Intelligent System
Assignment 1 - Option B
Applying Macroeconomic Indicators to Predict Stock Prices

Xuan Tuan Minh Nguyen - 103819212

October 27, 2024

Contents

Abstract

The major purpose of this research is to showcase **the application of machine learning** in predicting stock prices with the combination of **historical** data of **Macroeconomic Indicators**. In this research, we will focusing on researching the affect of the **Macroeconomic Indicators** to the stock price prediction. We will be using two most popular models for time series prediction, which are the **XGBoost** and **RandomForest** models. In addition, the stock price data will be taken from **Yahoo Finance**, and the Macroeconomic Indicator data will be taken from **FRED**. The research will be conducted on **Tesla Inc.** (TSLA) stock from the date of 2015-01-01 to 2023-08-25. This research will deep dive into **differentiating** each **potential solutions** to improve the performance of the **v0.5** codebase, **pick and implement** the solution that fits the most for the context. And **visualize** the **prediction results** with the actual stock prices to evaluate the performance of the models.

1 Introduction

With the significant development of the stock market, more and more investors are interested in pouring their money into the stock market to gain profits. However, with the unpredictable nature of the stock market, it is very difficult for investors to see when is the perfect time to buy or sell their stocks. Therefore, with the phenomenon growth of the **machine learning** field, investors could empower the **machine learning** to predict the future direction of the stock market and could make their own decisions based on the predicted results.

In this research, we will be exploring the application of **machine learning** in the context of predicting the stock prices. Stock price prediction could be a very complicated task as there are multiple factors to consider, such as the **fundamental analysis** and **technical analysis**. In this research, we will be analyzing the potential solutions to improve the performance of the **v0.5** model, which itself strongly relies on the **historical stock prices** and have a bad performance in the prediction accuracy. After that, we will be discussing and picking the best solution to give our **v0.5** an update to **v0.6**, which has a better and more reliable prediction accuracy.

2 Review of the Potential Solutions

Nowadays, there are multiple different approaches that have been proposed to improve the performance of the traditional stock price prediction models. These approaches aimed for increase the accuracy and broaden the scope of the prediction features by adding more data sources to the model, extending the modeling and data processing techniques and inputting external factors that could affect the stock prices. These solutions could be categorized into:

2.1 Improving the current Ensemble Prediction Model

To improve the current ensemble prediction model, using an ensemble of multiple models could be a good solution. Two key approaches that could be used to improve the performance of the ensemble prediction model are:

- **Stacking Ensemble**: With the powerful non-linear nature of the stacking ensemble, which allows it to combine the predictions from multiple models to create a new model that has a better performance than the individual models, we could use the stacking ensemble to ensemble the predictions from the state of the are models, such as **Random Forest** and **XGBoost** models, to achieve a higher preidction accuracy and reducing the generalization error. [6]

- **Bagging / Boosting Ensemble**: In addtion to the stacking ensemble, the Bagging and Boosting are the two most popular ensemble methods to improve the performance of the prediction model. The bagging ensemble method is used to reduce the variance of the prediction model, while the boosting ensemble method is used to reduce the bias of the prediction model. Thus could help to improve the prediction accuracy and reducing the generalization error. [6]

2.2 News and Social Media Content

The sentiment of a market could strongly affect the stock prices, as the sentiment of the market could affect the behavior of the investors, which in turn could affect the stock prices. Therefore, with the development of the **natural language processing** field, more and more research aimed to use the sentiment analysis to predict the future direction of the stock prices. The following methods were explored:

- **News and Social Media Content**: By collecting the news and social media content, we could use the sentiment analysis to predict the future direction of the stock prices, like what [2] did.
- **Sentiment Analysis**: Natural Language Processing (NLP) is a field that focuses on the interaction between computers and humans using natural language. With the development of the NLP field, we could use the sentiment analysis to predict the future direction of the stock prices, which could result in a higher accuracy. [2]

2.3 Macroeconomic Indicators

It is a well-known fact that some macroeconomic indicators could have a strong correlation with the stock prices, such as the **unemployment rate** and **inflation rate** [4]. The following methods were found to be effective in predicting the stock prices:

