

## **# Stock Price Prediction using Various Machine Learning Models**

### **## Project Overview**

This project aims to predict stock prices using historical data of S&P 500 stocks. Multiple machine learning models including Support Vector Regression (SVR), Random Forest (RF), Long Short-Term Memory (LSTM), LightGBM, CatBoost, and XGBoost are trained and evaluated based on their performance.

For running the StockWebsite all you need is to have streamlit installed.

Then you can run:

```
```
```

```
streamlit run stock_analyzer.py
```

```
```
```

This will make the website run.

### **## Directory Structure**

#### **### Stocks Analyzer Main/Backend**

\* `data/` : Contains raw and processed data.

- `raw/` : Contains raw data files.

- `all\_stocks\_5yr.csv`

- `processed/` : Contains processed data files.

- `features_data.csv`
- `processed_data.csv`
- `code/` : Contains scripts for data preprocessing, feature engineering, model training, and evaluation.

- `train_svr.py`
- `train_rf.py`
- `train_lstm.py`
- `train_lightgbm.py`
- `train_catboost.py`
- `train_xgboost.py`
- `evaluation.py`
- `data_preprocessing.py`
- `ensemble.py`
- `feature_engineering.py`
- `models/` : Contains trained models in `.joblib` format.

- `svr_model.joblib`
- `rf_model.joblib`
- `lstm_model.h5`
- `lightgbm_model.joblib`
- `catboost_model.joblib`
- `xgboost_model.joblib`
- `scripts/` : Contains shell scripts to run the pipeline.

- `run_ensemble.sh`
- `run_evaluation.sh`
- `run_preprocessing.sh`
- `run_training_lstm.sh`

- `run\_training\_rf.sh`
- `run\_training\_svr.sh`
- `results/` : Contains predictions and evaluation metrics (optional).

### **### StockWebsite/Frontend**

- csv/: Contains the necessary csv to display data in website.

- processed\_data\_with\_predictions.csv
- media/: Contains the necessary media to display in website

- candlestick.jpg
- catboost.png
- lgbm.png
- lstm.mp4
- random\_forest.mp4
- stocks.jpeg
- svm.mp4
- traintest.png
- xgboost.png
- stock\_analyzer.py: This the program to run the website itself.
- `README.md` : Project documentation.
- `requirements.txt` : Required packages.

## **## Setup and Usage**

### **### Step 1: Install Required Packages**

Make sure you have Python 3.7 or later. Install the required packages using pip:

```
` `` sh
```

```
pip install -r requirements.txt
```

```
```
```

### **### Step 2: Run Data Preprocessing**

```
```sh
```

```
`python code/data_preprocessing.py`
```

```
```
```

### **### Step 3: Run Feature Engineering**

```
```sh
```

```
python code/feature_engineering.py
```

```
```
```

### **### Step 4: Train the Models**

```
```sh
```

```
`python code/train_svr.py
```

```
python code/train_rf.py
```

```
python code/train_lstm.py
```

```
python code/train_lightgbm.py
```

```
python code/train_catboost.py
```

```
python code/train_xgboost.py`
```

```
```
```

### **### Step 5: Evaluate the Models**

```
```sh
```

```
`python code/evaluation.py`
```

```
` `` `
```

## **## Model Explanation**

### **### Support Vector Regression (SVR)**

Support Vector Regression (SVR) is a type of Support Vector Machine (SVM) that is used for regression tasks. The main idea of SVR is to find a function that approximates the mapping from the input space to the output space by minimizing the prediction error. SVR uses a linear model in a high-dimensional space created by a kernel function, allowing it to handle non-linear relationships in the data. The key parameters of SVR include the kernel type (e.g., linear, polynomial, RBF), the regularization parameter (C), and the epsilon parameter which defines a margin of tolerance where no penalty is given to errors.

#### **#### Summary of SVR Data**

\* **Training Set** : Initially 80% of the data, further reduced to 500 samples for quicker training.

