Artificial Neural Network Midterm Hackathon Session 2 (December, 2023)

Furkan Bulut 210316011

Manisa Celal Bayar University Faculty of Engineering

Manisa, TURKEY

210316011@ogr.cbu.edu.tr

***Abstract*— This research aims to estimate exchange rates by using information on other foreign currencies. In recent years, the dynamics of global financial markets have become increasingly interconnected, necessitating more complex approaches to exchange rate forecasting. This article proposes a methodology based on multi-currency analysis to improve the accuracy of forecasts. The report discusses the introduction of the problem, the chosen methodology, alternative approaches, possible fixes, and concludes with recommendations for the future.**

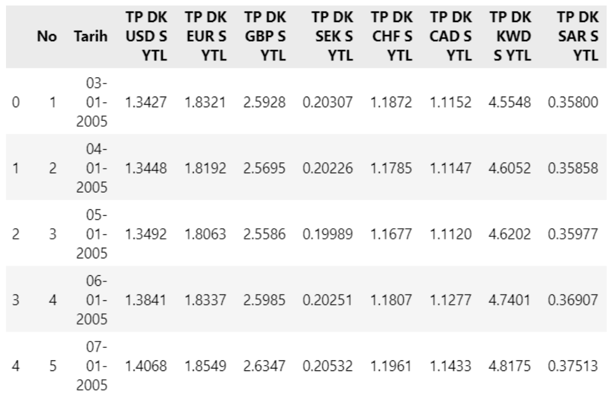
I. Introduction

Foreign exchange markets play an important role in the global economy, serving as the cornerstone of international trade and investment. The ability to accurately predict foreign exchange values is of great importance to businesses, investors and policy makers. In this context, the challenge ahead is to develop effective models to predict currency movements. Specifically, the focus is on leveraging the relationships between various foreign currencies to estimate the value of the target currency. Our aim is to improve forecasting methods and deepen our understanding of the ever-changing global financial landscape by exploring the intricacies of predicting currency values through the relationships between different currencies.

II. Methodology

A. Data Preprocessing

This document outlines the data preprocessing steps applied to a financial dataset. The procedures encompass data loading, structural refinement, column renaming, and data quality assessments. The dataset was cleaned by removing unnecessary records, handling non-numeric values, and addressing missing data, culminating in a standardized, numeric representation suitable for analysis.



	No	Tarih	TP DK USD S YTL	TP DK EUR S YTL	TP DK GBP S YTL	TP DK SEK S YTL	TP DK CHF S YTL	TP DK CAD S YTL	TP DK KWD S YTL	TP DK SAR S YTL
0	1	03-01-2005	1.3427	1.8321	2.5928	0.20307	1.1872	1.1152	4.5548	0.35800
1	2	04-01-2005	1.3448	1.8192	2.5695	0.20226	1.1785	1.1147	4.6052	0.35858
2	3	05-01-2005	1.3492	1.8063	2.5586	0.19989	1.1677	1.1120	4.6202	0.35977
3	4	06-01-2005	1.3841	1.8337	2.5985	0.20251	1.1807	1.1277	4.7401	0.36907
4	5	07-01-2005	1.4068	1.8549	2.6347	0.20532	1.1961	1.1433	4.8175	0.37513

Fig. 1. Sample Data From Given Dataset

1) Data Loading and Initial Inspection:
The dataset was successfully loaded, and an initial inspection was conducted, with potential loading errors addressed through exception handling.

2) Cleaning the Dataset:
a. Structural Refinement: Descriptive initial rows were skipped, enhancing dataset clarity.
b. Column Renaming: Columns were standardized for improved interpretability.
c. Record Removal: Redundant information from the first row was dropped for dataset streamlining.

3) Data Quality Assessment:
a. Identification of Non-Numeric and Missing Values: Non-numeric columns and missing values were identified to gauge data quality.

4) Handling Non-Numeric and Missing Values:
a. Non-Numeric Values: Columns with non-numeric data types were converted to numeric, ensuring uniformity.
*b***.** NaN Handling: Rows with missing values were removed to eliminate noise in the analysis.

5) Data Type Standardization:
a. Numeric Conversion: Relevant columns were uniformly converted to numeric data types for consistency.

8) Final Data Quality Check:
a. Post-Processing Data Quality: A final check ensured the elimination of non-numeric values and missing entries.

9) Conclusion:
The outlined preprocessing steps contribute to data quality, preparing the dataset for meaningful analysis by addressing structural inconsistencies, non-numeric values, and missing entries.

B. Feature Selection

In the realm of predictive modeling, the art of feature selection plays a pivotal role in crafting robust and efficient models. Feature selection is a strategic process involving the identification and retention of the most influential variables while discarding redundant or irrelevant ones. Its primary objectives include enhancing model performance, mitigating the risk of overfitting, and expediting model training.

