Machine Learning Project

Eloi Goutay / Samuel Bontemps

December 2024

1 Stage 1

In this first stage of the project, we have implemented three standard machine learning models integrated with historical, economic, and technical data to make predictions of Brent crude oil prices. These predictions can be used to inform operational decisions, trading strategies, and financial planning.

1.1 Steps in Data Preparation

- Importing the necessary libraries.
- Collecting historical prices of Brent crude oil.
- Collecting and adjusting Economic Indicators data.

The Global GDP and inflation data retrieved from the World Bank API were available as annual values from 2010 to 2023. To integrate these indicators into the dataset, we first added forecasted values for 2024. Then, we resampled the data to a daily frequency, with missing values filled using linear interpolation. This approach ensured compatibility with the daily oil prices dataset and provided smooth transitions between annual data points.

- Linearly interpolating to generate daily data for GDP and inflation.
- Adding returns over the horizon of from 1 to 10 days.

We then calculated the oil price returns on different horizons as well as technical indicators. The idea behind this is to provide models with data that can help detect trends in past values, increasing the accuracy of predictions.

1.2 Technical Indicators

The following indicators were calculated:

- **SMA (Simple Moving Average):**

$$\text{SMA}_n = \frac{1}{n} \sum_{i=0}^{n-1} P_{t-i}$$

where n is the period (e.g., 20 or 50 days), and P_t is the price at time t.

- **RSI (Relative Strength Index):**

$$\text{RSI} = 100 - \frac{100}{1 + RS}$$

where $RS = \frac{\text{Average Gain over } n \text{ periods}}{\text{Average Loss over } n \text{ periods}}$, and n is typically set to 14.

- **MACD (Moving Average Convergence Divergence) and Signal Line:**

$$\text{MACD} = \text{EMA}_{12} - \text{EMA}_{26}$$

$$\text{Signal Line} = \text{EMA}_9(\text{MACD})$$

where EMA_n is the Exponential Moving Average over n periods.

- **Bollinger Bands:**

$$\text{Upper Band} = \text{SMA}_n + (k \times \sigma)$$

$$\text{Lower Band} = \text{SMA}_n - (k \times \sigma)$$

where σ is the standard deviation of the prices over n periods, and k is typically set to 2.

1.3 Pre-Processing

The calculation of rolling indicators, such as returns, moving averages (SMA20, SMA50), and technical indicators like RSI and Bollinger Bands, which require a certain number of prior data points, led to the appearance of missing values in the first rows of the dataset.

To handle these missing values effectively:

- Rows with more than 20% missing values were removed, ensuring a cleaner dataset for analysis.
- For remaining gaps (e.g., in the SMA50 column), missing values were filled with the column's mean to maintain consistency without distorting trends.

This approach effectively managed missing data while preserving the dataset's integrity for modeling.

1.4 Linear Regression

We first selected linear regression as a baseline model due to its simplicity, interpretability, and suitability with time series data. Because we are dealing with time series data, we split the data sequentially into training and testing sets instead of a random split. This method ensures a chronological separation to simulate real-world forecasting.

In order to monitor under-/over-fitting and maximize the model's performance, we evaluated different training sizes (50% to 90%) to determine the optimal data split for minimizing bias and variance, and we compared training and testing MSE.

- **Best Training Size:** 60%, mitigating overfitting while maintaining performance.
- **Metrics:**

$$\mathrm{R}^2 = 0.9987., \quad \mathrm{MSE} = 0.468$$

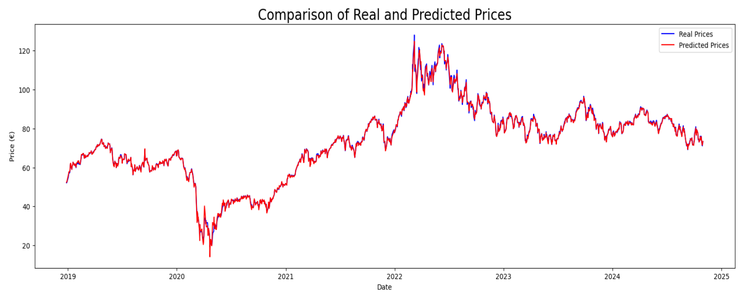

Figure 1: Real Prices and predicted prices with Linear Regression

The gap between training and testing MSE values (0.1526 vs. 0.468 at the optimal split) suggests that there may still be a little bit of overfitting. However, the model achieved excellent overall performance for this dataset.