\* **Testing Set** : 20% of the data.

This means that the SVR model is trained on a small, randomly sampled subset of the original dataset (500 samples), and it is evaluated on 20% of the original dataset. This sampling is done to speed up the training process, but it might affect the model's performance depending on the dataset size and characteristics.

### **### Random Forest (RF)**

Random Forest is an ensemble learning method that builds multiple decision trees and merges them together to get a more accurate and stable prediction. Each tree in the forest is trained on a bootstrap sample from the training data, and during the construction of the tree, a random subset of features is considered for splitting at each node. This randomness helps to reduce the variance of the model and prevent overfitting. The final prediction of the Random Forest is obtained by averaging the predictions of all individual trees (for regression) or by majority voting (for classification).

### ### Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) is a type of Recurrent Neural Network (RNN) that is designed to model temporal sequences and long-range dependencies. Unlike standard RNNs, LSTMs can effectively capture long-term dependencies in the data without suffering from the vanishing gradient problem. An LSTM network consists of a series of LSTM cells, each containing three gates: the input gate, the forget gate, and the output gate. These gates control the flow of information into, out of, and through the cell, allowing the network to maintain a memory of past inputs over long sequences. LSTMs are particularly well-suited for time series forecasting tasks.

### ### LightGBM

LightGBM (Light Gradient Boosting Machine) is a gradient boosting framework that uses tree-based learning algorithms. It is designed to be highly efficient and scalable, making it suitable for handling large datasets with high-dimensional features. LightGBM introduces several optimizations, such as histogram-based learning, leaf-wise tree growth, and exclusive feature bundling, which significantly speed up training and reduce memory usage. The key hyperparameters of LightGBM include the number of boosting iterations, learning rate, number of leaves, and the objective function.

### ### CatBoost

CatBoost (Categorical Boosting) is a gradient boosting library that is specifically designed to handle categorical features without the need for extensive preprocessing. CatBoost uses an ordered boosting approach, which helps to reduce overfitting and improve the generalization ability of the model. It also incorporates techniques such as target-based categorical feature encoding and Bayesian bootstrapping. CatBoost is known for its ease of use, high accuracy, and fast training speed. The key hyperparameters of CatBoost include the number of boosting iterations, learning rate, depth of the trees, and the loss function.

### ### XGBoost

XGBoost (eXtreme Gradient Boosting) is an optimized implementation of gradient boosting that is designed to be highly efficient, flexible, and portable. XGBoost introduces several enhancements over traditional gradient boosting, such as regularization techniques to prevent overfitting, parallel and distributed computing capabilities, and the use of sparsity-aware algorithms. It supports

various objective functions, custom evaluation metrics, and early stopping to improve performance. The key hyperparameters of XGBoost include the number of boosting rounds, learning rate, maximum tree depth, and the objective function.

## **## Model Performance Evaluation**

The performance of each machine learning model is evaluated using the Mean Squared Error (MSE) metric. Below are the explanations for the MSE values of each model:

### **### SVR MSE: 1.555318016401746**

The SVR (Support Vector Regression) model has an MSE of approximately 1.56%. This means that, on average, the SVR model's predictions are off by about 1.56% from the actual stock prices. For instance, if the actual stock price is \$100, the SVR model's prediction would typically be within the range of \$98.44 to \$101.56. This indicates that the SVR model's predictions are the closest to the true values among all the models evaluated.

### **### RandomForest MSE: 3.806296152736704**

The RandomForest model has an MSE of approximately 3.81%. This means that, on average, the RandomForest model's predictions are off by about 3.81% from the actual stock prices. For example, if the actual stock price is \$100, the RandomForest model's prediction would typically be within the range of \$96.19 to \$103.81. This indicates that the RandomForest model's predictions are less accurate compared to the SVR model and have larger deviations from the true values on average.

