Machine Learning
Gold Prices Project

Prof.Abdel-Rahman Hedar
Date: 20 May 2024

Heba Mostafa Ahmed
Mariam Hamed Abdel Wahab
Waad Ahmed Shabaan
Asmaa Mohamed Bayoumi
Rehab Ashraf Abd Elwadood

Prepared by:

Table of Contents

I.Introduction

1. Introduce the Project Problem

In dynamic financial markets, accurately predicting the price of gold is of significant importance to investors, traders, and financial analysts. Gold is often seen as a safe-haven asset, and its price is influenced by various factors such as inflation rates, currency exchange rates, and the prices of other commodities. The ability to predict gold prices can provide a competitive edge, aiding in investment decisions and risk management. This project aims to utilize machine learning techniques, specifically regression models, to predict future gold prices based on historical data and other influencing factors.

2. Formal Definition and Importance Formal Definition

Machine Learning: Machine learning is a branch of artificial intelligence that involves the development of algorithms and statistical models that enable computers to learn from and make predictions based on data. Supervised learning, a subset of machine learning, involves training a model on labeled data, where the correct output is known, to predict outcomes for new data.

Supervised Learning: Supervised learning is a type of machine learning where the model is trained on input-output pairs, learning to map inputs to outputs to predict the label of unseen data. It encompasses two main types: classification (predicting categorical outcomes) and regression (predicting continuous outcomes).

Regression: Regression is a supervised learning technique used for predicting continuous numerical values. The objective is to model the relationship between a dependent variable (target) and one or more independent variables (predictors). Common regression algorithms include linear regression, polynomial regression, decision tree regression, support vector regression (SVR), and random forest regression.

Random Forest Algorithm: The Random Forest algorithm is an ensemble learning method used for both classification and regression tasks. It operates by constructing multiple decision trees during training and averaging their predictions to improve accuracy and control overfitting. Random Forest is known f
robustness and ability to handle high-dimensional data effectively.

-Importance

The importance of applying machine learning, and specifically supervised learning with regression techniques, lies in its ability to provide accurate and data-driven predictions, which are crucial for making informed decisions in various fields

.

1. Predictive Power: Machine learning models can identify complex patterns in data and make accurate predictions, which is essential for tasks like financial forecasting, medical diagnosis, and marketing analytics.

2: Efficiency and Automation: These models can process and analyze large datasets more efficiently than manual methods, automating the prediction process and saving time and resources.

3-Data-Driven Decision Making: By leveraging historical data, machine learning models help organizations make informed decisions, reducing uncertainty and improving outcomes.

4: Adaptability: Models like Random Forest can handle diverse types of data and are robust to overfitting, making them versatile tools for predictive modeling.

3. Analyze the Considered Problem Solvers and Algorithms

Several machine-learning algorithms can be applied to the problem of predicting gold prices. These include:
-Linear Regression: A simple approach that models the relationship between the dependent and independent variables as a linear function. While it is easy to interpret, it may not capture complex patterns in the data.
-Polynomial Regression: An extension of linear regression that models nonlinear relationships by introducing polynomial terms of the independent variables.
-Support Vector Regression (SVR): Uses the principles of support vector machines for regression tasks, offering robustness against outliers and effective handling of high-dimensional data.
-Decision Tree Regression: Builds a tree-like model of decisions, making predictions based on the splits in the tree. It is intuitive and easy to visualize, but it is prone to overfitting.

Nearest Neighbors Regression (KNN): Makes predictions based on the average of the nearest data points. It is simple but can be computationally expensive and sensitive to the choice of 'k'.

Random Forest Regression: An ensemble method that constructs multiple decision trees and averages their predictions. It reduces overfitting and improves accuracy by combining the strengths of individual trees

In this project, we applied both the K-Nearest Neighbors (KNN) and Random Forest algorithms to predict gold prices. The Random Forest algorithm was chosen as the primary model due to its superior performance in terms of the R-squared score and accuracy. The Random Forest algorithm excels in handling complex datasets with multiple features and provides robust predictions by averaging the outputs of multiple trees, which mitigates the risk of overfitting and enhances the model's generalization capability.
By employing the Random Forest algorithm, we aim to develop a reliable predictive model that can assist investors and analysts in making data-driven decisions, ultimately contributing to more effective financial planning and risk management. The better R-squared score and higher accuracy of the Random Forest model underscore its suitability for predicting gold prices in this context.