1) Independent Variables (Features):
The identification of pertinent features is a crucial step in building predictive models. In this scenario, the following currency exchange rates were discerned as potential predictors:
- USD (TP DK USD S YTL)
- EUR (TP DK EUR S YTL)
- GBP (TP DK GBP S YTL)
- SEK (TP DK SEK S YTL)
- CHF (TP DK CHF S YTL)
- CAD (TP DK CAD S YTL)
- KWD (TP DK KWD S YTL)

2) Dependent Variable (Target):
The target variable, embodying the currency exchange rate for Saudi Riyal (SAR), was defined as:
- SAR (TP DK SAR S YTL) (Output)

3) Significance of Separation:
The dataset underwent a meticulous partitioning into independent variables (X) and the dependent variable (y). This segregation sets the stage for subsequent model training and evaluation.

4) Preview of Variables:
A sneak peek into the initial five entries of both independent variables (X) and the dependent variable (y) is provided, offering a glimpse into the dataset's structure.

C. Cost/Lost Function

The Mean Squared Error (MSE) is a fundamental metric in the realm of regression analysis, providing a quantitative measure of the accuracy of a predictive model. The MSE is computed by taking the average of the squared differences between actual and predicted values. This penalizes larger errors more heavily, offering a comprehensive assessment of the model's performance.

The `mean_squared_error` function presented here is a concise implementation of MSE calculation. It takes the true values (`y_true`) and the predicted values (`y_pred`) as inputs and returns the average of the squared differences.

In the specific context of the code snippet, the MSE is calculated for a set of actual and predicted values stored in a DataFrame (`results_df`). The variables 'Actual' and 'Predicted' within the DataFrame correspond to the true and predicted values, respectively. The calculated MSE (`mse_all`) provides a consolidated measure of the model's performance across the entire dataset.

D. Derivate of Cost/Lost Function

In machine learning optimization, understanding the derivative of the Mean Squared Error (MSE) function is crucial, especially in the context of gradient descent-based algorithms. The derivative represents the slope or rate of change of the MSE with respect to the model parameters. The `derivative_mse` function provided here computes this derivative, facilitating the optimization of model coefficients.

1) Function Explanation:
The `derivative_mse` function takes three parameters: `y_true` (true values), `y_pred` (predicted values), and `X` (independent variables). It employs the formula:

$$\text{Derivative of MSE} = \frac{-2}{\text{len}(y_{\text{true}})} \times X^T \times (y_{\text{true}} - y_{\text{pred}})$$

This expression represents the gradient of the MSE function with respect to the model parameters, providing insights into the direction and magnitude of adjustments needed during optimization.

2) Application to Data:
In the code snippet, the `derivative_mse` function is applied to the actual and predicted values stored in the DataFrame (`results_df`). The result (`mse_derived`) signifies the gradient of the MSE function with respect to the model parameters for the given dataset.

3) Interpretation:
The negative sign in the formula indicates the direction of steepest descent. The computed derivative guides the

adjustment of model parameters, minimizing the MSE and refining the model's predictive capabilities.

E. Optimizer Function

In the quest for robust machine learning models, the introduction of regularization techniques becomes paramount to mitigate overfitting and enhance generalization. The `MultipleLinearRegressionRegularized` class presented here encapsulates a Multiple Linear Regression (MLR) model augmented with regularization, specifically a combination of L1 (Lasso) and L2 (Ridge) regularization.

1) Model Parameters:
The model is equipped with several hyperparameters:
- `l1_ratio`: Proportion of L1 regularization.
- `alpha`: Regularization strength (violence).
- `epochs`: Number of training iterations.
- `learning_rate`: Step size for gradient descent.

2) Training Process:
The `fit` method employs gradient descent to iteratively update model coefficients. Noteworthy steps include:
- Adding a bias column to the independent variables (X).
- Initializing coefficients.
- Iteratively updating coefficients using gradient descent.
- Incorporating L1 and L2 penalties to mitigate overfitting.

3) Regularization Mechanism:
- The L1 penalty (`l1_penalty`) enforces sparsity in the coefficients by introducing the absolute values of the coefficients.
- The L2 penalty (`l2_penalty`) penalizes large coefficients to prevent overfitting.
- Both penalties are integrated into the gradient calculations and contribute to the iterative coefficient updates.

4) Predictions:
The `predict` method utilizes the trained coefficients to make predictions. Similar to the training phase, a bias column is added to the input variables (X) before applying the dot product with the coefficients.

5) Implementation:
The code snippet initializes and trains the regularized MLR model (`mlr_Regularized`) on the provided data (X, y). Predictions are then made on a subset of the training set (`X.values[:5]`).