### **### LSTM MSE: 1653137682715647.8**

The LSTM (Long Short-Term Memory) model has an extremely high MSE, indicating that its predictions are significantly off from the true values. This poor performance suggests that the LSTM model did not capture the underlying patterns in the stock price data effectively.

### **### XGBoost MSE: 272.8561486297449**

The XGBoost model has an MSE of approximately 272.86%. This means that, on average, the XGBoost model's predictions are off by about 272.86% from the actual stock prices. For instance, if the actual stock price is \$100, the XGBoost model's prediction would typically be within the range of \$-172.86 to \$372.86. This high percentage indicates that the XGBoost model's predictions are not very close to the true values. While it performs better than the LSTM model, it is still far from the performance of the SVR and RandomForest models.

### **### LightGBM MSE: 340.29349361064504**

The LightGBM model has an MSE of approximately 340.29%. This means that, on average, the LightGBM model's predictions are off by about 340.29% from the actual stock prices. For example, if the actual stock price is \$100, the LightGBM model's prediction would typically be within the range of \$-240.29 to \$440.29. This high MSE indicates poor performance in predicting stock prices, with large deviations from the true values on average.

### **### CatBoost MSE: 268.12847603201175**

The CatBoost model has an MSE of approximately 268.13%. This means that, on average, the CatBoost model's predictions are off by about 268.13% from the actual stock prices. For instance, if the actual stock price is \$100, the CatBoost model's prediction would typically be within the range of \$-168.13 to \$368.13. This high MSE indicates that the CatBoost model's predictions are also not very close to the true values. However, it performs slightly better than the XGBoost and LightGBM models.

## **## Why does SVR work so well?**

The SVR (Support Vector Regression) model works well for several reasons, especially in the context of predicting stock prices using historical data. Here are the key factors contributing to its effectiveness:

### **### 1. \*\*Ability to Handle Non-Linear Relationships\*\***

SVR can handle both linear and non-linear relationships in the data:

\* **Kernel Trick** : By using different kernel functions (e.g., linear, RBF), SVR can map the input features into higher-dimensional spaces, enabling it to capture complex, non-linear relationships. This is particularly useful in stock price prediction where the relationships between features and target can be non-linear.

### ### 2. **Regularization**

SVR includes a regularization parameter `C` that helps to balance the trade-off between achieving a low training error and minimizing the model complexity:

\* **C Parameter** : A higher value of `C` aims to fit the training data as well as possible, while a lower value of `C` will create a simpler model that might not fit the training data perfectly but generalizes better to new data.

### ### 3. **Margin of Tolerance (Epsilon) Parameter**

SVR introduces an epsilon parameter (`epsilon`) that specifies a margin of tolerance where no penalty is given for errors. This helps to make the model robust to noise in the data:

\* **Epsilon-Insensitive Loss** : This loss function ignores errors within a specified epsilon margin, making the model less sensitive to outliers and noise, which are common in stock price data.

### ### 4. **Feature Scaling**

The use of `StandardScaler` in the pipeline ensures that all features are scaled to have zero mean and unit variance:

\* **Standardization** : This step is crucial for SVR as it helps the algorithm to converge faster and prevents features with larger ranges from dominating the model training process.

### ### 5. **Hyperparameter Optimization**

The use of `GridSearchCV` allows for an exhaustive search over the specified hyperparameter grid to find the best combination of parameters:

\* **Cross-Validation** : Using cross-validation helps to ensure that the model performs well on unseen data and prevents overfitting.

\* **Hyperparameter Tuning** : By testing different values of `kernel`, `C`, and `gamma`, the model is fine-tuned to achieve the best performance.

### 6. **Robustness to Overfitting**

SVR is generally robust to overfitting due to its regularization capabilities and the use of cross-validation:

\* **Regularization** : The regularization parameter (`C`) helps to prevent the model from overfitting to the training data by penalizing large coefficients.

\* **Cross-Validation** : By validating the model on different subsets of the data, `GridSearchCV` helps to ensure that the selected model generalizes well to new data.