II.Methodology

2.1 Brief about the algorithm and its steps

A simple yet crisp definition to understand what a Random Forest Regression Algorithm is will be, "Random Forest Regression is a supervised learning algorithm that uses ensemble learning methods for regression. It operates by constructing several decision trees during training time and outputting the mean of the classes as the prediction of all the trees."

How does the Random Forest Algorithm operate?

Select a random sample of the data from the dataset. This is known as a bootstrap sample, and it is used to train a single decision tree.

Randomly select a subset of features from the dataset. The number of features to select is a hyperparameter that can be set by the user.

Use the selected features to train the decision tree on the bootstrap sample.

Repeat steps 1–3 multiple times to create a collection of decision trees, each trained on a different bootstrap sample and subset of features.

To predict a new data point, pass the data point through all the decision trees in the collection and obtain a prediction from each tree.

6. For regression tasks, use averaging to obtain the final prediction.

2.2 The main steps for the random forest algorithm

1-Data Preparation:

Split the dataset into a training set and a test set.

2: Building the Forest:

Bootstrap Sampling: Randomly select samples from the training set with replacements to create multiple subsets (bootstrap samples).

-Tree Construction:
For each bootstrap sample:
Grow an unpruned decision tree.
At each node, select a random subset of features.
Find the best split among these features based on some criterion (e.g., Gini impurity for classification or mean squared error for regression).
Split the node into child nodes based on the best split.
Repeat the process until the maximum depth is reached or the node is pure.
3-Making Predictions:

For a given test sample, pass it through each of the trees in the forest to get individual predictions.
Aggregation:
For classification, use the majority vote from all the trees.
For regression, use the average of all the tree predictions.
Final Output:

The final output is the aggregated result from all the individual trees.

2. Flowchart:

Pseudocode implementation of the random forest:

```
function RandomForest(train_data, num_trees, max_features):
    forest = []

    for i = 1 to num_trees:
        # Step: Select partial samples randomly (with put-back)
        bootstrap_sample = bootstrap(train_data)

        # Step: Select partial features as feature selection randomly
        feature_subset = random_subset(features, max_features)

        # Build the decision tree
        tree = BuildTree(bootstrap_sample, feature_subset)

        # Step: Storage decision tree
        forest.append(tree)

        Check if the number of decision trees meets the requirement
        if len(forest) >= num_trees:
            break

    return forest

function BuildTree(data, feature_subset):
    if can_become_leaf_node(data):
        return LeafNode(data)
    elif stopping_criteria_met(data):
        return LeafNode(data)
    else:
        best_split = find_best_split(data, feature_subset)
        left_data, right_data = split(data, best_split)

        # Perform branching
        left_tree = BuildTree(left_data, feature_subset)
        right_tree = BuildTree(right_data, feature_subset)

        return DecisionNode(best_split, left_tree, right_tree)

function find_best_split(data, features):
    best_feature = None
    best_threshold = None
    best_impurity = infinity
```

```
for a feature in features:
        for threshold in unique_values(data[feature]):
            impurity = calculate_impurity(data, feature, threshold)

            if impurity < best_impurity:
                best_feature = feature
                best_threshold = threshold
                best_impurity = impurity

    return best_feature, best_threshold

function predict(forest, sample):
    predictions = [predict_tree(tree, sample) for tree in forest]
    return aggregate_predictions(predictions)

function predict_tree(tree, sample):
    If the tree is LeafNode:
        return tree.prediction
    if sample[tree.split.feature] <= tree.split.threshold:
        return predict_tree(tree.left, sample)
    else:
        return predict_tree(tree.right, sample)

function aggregate_predictions(predictions):
    if classification_task:
        return mode(predictions)
    else:
        return mean(predictions)

# Auxiliary functions
function bootstrap(data):
    # Select samples randomly with replacement
    return [random.choice(data) for _ in range(len(data))]

function random_subset(features, max_features):
    Select a random subset of features
    and return random.sample(features, max_features)

function unique_values(column):
    # Return unique values in the column
    return set(column)

function calculate_impurity(data, feature, threshold):
    # Calculate impurity (e.g., Gini impurity, mean squared error)
    # based on the feature and threshold
    pass

function can_become_leaf_node(data):
    Check if data can be considered a leaf node based on stopping criteria
    pass

function stopping_criteria_met(data):
    Check if stopping criteria are met for tree growth
Pass
```

