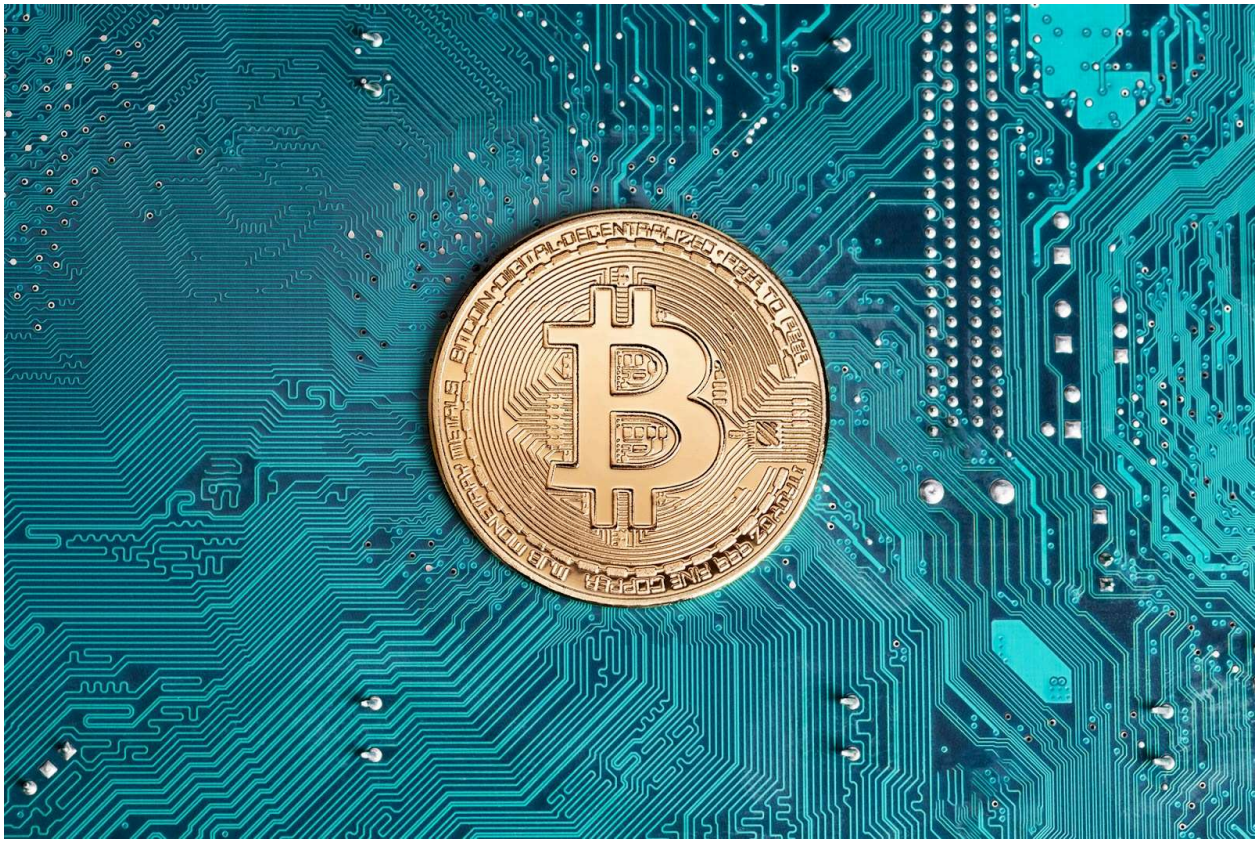
Machine Learning Course

Bitcoin Price Classification Task (USD)

11-MAY-2024

Under the supervision of doctor/ Abdelrhman Header



Team Members :

Mahmoud Mohamed Ahmed Mohamed

Mohamed Eid Saber Saleh

Ahmed Magdy Mohamed

Table Of Content :

Introduction :

In this report we discuss the bitcoin price trades in each minute from 1 January, 2021 to 12 May, 2021 on 1 minute interval (For each row).

This data set is generated by the help of Binance Api (The Binance API is a method that allows you to connect to the Binance servers via Python or several other programming languages. With it, you can automate your trading.)

More specifically, Binance has a RESTful API that uses HTTP requests to send and receive data. Further, there is also a WebSocket available that enables the streaming of data such as price quotes and account updates.