6) Optimization Mechanism:
Regularization aids in model optimization by constraining the coefficients, preventing them from becoming excessively large.

F. Creating a Machine Learning Model

In the realm of machine learning, the development of predictive models is a cornerstone of data-driven decision-making. The model is trained on a dataset comprising selected independent variables (features) and a corresponding dependent variable (target).

1) Model Architecture:
The MLR model is encapsulated within the `MultipleLinearRegression` class, offering a modular and organized structure. The model's core attribute, `coefficients`, serves as a repository for the calculated regression coefficients.

2) Training the Model:
The `fit` method is employed to train the model using the least squares method. This involves the addition of a bias column to the independent variables (X) and the application of normal equations to determine the optimal coefficients (`beta`). The training process culminates in the assignment of these coefficients to the `self.coefficients` attribute.

3) Making Predictions:
The `predict` method utilizes the trained coefficients to make predictions for new data. Similar to the training phase, a bias column is added to the input variables (X) before applying the dot product with the coefficients.

4) Implementation:
The code snippet initializes the MLR model, fits it to the training data (X, y), and subsequently makes predictions on a subset of the training set (`X.values[:5]`). The predicted values are stored in the `predictions` variable.

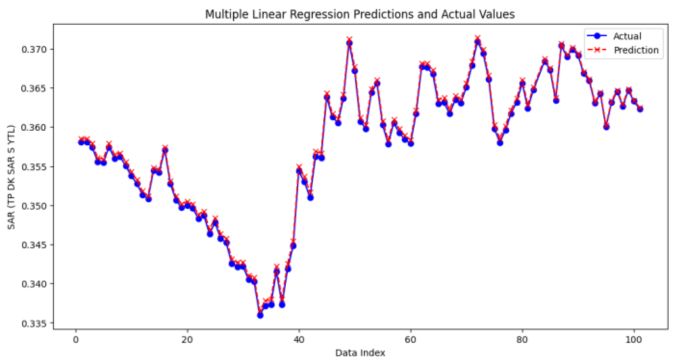


Fig. 2. Multiple Linear Regression Predictions and Actual Values Graph.

5) Insights from Predictions:
The `predictions` variable encapsulates the model's output, providing insights into how well the trained MLR model performs on a subset of the training data.

G. Model Evaluation and Testing

1) User Interaction:
- Users are prompted to input values for each independent variable, creating a dynamic and personalized forecasting experience.
- The script ensures the user's input is of numeric type, promoting robustness and preventing potential errors.

2) Model Prediction:
- The user-provided values are converted into a numpy array and used as input for the pre-trained MLR model.
- The model processes the input and generates a predicted output, representing the forecasted currency exchange rate.

3) Result Presentation:

- The predicted output is then displayed to the user, offering insights into the anticipated currency exchange rate based on their input.

III. Discussion

Although our current approach is promising, ensuring the robustness of our findings warrants exploration of alternative methods. An effective alternative involves utilizing time series analysis specifically designed for foreign exchange forecasting. This method allows for a nuanced examination of historical currency data over sequential time periods, capturing patterns and trends that our current multiple linear regression model does not fully capture. By adopting time series analysis, we improve our ability to account for temporal dependencies in currency fluctuations and potentially improve forecast accuracy.

Additionally, combining financial news and social media sentiment analysis emerges as an interesting boost. This process involves systematic evaluation of textual data to discern prevailing sentiments towards public currencies. By integrating sentiment analysis, we gain valuable insight into the mood of the market, which influences trading decisions and currency values. This additional layer of information contributes to a more contextually informed forecasting model, enriching our understanding of the psychological factors that shape market dynamics. In summary, alternative methods such as time series analysis and sentiment analysis broaden our research scope, acknowledging the multifaceted nature of foreign exchange forecasting and significantly contributing to model robustness.

IV. Conclusion

Consequently, our research addresses the challenge of forecasting foreign currency values using multicurrency analysis. Although our current methodology is promising, we have discussed alternative approaches such as time series analysis and sensitivity analysis that can further improve the accuracy of our forecasts. These alternatives add depth to our understanding of the complex factors that influence currency values.

For future research, we recommend exploring hybrid models that combine the strengths of more than one approach. Integrating time series analysis, sentiment analysis, and perhaps other advanced machine learning techniques could lead to a more comprehensive and robust forecasting model. In addition, following technological developments closely and constantly updating the model with the latest data will be of great importance in terms of adapting to the evolving nature of financial markets.