2.3 The solution to our problem using random forest Algorithm

Problem Description:

The problem at hand is to predict the future prices of gold using historical gold price data. Predicting gold prices accurately is crucial for investors, financial analysts, and policymakers, as it aids in making informed decisions regarding investments and economic policies. Given the volatile nature of gold prices, which are influenced by various factors such as market demand, geopolitical events, and economic indicators, a robust predictive model is essential.

Solution Overview:

The solution involves utilizing a Random Forest Regressor, a machine-learning algorithm known for its high accuracy and ability to handle complex datasets. The workflow diagram outlines the steps involved in the process of building this predictive model.

description of each step:

1. Gold Price Data Collection:
   Gather historical gold price data from reliable sources. This data includes daily, weekly, or monthly gold prices over a significant period.

2. Data Preprocessing:
   Clean the data to handle missing values, outliers, and noise. Transform the data into a suitable format for analysis.
Normalize or standardize the data if required to ensure that all features contribute equally to the model.

3. Data Analysis:
Perform exploratory data analysis (EDA) to understand the underlying patterns and trends in the data.
Visualize the data to identify any seasonal patterns, correlations with other financial indicators, and potential influencing factors.

4. Train-Test Split:
Split the data into training and testing sets. Typically, a common split is 80% for training and 20% for testing. Ensure that the split maintains the temporal order of the data to avoid data leakage.

5. Random Forest Regressor:
Train the Random Forest Regressor on the training dataset. This involves creating multiple decision trees and combining their predictions to improve accuracy and reduce overfitting.
Optimize the hyperparameters of the Random Forest model using techniques such as grid search or random search.

6. Evaluation:
Evaluate the model's performance on the test dataset using metrics such as mean absolute error (MAE), mean squared error (MSE), and R-squared.
Compare the model's predictions with actual gold prices to assess its accuracy.

III. Experimental Simulation

3.1 The Used Environment and Programming Language

We employed the Random Forest algorithm in Python, utilizing key libraries and tools: NumPy for efficient array operations, Pandas for data manipulation via DataFrames, and Matplotlib. plot alongside Seaborn for diverse data visualization. Additionally, Scikit-learn, a machine learning library, facilitated model construction, with the RandomForestClassifier being specifically employed for our implementation.

So, the process of solving the problem was divided into the following main steps:

Importing the necessary libraries.

Loading the required data.

Analyzing the data.

Building and evaluating the model.

3.2 The primary function & Explain the test cases

This is a Python code that imports several modules: numpy, pandas, matplotlib, pyplot, seaborn, and several from sklearn. Without more context or code, it's difficult to say exactly what this code does. The numpy module is a popular library for numerical computing in Python. The Panda's module is used for data manipulation and analysis. The matplotlib, pyplot, and seaborn modules are used for data visualization.

The sklearn modules are used for machine learning tasks; train_test_split is for splitting datasets training and testing sets.MinMaxScaler is for scaling features to a specified range.RandomForestRegressor, SVR, DecisionTreeRegressor, and KNeighborsRegressor are different regression models. mean_absolute_error, mean_squared_error, r2_score, and confusion_matrix are metrics for evaluating model performance. cross_val_score and KFold are used for cross-validation.

This code may define functions or classes that use these modules, or it may simply be importing them for use in other code that is not shown here.

This script illustrates how to import a dataset from a CSV file using the `pandas` library in Python. It begins by employing the `pd.read_csv()` function to read the CSV file "gld_price_data.csv" and load the data into a DataFrame named `gold_data`. Although this function already returns a DataFrame, the script explicitly creates another DataFrame `df` from `gold_data` using `pd.DataFrame(gold_data)`, which is somewhat redundant but underscores that `df` is intended for subsequent use.

