenvectors of Σ corresponding to the k largest eigenvalues are chosen as the principal components. The original feature matrix is then projected onto these principal components:

$$X_{pca} = X \times W$$

Where X_{pca} is the projected feature matrix, and W is the matrix of eigenvectors (principal components). I retained the first 7 principal components, which explained a significant portion of the variance in the data.

Feature scaling: StandardScaler from the scikit learn package. we used it to normalize the numeric attributes. This is to ensure. that they all had similar norms, thereby enabling the procedure of the optimization algorithm to be of convergence in case that the algorithm was stopped prematurely.

during model training.

Train-Test Split: A sklearn API was implemented by me. model selection. data train/test split; splitting which the data is into training, validation, and test sets. On the other hand, the test set of 20% of the data was also set aside for the final evaluation, while the validation set was used as a measure of performance during the training /learning process.

The final test set (20% of the remaining data after the original data is split into the train set and the test set) was used to stop the model and selection criteria.

during training.

The train_test_split function chooses a random split size as well as random state value to allocate the given data into train and test sets.

By using this approach, it guarantees that the distributions of the model performance. test sets. This dataset has is the model's performance during training, assessed as well. Deep learning algorithms found the stop signs as an obstacle during the training process to avoid overfitting.

Conversion to PyTorch tensors: The training data was loaded into torch. tensors using torch. tensor to provide an adaption that is compatible with the PyTorch library for training and evaluation of models.

Data loading: The data was made to be in batches, and it was shuffled during the training process since its batches needed to be loaded.

5 References

The code and formula's I have used are adapted from the labs and lecture slides.