1.5 Decision Tree Regressor

We then implemented the Decision Tree Regressor, which models non-linear relationships. However, decision trees are prone to overfitting, as reflected in the results.

- **Training Size:** 90%, minimizing bias and variance.
- **Metrics:**

$$\mathrm{R}^2 = 0.86, \quad \mathrm{MSE} = 4.29$$

To address overfitting, we experimented with regularization (e.g., setting a maximum depth). Although this initially reduced overfitting, the model's test performance did not improve, and the original unregularized model was retained.

1.6 K-Cross Validation

To reduce overfitting, we applied K-cross validation to the Decision Tree Regressor:

- The dataset was split into k subsets (folds).
- For each iteration, one fold was used as the test set, while the remaining $k-1$ folds served as the training set.

Best Number of Folds: $k = 5$.

- **Metrics (across 5 folds):**

$$\mathrm{R}^2 = 0.9234, \quad \mathrm{MSE} = 14.26$$

This analysis confirmed improved generalization with k-cross validation, though overfitting tendencies persisted.

1.7 Conclusion

After implementing and evaluating three models : Linear Regression, Decision Tree Regressor, and Decision Tree with K-Cross Validation; we observed significant differences in their performance.

- The Linear Regression model provided the best R2 score and low error metrics, though it might struggle with abrupt shocks or regime switches.
- The Decision Tree Regressor demonstrated overfitting but was improved with K-cross validation.
- The Decision Tree with K-cross validation balanced training size and test diversity but retained overfitting tendencies.

From a business perspective, more advanced models are required to improve accuracy and reliability in forecasting Brent oil prices.

2 Stage 2

2.1 KNN model

In Stage 2, we decided to implement the k-nearest neighbors (KNN) algorithm on our dataset to enhance our predictions. After conducting several manual tests, classified based on different metrics, we chose a training split size of 0.6. We used the "GridSearchCV" function with the KNeighborsRegressor model to optimize our parameters for the k-nearest neighbors algorithm. The first parameter we optimized was the "metric," which determines how the distance between a test sample and all training samples is calculated. The second parameter was k, which represents the number of nearest neighbors the model will consider (i.e., the k smallest distances from the test sample among all training samples). We explored a wide range of k-values, estimating that the optimal value of k would likely fall between 1 and 50.

$$d(x, x_i) = \|x - x_i\|_p$$

where $\|\cdot\|_p$ represents the p-norm of the vector, and p is the distance metric (e.g., $p = 1$ for Manhattan distance or $p = 2$ for Euclidean distance).

For prediction, the KNN model computes the average of the k nearest neighbors' labels:

$$\hat{y}(x) = \frac{1}{k}\sum_{i=1}^{k} y_i$$

The GridSearchCV function ranks the results based on the R^2 score metric. In other words, the output of the function consists of the parameter combinations that yield the best R^2 score. For our optimized model, we applied the "Manhattan" distance metric and set $k = 6$, as shown in the code below.

We evaluated the performance of our optimized model using the following metrics after predicting the test samples :

$$\mathrm{R}^2 = 0.8922, \quad \mathrm{MSE} = 39.9717$$

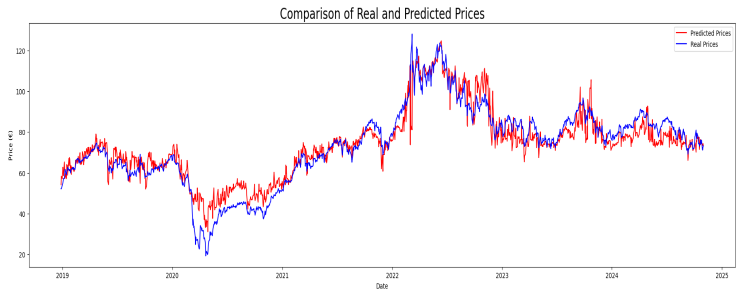

Figure 2: Real Prices and predicted prices with KNN model

The graph above compares the predicted values (in red) with the actual price values. Overall, we observe that the prediction curve closely follows the trend of the actual prices, suggesting relatively low variance. However, there are instances where the prediction curve deviates slightly from the actual price curve, indicating some bias. This bias is also reflected in our evaluation metrics.

2.2 Ensemble Learning

After testing a few simple models, we decided to combine them to amplify their strengths and minimize their weaknesses. This approach is called *Ensemble Learning*.