### 7. **Handling of Multicollinearity**

SVR can handle multicollinearity (high correlation between features) effectively:

\* **Kernel Methods** : The kernel methods used in SVR can manage multicollinearity by projecting the data into higher-dimensional spaces where the relationships between features might become more linear and less correlated.

### 8. **Effectiveness in Time Series Data**

Although SVR is not specifically designed for time series data, it can be effective in this context due to its ability to model complex relationships:

\* **Feature Engineering** : By incorporating features such as rolling mean and rolling standard deviation, the model can capture temporal patterns and trends in the stock prices.

### ### Summary

SVR's effectiveness in stock price prediction stems from its ability to handle non-linear relationships, regularization to prevent overfitting, robust performance through hyperparameter tuning, and the inclusion of feature scaling. These factors, combined with its capability to handle noise and outliers, make SVR a powerful tool for this type of regression task.

## ## Why does RF work so well?

The Random Forest (RF) model works very well for stock price prediction due to several inherent advantages and properties of the Random Forest algorithm. Here are the key factors contributing to its effectiveness:

### ### 1. Ensemble Learning

Random Forest is an ensemble learning method that builds multiple decision trees and merges them together to get a more accurate and stable prediction:

\* **Multiple Trees** : Each tree is trained on a different subset of the data, which helps to capture different patterns in the data.

\* **Averaging Predictions** : For regression tasks, the final prediction is obtained by averaging the predictions of all individual trees, which helps to reduce variance and avoid overfitting.

### ### 2. Robustness to Overfitting

Random Forests are generally robust to overfitting, especially when dealing with large datasets:

\* **Bootstrap Aggregation (Bagging)** : By training each tree on a bootstrap sample (randomly sampled with replacement) from the training data, the model reduces the risk of overfitting.

\* **Random Feature Selection** : At each split in a tree, a random subset of features is considered, which further helps to decorrelate the trees and improve generalization.

### ### 3. **Handling of Non-Linear Relationships**

Random Forest can capture complex non-linear relationships in the data:

\* **Decision Trees** : Each tree in the forest can model non-linear interactions between features, making the ensemble capable of handling complex patterns.

### ### 4. **Feature Importance**

Random Forests can provide estimates of feature importance, which can be useful for understanding the underlying data:

\* **Feature Importance Scores** : The model can rank features based on their importance, helping to identify which features contribute most to the prediction.

### ### 5. **Flexibility and Scalability**

Random Forests are flexible and can handle both small and large datasets effectively:

\* **Scalability** : The model can be scaled to handle large datasets by increasing the number of trees ( `n_estimators` ).

\* **Parallelism** : Training of individual trees can be done in parallel, leveraging multi-core processors to speed up training.

### ### 6. **Hyperparameter Optimization**

The use of `GridSearchCV` allows for an exhaustive search over the specified hyperparameter grid to find the best combination of parameters:

\* **Cross-Validation** : Using cross-validation helps to ensure that the model performs well on unseen data and prevents overfitting.

\* **Hyperparameter Tuning** : By testing different values of `n_estimators`, `max_depth`, `max_features`, and `criterion`, the model is fine-tuned to achieve the best performance.

### 7. **Robustness to Outliers and Noise**

Random Forests are relatively robust to outliers and noise in the data:

\* **Averaging Mechanism** : The averaging of predictions from multiple trees helps to smooth out noise and reduce the impact of outliers.

### 8. **Reduced Bias**

Random Forests can achieve low bias by combining multiple weak learners (decision trees):

\* **Low Bias** : Each individual tree might have high bias, but combining them reduces the overall bias of the ensemble.

### Summary

Random Forest's ability to handle non-linear relationships, robustness to overfitting, flexibility, scalability, and effective hyperparameter tuning make it a powerful tool for stock price prediction. The provided code leverages these advantages to train and fine-tune the model for optimal performance.