The df.tail() function in Pandas is used to return the last few rows of a data frame. By default, it returns the last 5 rows.

This line of code retrieves the dimensions of a data frame named `df`, which holds a dataset. By accessing the `shape` attribute (`df. shape`), a tuple is returned containing the number of rows and columns in the DataFrame.

This line of code provides a quick summary of the DataFrame `df`, which contains a dataset. By calling the info() method on the DataFrame, `df.info()` returns essential information, including the number of entries (rows) and columns, the names of the columns, the data type of each column, the number of non-null values in each column, and the memory usage of the DataFrame. Executing this line offers a comprehensive overview of the dataset's structure and content, helping to identify missing values and understand the data types of each feature, which are crucial steps for effective data cleaning and preprocessing.

This line of code generates a statistical summary of the dataset contained within the DataFrame `gold_data`. By invoking the `describe()` method (`gold_data.describe()`), key statistical metrics are computed for each numerical column. These metrics typically include the count, mean, standard deviation, minimum value, 25th percentile (first quartile), median (50th percentile), 75th percentile (third quartile), and maximum value.

This line of code calculates the number of unique values present in each column of the DataFrame `df`. By utilizing the `nunique()` method on the DataFrame (`df. nunique()`), a series is generated containing the count of distinct values for each column. This summary aids in comprehending the diversity and uniqueness of data across different features.

This line of code computes and returns the count of missing values present in each column of the DataFrame `df`. Initially, the ISNA () method is applied to the DataFrame, generating a new DataFrame consisting of Boolean values indicating whether each element is missing (true) or not missing (false). Following this, the `sum()` method is employed to calculate the sum of these Boolean values along each column axis. The resulting series provides a succinct summary of the number of missing values for each column in the data frame.

This line of code generates a box plot for the DataFrame `df`, which contains data from columns `gold', 'sex', 'up`, `uso`, `slv`, and `eur/usd`. Box plots offer a visual representation of data distribution, aiding in the understanding of central tendency, variability, and the identification of potential outliers. This plot is created using the `boxplot()` method, specifying a plot size of 15×5 inches.

These lines of code generate individual box plots for four specific columns (`GLD`, `USO`, `SLV`, `EUR/USD`) in the DataFrame `df`. Each box plot is created using Seaborn's `boxplot()` function, which effectively visualizes the distribution of values within the respective column.

Before generating each box plot, the statement `plt.figure(figsize=(2, 2))` initializes a new figure with dimensions of 2 inches in width and 2 inches in height. This ensures that each box plot is presented compactly and clearly.

The Python function called `capping_outliers` is crafted to handle outliers within a specific column o DataFrame `df`. It initiates by computing the first quartile (Q1) and third quartile (Q3) of the designa column, which facilitates the calculation of the interquartile range (IQR). The IQR quantifies the spread of the middle 50% of the data. Utilizing the IQR, upper and lower limits are established to pinpoint outliers.

Outliers surpassing the upper limit are capped by substituting them with the upper limit value, while outliers falling below the lower limit are managed similarly. By invoking this function with the name of a column as an argument, outliers within that column are identified and adjusted based on the computed upper and lower limits. This systematic approach ensures that outliers are handled within acceptable boundaries, thereby enhancing the robustness and reliability of data analysis outcomes.

These code lines first handle outliers in each column of DataFrame `df` using the Python function 'capping_outliers`, then visualize their distribution via a box plot with Seaborn's `boxplot()` function. This combined approach ensures outlier management and provides a clear summary of data distribution for analysis.

These lines first extract numeric columns into a DataFrame `numeric_df` from `df`, then compute the correlation matrix `cor` using `corr()`, and visualize it as a heatmap with Seaborn's `heatmap()` function. This approach efficiently explores and illustrates relationships between numeric variables, facilitating data analysis and decision-making.

These lines print correlation values between the 'GLD' column and all other numeric columns in DataFrame `df`, providing insight into their linear relationships. Positive values indicate a positive correlation, negative values denote a negative correlation, and values near zero signify a weaker correlation.