2.2.1 Bagging

We first combined our k-nearest neighbors model with our decision tree regressor. For the Bagging technique, we averaged the predictions from each model. Specifically, we predicted prices by averaging the predictions from the k-nearest neighbors and the decision tree. Mathematically, the bagging ensemble prediction is computed as:

$$\hat{y}_{\text{bagging}} = \frac{1}{N} \sum_{i=1}^{N} \hat{y}_i$$

where $\hat{y}_i$ is the prediction from each base model (KNN or decision tree) and N is the number of base models.

Initially, we did not use the same test size for each model due to metric optimization. To ensure comparability, we reduced the larger test size (k-nearest neighbors: 0.4) to match the smaller one (decision tree: 0.1).

$$\text{R}^2 = 0.751, \quad \text{MSE} = 7.6865$$

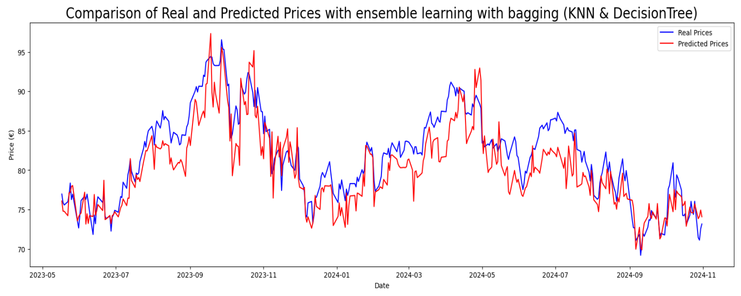

Figure 3: Real Prices and predicted prices with Bagging model

The graph above shows the predicted values (in red) compared to the actual prices (in blue). We observe a noticeable improvement compared to the individual models. Firstly, the prediction curve follows the trend of the actual prices much more accurately than the decision tree regressor's prediction. Additionally, we noticed some bias in the k-nearest neighbors model. By combining it with the decision tree regressor, the bias in the predictions was reduced. However, some instances still show bias, although it is improved compared to both individual models.

2.2.2 Stacking

We then decided to combine our k-nearest neighbors model and our decision tree regressor with a static technique. The goal of this method is to improve predictive performance by leveraging the strengths of different types of models. To achieve this, the predictions from the base models are used as inputs for a meta-model (a second-level model, in our case we chose a linear regression model) that learns to make the final prediction.

For the stacking ensemble, the final prediction is a weighted sum of the predictions from the base models:

$$\hat{y}_{\text{stacking}} = \sum_{i=1}^{N} w_i \hat{y}_i$$

where w_i are the weights learned by the meta-model (linear regression) and $\hat{y}_i$ are the predictions from each base model.

We optimized the stacking model's performance by combining overfitting control and hyperparameter optimization. We used a grid search function to find the best hyperparameters for each base model, and we split the dataset into different training sizes ranging from 50% to 90% of the data to find the optimal balance between R^2 score and MSE.

$$\text{R}^2 = 0.9453, \quad \text{MSE} = 17.556$$

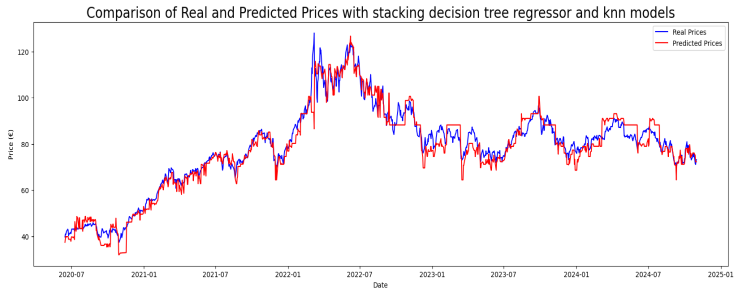

Figure 4: Real Prices and predicted prices with Stacking model

As seen from the plot and the error metrics, the stacking method significantly outperforms the bagging method. Additionally, it greatly enhances the performance of the individual base models (decision tree and KNN).

This strong accuracy achieved by the stacking method can be attributed to its underlying principle. As we saw before, the decision tree model produces low prediction errors but struggles to capture the variance in the data. On the other hand, the KNN model provides a better explanation of the data's variance but has higher prediction errors. By combining these models and weighting their values with another model, we can leverage their strengths while mitigating their weaknesses.