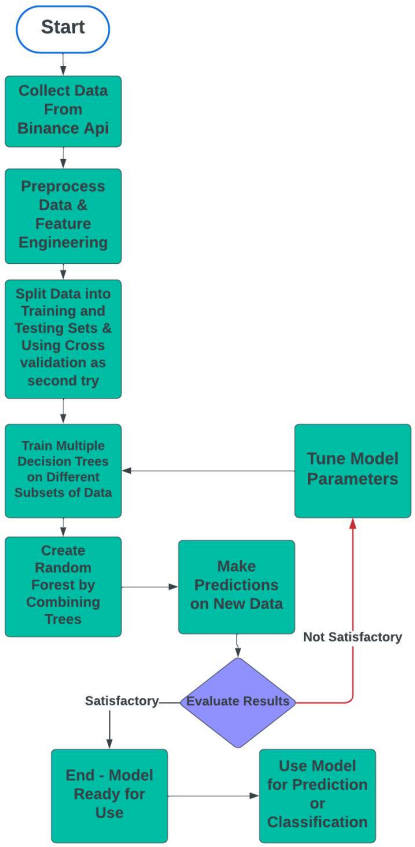
We used Random Forest model as a supervised ML algorithm for the classification task

Random forests for classification build a multitude of decision trees on random subsets of data and features. Each new data point is voted on by the trees, with the majority class label becoming the forest's prediction. This approach reduces overfitting and improves overall accuracy

Methodology :

1. Load and analyze trading data using pandas.
2. Generate relevant features for a machine learning model.
3. Train a Random Forest model for price prediction.
   - 3.1. Create a large number of decision trees (hundreds or even thousands)
   - 3.2. Each tree is trained on a random subset of the training data(with replacement, a technique called bootstrapping). This Injects diversity into the forest.
   - 3.3. When making a decision at each node of a tree, only a random subset of features is considered as potential splitting criteria. This further diversifies the trees and prevents overfitting.
4. Backtest a trading strategy based on the model's predictions.
5. Evaluate the performance of the trading strategy.

6. Flowchart:



7. **The Time Complexity analyze :** **The time complexity for training a random forest classifier depends on several factors(Number of trees in the forest(n_estimators), Number of samples (188316), Number of features m (12 feature), depth of each tree(d))**

Assuming default settings, where n_estimators is typically 100, and trees are grown until all leaves are pure or until they contain less than a minimum number of samples, the complexity is approximately:

$$O(n * m * n(trees) * logn)$$

$$O(188316 * 12 * 100 * log(188316))$$

7.1. **Using Cross validation with CV = 5 :**

$$O(n * m * n(trees) * logn * cv)$$

$$O(188316 * 12 * 100 * log(188316) * 5)$$

Experimental Simulation

1. **Programming Languages and Environments**
    - 1.1. **Languages:**
        - 1.1.1. **Python:** The primary language used in the project. Python is chosen for its simplicity, readability, and extensive libraries for machine learning, such as scikit-learn.
    - 1.2. **Environments**:
        - 1.2.1. **Jupyter Notebook:** An interactive computing environment that allows for writing and running code in a cell-based format, which is particularly useful for data analysis and iterative development
    - 1.3. **Libraries:**
        - 1.3.1. **scikit-learn:** A machine learning library in Python used for implementing the Random Forest algorithm.
        - 1.3.2. **numpy:** Used for numerical operations and handling arrays.
        - 1.3.3. **pandas:** Utilized for data manipulation and analysis.
        - 1.3.4. **seaborn:** For statistical data visualization.
        - 1.3.5. **matplotlib:** For plotting graphs
2. **Details of Programming the Primary Function and Procedures**

    Primary Function: Training, Evaluating, and Visualizing a Random Forest Classifier

    - 2.1. Importing Libraries
    - 2.2. Loading and processing data
    - 2.3. Feature Engineering
    - 2.4. Cross Validation
    - 2.5. Creating, Training, and Evaluating Model
    - 2.6. Final Model Evaluating and Visualization

3. Test Cases Used to Test the Programmed Codes

3.1. Basic Functionality:

3.1.1. Objective: Ensure the model can be created, trained, evaluated, and visualized without errors.

3.1.2. Method: Run the full script with a subset of the data.

3.1.3. Expected Outcome: The model should be trained, cross-validation scores printed, and the confusion matrix plotted.

3.2. Edge Cases:

3.2.1. Empty Dataset:

3.2.1.1. Objective: Test how the model handles an empty dataset.

3.2.1.2. Method: Pass empty DataFrames to the training function.

3.2.1.3. Expected Outcome: The function should raise an appropriate error.

3.2.2. Single Class Target:

3.2.2.1. Objective: Test the model's behavior with a target variable having a single class.

3.2.2.2. Method: Provide a target variable with only one class.

3.2.2.3. Expected Outcome: The model should raise an error or warning.

3.3. Performance Testing:

3.3.1. Large Dataset:

3.3.1.1. Objective: Ensure the model can handle large datasets efficiently.

3.3.1.2. Method: Use the full dataset with 188316 rows and 12 columns.

3.3.1.3. Expected Outcome: The model should train, evaluate, and visualize successfully.

4. Setting Program Parameters and Constants:

4.1. Random Forest Parameters:

4.1.1. n_estimators: Number of trees in the forest. Default is 100.