Additional Codes

```
import pandas as pd
import numpy as np
file_path = 'data.xlsx'
data = pd.read_excel(file_path)
data_cleaned = pd.read_excel(file_path,
skiprows=11)
data_cleaned = data_cleaned.drop(0)
non_numeric =
data_cleaned.select_dtypes(exclude=[np.nu
mber]).columns
nan_values = data_cleaned.isna().sum()
data_cleaned = data_cleaned.dropna()
for col in data_cleaned.columns[2:]:
    data_cleaned[col] =
pd.to_numeric(data_cleaned[col],
errors='coerce')
non_numeric_clean, nan_values_clean =
data_cleaned.select_dtypes(exclude=[np.nu
mber]).columns, data_cleaned.isna().sum()


features = ["USD (TP DK USD S YTL)", "EUR
(TP DK EUR S YTL)", "GBP (TP DK GBP S
YTL)",
           "SEK (TP DK SEK S YTL)", "CHF
(TP DK CHF S YTL)", "CAD (TP DK CAD S
YTL)",
           "KWD (TP DK KWD S YTL)"]
target =   ["SAR (TP DK SAR S YTL)
(Output)"]
X = data_cleaned[features]
y = data_cleaned[target]

class MultipleLinearRegression:
    def __init__(self):
        self.coefficients = None

    def fit(self, X, y):
        X = np.insert(X, 0, 1, axis=1)
        X_transpose = np.transpose(X)
        beta =
np.linalg.inv(X_transpose.dot(X)).dot(X_t
ranspose).dot(y)
        self.coefficients = beta

    def predict(self, X):
        X = np.insert(X, 0, 1, axis=1)
        return X.dot(self.coefficients)
model = MultipleLinearRegression()
model.fit(X.values, y.values)

results = pd.DataFrame({'Actual': y,
'Predicted': model.predict(X.values)})

from matplotlib import pyplot as plt
plt.figure(figsize=(12, 6))
plt.plot(results['Actual'][:100],
label='Actual', color='green',
marker='o')
plt.plot(results['Predicted'][:100],
label='Prediction', color='red',
linestyle='--', marker='x')
plt.title('Multiple Linear Regression
Predictions and Actual Values (First 100
Samples)')
plt.xlabel('Data Index')
plt.ylabel('SAR (TP DK SAR S YTL)')
plt.legend()
plt.show()

def mean_squared_error(y_true, y_pred):
    return np.mean((y_true - y_pred) **
2)

mse_all =
mean_squared_error(results['Actual'],
results['Predicted'])

def derivative_mse(y_true, y_pred, X):
    return -2 * np.dot(X.T, (y_true -
y_pred)) / len(y_true)

mse_derived =
derivative_mse(results['Actual'],
results['Predicted'], X.values)

def gradient_descent(X, y, learning_rate,
epochs):
    weights = np.random.rand(X.shape[1])
    mse_history = []
    for epoch in range(epochs):
        predictions = np.dot(X, weights)
        mse = mean_squared_error(y,
predictions)
        mse_history.append(mse)
        gradient = derivative_mse(y,
predictions, X)
        weights -= learning_rate *
gradient
    return weights, mse_history


class
MultipleLinearRegressionRegularized:
    def __init__(self, l1_ratio=0.5,
alpha=0.1, epochs=1000,
learning_rate=0.01):
        self.l1_ratio = l1_ratio
        self.alpha = alpha
        self.epochs = epochs
        self.learning_rate =
learning_rate
        self.coefficients = None

    def fit(self, X, y):
        X_extended =
np.column_stack((np.ones(len(X)), X))
        m, n = X_extended.shape
        self.coefficients = np.zeros(n)

        for _ in range(self.epochs):
            y_pred = 
X_extended.dot(self.coefficients)
            gradients = -2/m * 
X_extended.T.dot(y - y_pred)
            l1_penalty = self.l1_ratio * 
self.alpha * np.sign(self.coefficients)
            l2_penalty = (1 - 
self.l1_ratio) * self.alpha * 
self.coefficients
            gradients += l1_penalty + 
l2_penalty
            self.coefficients -= 
self.learning_rate * gradients

    def predict(self, X):
        X_extended = 
np.column_stack((np.ones(len(X)), X))
        return 
X_extended.dot(self.coefficients)

mlr_regularized = 
MultipleLinearRegressionRegularized()
mlr_regularized.fit(X.values, y.values)

predictions_regularized = 
mlr_regularized.predict(X.values[:5])


user_values = []

print("Please enter values for the 
following independent variables:")
independent_variables = ["USD", "EUR", 
"GBP", "SEK", "CHF", "CAD", "KWD"]

for variable in independent_variables:
    user_input_value = 
float(input(f"{variable} Value: "))
    user_values.append(user_input_value)

user_values_array = 
np.array([user_values])

predicted_output = 
model.predict(user_values_array)
print(f"Predicted Output: 
{predicted_output[0]}")
```