Modèle	R^2	**MSE**
KNN	0.8922	39.971
Decision Tree	0.8564	4.4320
Bagging	0.7510	7.6865
Stacking	0.9849	4.8585

Table 1: Comparaison of metrics for different models.

2.3 Comparaison of the models

3 Stage 3

In the final stage, we delved deeper into our project by implementing more sophisticated algorithms to enhance our results. We first employed an ensemble learning method known as XGBoost. Following this, we incorporated a deep learning approach through the GA-SVR-GRNN hybrid model to further improve the predictions.

3.1 XGBoost

We began by splitting the dataset into the explanatory variables X and the target variable Y, with 70% of the data used for training. We then optimized the model by adjusting several hyperparameters, including the depth of the trees, learning rate, number of estimators, and sample fractions. Instead of using GridSearch, we opted for RandomizedSearch to accelerate the search for the best parameters.

After fitting the model and making predictions, we evaluated its performance using standard metrics:

$$\mathrm{R}^2 = 0.9848, \quad \mathrm{MSE} = 4.8585$$

The XGBoost model showed strong results, with the predictive curve closely following the overall trend, though it encountered some difficulty in tracking larger fluctuations. However, the model displayed minimal bias, with predictions typically being quite accurate.

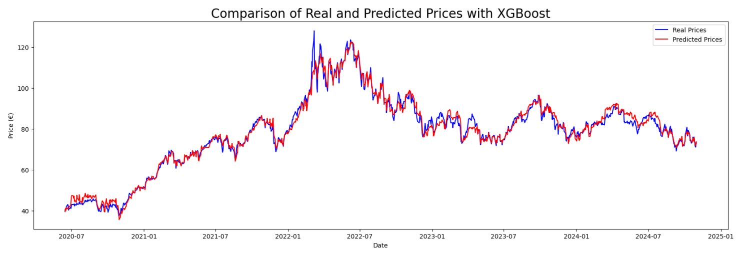

Figure 5: Real Prices and predicted prices with XGBoost

3.2 GA-SVR-GRNN

In the final part of the project, we implemented a GA-SVR-GRNN hybrid model, which combines three distinct techniques: Genetic Algorithms (GA), Support Vector Regression (SVR), and General Regression Neural Networks (GRNN). This model focuses on hyperparameter optimization, residual correction, and utilizes deep learning to enhance prediction accuracy.

The dataset was split in the same manner as before, with 70% used for training. The GA optimized the hyperparameters of the SVR model by minimizing the Mean Squared Error (MSE). We fine-tuned the penalty parameter C, the loss tolerance ϵ, and the kernel parameter γ, ensuring that all parameters were positive as required by the SVR model.

Best SVR parameters: $C = 50.6246$, $\epsilon = 2.6003$, $\gamma = 2.1459$

Once the SVR parameters were optimized, the model was trained and tested. Residuals, or errors in the initial predictions, were then modeled and corrected using a GRNN. The GRNN was trained to predict these residuals, with normalized input features improving its performance. The final prediction was made by combining the GRNN corrections with the original SVR predictions.

After adjusting the residuals, the model showed significantly improved performance:

$$\mathrm{R}^2 = 0.9998, \quad \mathrm{MSE} = 0.0797$$

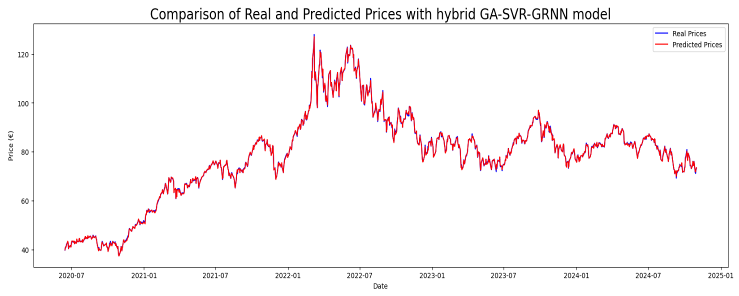

Figure 6: Real Prices and predicted prices with GA-SVR-GNN

The results, both visually and numerically, demonstrated that the GA-SVR-GRNN hybrid model significantly outperformed the previous models, showcasing the advantage of combining traditional machine learning techniques with deep learning. The MSE and R^2 scores indicate a substantial improvement in predictive accuracy.

The GA-SVR-GRNN model outperformed all other models, including the classic linear regression, in terms of both MSE and R^2, confirming its superior performance.