4.1.2. max_depth: Maximum depth of the tree. None means nodes are expanded until all leaves are pure.

4.2. Cross-Validation Parameter:

4.2.1. Number of cross validation folds = 5

4.3. Random State:

4.3.1. Objective: Ensure reproducibility.

4.3.2. Setting random_state = 0

Results and Technical Discussion

1. **The main problem result:** The main problem result is classifying the direction of Bitcoin price changes depending on whether the price will increase or decrease on a minute-by-minute basis. This prediction is based on the percentage change in Bitcoin prices between consecutive minutes.
2. **The Outputs:**
    - **2.1. Cross-Validation Scores:** These scores represent the model's performance during the training phase using 5-fold cross-validation. The scores provide an indication of how well the model generalizes to unseen data during training.
    - **2.2. Confusion Matrix**: The confusion matrix visualizes the model's performance on the test set by showing the number of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN).
    - **2.3.**
3. **Test/Evaluation experimental procedure and analysis of results:**
    - **3.1. Data preparation:**
        - **3.1.1.** Convert 'Open Time' and 'Close Time' from unix timestamp to human-readable time.
        - **3.1.2.** Check for missing values in the dataframe
    - **3.2. Feature Engineering:**
        - 3.2.1. Calculate the returns from the 'Close' price
        - 3.2.2. Calculate the standard deviation (volatility) over a 7-period timescale
        - 3.2.3. Calculate the 7-period moving average
    - **3.3. Data Splitting:** The dataset was split into training (70%) and testing (30%) sets to evaluate the model's performance on unseen data.
    - **3.4. Model Training:** A Random Forest Classifier with default parameters (e.g100 trees, no maximum depth limit) was created and trained on the training set.
    - **3.5. Cross-Validation:** the model's performance using 5 fold cross-validation on the training set to ensure robustness and avoid overfitting

3.5.1. **Analysis Cross validation Scores:** the cross-validation scores for the random forest model were [100, 100, 100, 100, 100] despite these scores indicating a great performance during cross-validation, that means the model discovers the pattern of the classification right not having overfitting.

3.6. **Final Evaluation:** The trained model was tested on the test set, and its performance was assessed using a confusion matrix to visualize the distribution of correct and incorrect predictions.

3.6.1. **Analysis Final Evaluation:** When evaluated on the test set, the model achieved an accuracy of 100%, correctly predicting all instances of positive and negative Bitcoin price changes.

3.7. **Analysis Performance Interpretation:**

3.7.1. The perfect accuracy on the test set suggests that the model has achieved an exceptionally high level of performance

3.7.2. The similarity between the cross-validation scores and the test set accuracy may indicate that the test set was particularly well-suited to the patterns the model learned during training.

3.8. **Analysis Quality of result :** While achieving 100% accuracy is an impressive result, it is crucial to validate these findings further to ensure that the model is genuinely robust and not overfitted.

Conclusions

1. **Model Performance:** The Random Forest model achieved a perfect accuracy of 100% on the test set for predicting the direction of Bitcoin trades on a minute-by-minute basis. This remarkable result demonstrates the model's potential to capture the underlying patterns in the data accurately.

Recommendation for Future Work

1. Expand Feature Set: Incorporate additional features like technical indicators, trading volume, and external data sources to enhance model robustness.
2. Advanced Models: Experiment with other machine learning models (e.g., gradient boosting, neural networks) and ensemble methods to potentially improve performance.
3. Robust Validation: Use time-series cross-validation and out-of-sample testing to ensure the model's generalizability.
4. Regularization and Tuning: Continue hyperparameter tuning and apply regularization techniques for optimal model performance.
5. Real-time Implementation: Develop a real-time system for continuous model updates and performance monitoring.
6. Comprehensive Analysis: Analyze feature importance and report additional performance metrics (e.g., precision, recall) for a thorough evaluation.

References

- Binance API. (2021). "Binance API Documentation." Available at: Binance API Docs
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, E. (2011). "Scikit-learn: Machine Learning in Python." Journal of Machine Learning Research, 12, 2825-2830. Available at: scikit-learn
- Hastie, T., Tibshirani, R., & Friedman, J. (2009). "The Elements of Statistical Learning: Data Mining, Inference, and Prediction." Springer.

Appendix

Project source code can be accessed via the provided link:
https://github.com/mahmoudwahman006/Time-Series-Projects