This line of code generates histograms for each column in the DataFrame `df`, which includes data from columns `gold', 'sex', 'up`, `uso`, `slv`, and `eur/usd`. Histograms offer a visual summary of interval-scale data by illustrating frequency distributions within defined bins or intervals. By applying the `hist()` method to the entire DataFrame `df`, histograms are automatically created for each column. Setting the `bins` parameter to 10 provides detailed distribution insights by dividing the data into 10 bins. The `fig size` parameter ensures clear visualization by setting the plot dimensions to 15×10 inches, while `grid=F removes gridlines for a cleaner appearance.

This code line utilizes Seaborn's `distplot()` function to generate a histogram visualizing the distribution of 'GLD' prices in DataFrame `df`, with histogram bars colored purple. Executing this code provides insights into 'GLD' price distribution, including central tendency, spread, shape, and potential outliers or skewness.

These lines split the dataset into input features (X) and the target variable (Y) by dropping the 'Date' and 'GLD' columns from DataFrame `df` for X and assigning the 'GLD' column to Y. This ensures the model predicts 'GLD' prices based on other available features. Printing both X and Y provides a clear view of the data for analysis, aiding in understanding the dataset structure. and print X and Y.

These lines utilize Scikit-learn's `train_test_split` function to divide input features (X) and target variables (Y) into training and testing sets, with 20% allocated for testing. Setting `random_state=2` ensures reproducibility. Outputs include X_train and X_test for features and Y_train and Y_test for target variables. and print all of them.

These lines instantiate and train a RandomForestRegressor model using training data (X_train, Y_train), make predictions on test data (X_test), and print predicted 'GLD' prices. This facilitates quick evaluation of model performance and provides predicted values for further analysis or decision-making.

These lines utilize Scikit-learn's `metrics.r2_score` function to compute the R-squared error between true target variable values (Y_test) and predicted values (predictions) from the RandomForestRegressor model. The computed R-squared score (R_scoreRF) is then printed to evaluate the model's goodness of fit, providing insights into its ability to explain variability in 'GLD' prices based on input features.

The line Y_test = list(Y_test) in Python is used to convert Y_test into a list. This can be necessary or convenient in various situations

These lines plot the actual 'GLD' prices (Y_test) against their corresponding indices in the test data, with a blue color labeled 'Actual Value'. Axis labels are set, and a legend is displayed.

These lines plot the predicted 'GLD' prices (predictions) against their corresponding indices in the test data, with a green color labeled 'Predicted Value'. Axis labels are set, and a legend is displayed.

These lines create a plot comparing actual ('Y_test') and predicted ('predictions') 'GLD' prices. Actual values are plotted in blue and predicted in green with labels and title sets. Executing this code generates a visual comparison, aiding in assessing the model's accuracy and performance.

This Python code demonstrates the process of training and evaluating a random forest classifier using synthetic data. It begins by importing the necessary libraries: `RandomForestClassifier` from `sklearn. ensemble`, `cross_val_score` from `sklearn.model_selection`, and `numpy`. It then generates a dataset with 1832 samples, where `X_train_RF` contains 10 randomly generated features per sample and `Y_train_RF` contains randomly generated continuous labels. The continuous labels are converted into three discrete categories using `np. digitize` and bins created with `np. airspace. After initializing the RandomForestClassifier with a fixed random state for reproducibility, the model is trained on the feature matrix and the binned labels. The accuracy of the training data is computed using `rf. score`. To further evaluate the model, 5-fold cross-validation is performed using `cross_val_score`, and the mean cross-validation score is calculated. Finally, the code compares the training accuracy with the mean cross-validation accuracy to assess the risk of overfitting, where a higher training score than the cross-validation score indicates potential overfitting.

The code compares the training score with the mean cross-validation (CV) score to check for overfitting. If the training score is higher than the mean CV score, it prints a warning about the risk of overfitting. Otherwise, it indicates no significant risk of overfitting.

IV. Results and Technical Discussion

Report the main program results and outputs

INPUT is the Data set before data cleaning

statistical measures of the data

This returns the columns with the number of missing values

This returns the columns with outliers

This returns the columns without outliers

This returns the correlation matrix to the dataset

Histogram

This returns value training and testing to the dataset

Output: Random Forest Algorithm

Output: k-Nearest Neighbor Algorithm

4.2 Test/Evaluation Experimental Procedure and Analysis of Results Using Random Forest Algorithm

User Input for Initial State:

The user provides the initial dataset or the problem's starting configuration that the Random Forest algorithm will use for training and evaluation.

User Input of Goal State:

The user specifies the desired outcome or goal state for the algorithm. This could be a target clas classification, a value for regression, or another specified result.

Evaluation Procedure:

Training the Model:

If the number of trees (n) in the forest is set to 1, the Random Forest algorithm initializes with a single decision tree. The model is trained using the provided dataset. The fit method is called with the training data (features and target) to build the tree.

Evaluating the Model:

After training, the model is evaluated using a predefined evaluation function. This function calculates performance metrics such as accuracy, precision, recall, or other relevant measures based on the goal state. The evaluation metrics are stored for further analysis.

Best Solution Identification:

The best_solution function is called with the trained model and the evaluation metrics as arguments to identify the best-performing tree or configuration within the Random Forest. The best configuration, or path, is stored in the best_path variable.

Output Results:

The best_path variable, representing the optimal sequence of decisions or the best model configuration, is printed to the console. Any unnecessary characters (e.g., square brackets) are replaced with spaces using the replace method.

Calculation of Metrics:

Total Moves: The total number of decisions or steps required to reach the goal state is calculated by subtracting 1 from the length of the best path. This value is printed on the console.

Nodes Visited: The number of nodes (decisions or data points) visited during the training and evaluation process is calculated as the difference between the size of the dataset and the number of unique nodes visited, stored in the visited variable. This value is also printed on the console.

Nodes Generated: The total number of nodes generated during the training process (i.e., the number of trees created) is printed to the console, represented by the length of the state.

4.3 Discuss the Main Results and Their Quality

The main results of a Random Forest algorithm are typically measured in terms of the quality o solutions it produces. The quality of the results of a Random Forest algorithm depends on several factors, such as the accuracy and diversity of the individual trees, the method of aggregating these trees, and the characteristics of the dataset being used.

Accuracy of Individual Trees: The trees built in a Random Forest should be strong and provide accurate estimates of the target outcomes. This is achieved by using random samples of the data for each tree and selecting random features when building the tree. This diversity helps reduce bias and improve accuracy.

Aggregation of Trees: The results of the individual trees are aggregated through voting or averaging, which enhances the predictive power of the model. This method is particularly effective in reducing variance and improving the stability and accuracy of the final results.

Characteristics of the Data: The properties of the dataset used can significantly affect the quality of the results. Datasets with high noise or imbalance may require additional preprocessing, such as cleaning or balancing, to ensure that the model can accurately learn the true patterns.

Overall, a well-tuned Random Forest model should provide accurate estimates and produce high-quality results when applied to an appropriate dataset. The quality of these results can be measured using various performance metrics such as accuracy, predictive power, and effectiveness measures like F1-score or ROC-AUC, depending on the nature of the problem.

Conclusions

5.1 Conclusion remakes

The machine learning project aimed at predicting the price of gold using regression techniques has demonstrated the potential of leveraging historical data and various economic indicators to forecast future prices. Through the application of multiple regression models, including linear regression, polynomial regression, and more advanced techniques like support vector regression (SVR) and neural networks, several insights were garnered:

5.2 Recommendations for future work

· Future Work:

· Incorporating real-time data and enhancing the models with live market feeds could improve prediction accuracy and timeliness.

Exploring hybrid models that combine regression with time-series forecasting techniques like ARIMA or LSTM (Long Short-Term Memory) networks could yield better results.

Continuous model updates and retraining with new data will ensure the models remain relevant accurate over time.

References

https://en.m.wikipedia.org/wiki/Machine_learning

https://www.analyticsvidhya.com/blog/2021/07/building-a-gold-price-prediction-model-using-machine-learning/

https://www.researchgate.net/figure/Flow-chart-of-the-random-forest-regression-algorithm_fig3_360831728

https://utsavdesai26.medium.com/mastering-random-forest-algorithm-a-step-by-step-learning-guide-f7abf2420a55 https://colab.research.google.com/drive/1r4uFJJWoa4ze3R3X1I904RbUNgGKMnbW?usp=sharing

Appendix