- **Inclusion of Macroeconomic Indicators**: Beside from the stock-specific data, such as the open, close, high and low prices, the volume of the stocks traded and the adjusted close price, we could include the macroeconomic indicators that have a strong correlation with the stock prices to improve the prediction accuracy, such as **GDP**, **inflation rate** and **unemployment rate**. [4]
- **Technical Indicators**: By adding technical indicators, such as **RSI**, **MACD** and **William Percent Range**, we could see the overbought and oversold levels on a chart and can help predict where a price is likely to go next, based on past performance. [3]

2.4 Advanced Model Optimization Techniques

Different optimization techniques are researched and have been widely used for improving the performance of the prediction model, such as:

- **Hyperparameter Optimization**: By finding the optimal hyperparameters for the model, we could improve the prediction accuracy and reducing the generalization error. [1]
- **Random Search**: Random search is a technique that randomly samples a set of hyperparameters from a given distribution and then evaluates the performance of the model with those hyperparameters. This technique can help to find the optimal hyperparameters for the model, which could improve the prediction accuracy and reducing the generalization error. [1]

- **Early Stopping and Learning Rate Scheduling**: Early stopping is a technique that stops the training process when the performance of the model starts to decrease, which could help to prevent overfitting. Learning rate scheduling is a technique that adjusts the learning rate during the training process, which could help to increase the prediction accuracy and reducing the error.

2.5 Advanced Model Applications

Moreover, there are some advanced applications that could be used to improve the performance of the prediction model, such as:

- **Deep Reinforcement Learning**: Deep Reinforcement Learning (DRL) allows agents to learn directly from raw sensory input, such as images or sounds, without relying on handcrafted features or domain-specific knowledge. Thus making DRL highly flexible and capable of automating tasks, including stock trading. [7]
- **Time series Analysis**: Using time-series based models, such as **ARIMA**, we could effectively predict the short-term stock price and stock returns movements, thanks to the ability of time-series models to capture the underlying patterns and trends in the data. [5]

3 Implementation

After carefully reviewing and considering the researched potential approaches, I have decided to go with the **Macroeconomic Indicators** approach, as it has a strong correlation with the stock prices, has a high compatibility with the current v0.5 codebase, and has a good prediction accuracy. In addition, the **Macroeconomic Indicators** approach could give me a good point of view to how the macroeconomic indicators could affect the stock prices and how we could use it for the prediction.

In this implementation, the **v0.6** will include some of the enhancements that focused on both adding technical indicators and including macroeconomic variables, change the current models into **XGBoost** and **RandomForest** models, and reduce some of the unnecessary features that could result in a less memory usage and a higher prediction accuracy.

3.1 Environment Setup

3.1.1 Create a new environment using Conda CLI

There are several different procedures to create an environment, but the given procedure below will encapsulate all of the required steps to create a safe and clean environment using Conda.

- Navigate to the Github repository that contains the source code for v0.6.
- Once navigated, download the source code by clicking on Code → Download ZIP or use the following command in the CLI (Terminal):

```
git clone https://github.com/cobeo2004/cos30018.git
```

- Once the source code is successfully cloned (downloaded), navigate to the Week 7/v0.6 folder and execute the file conda-config.sh using the following command:

```
bash conda-config.sh
```

- The given file config.sh will execute the following procedure:
  - Generate an environment with a pre-defined name (you can change the name if you want to) in Python 3.10.9 by using the command:

    ```
    conda create -n cos30018_env_w7_v0.6 python=3.10.9
    ```

  - Activate the created environment using:

    ```
    conda activate cos30018_env_w7_v0.6
    ```

  - Check and validate if the conda environment is successfully initialized by running `conda info --envs` for listing conda environments and see which environment that we are in and current Python version using `python --version`.

3.1.2 Installing required dependencies

Once the environment is successfully initialized, we can start installing the dependencies (libraries) that are required by the program. There are multiple pathways to install dependencies in Python, but the most popular steps are:

- Scan through the code to find out the required dependencies; for example, consider the file stock_prediction.py. We could see that there are quite a few required dependencies, such as: numpy matplotlib pandas tensorflow scikit-learn pandas-datareader yfinance TA-lib statsmodels plotly pmdarima fredapi xgboost. Especially, **statsmodels** will be used for finding parameters for ARIMA model, **fredapi** for getting macroeconomic indicators data and **xgboost** for constructing XGBoost model.
- Once dependencies are scanned, use the following command to install the dependencies:

  ```
  pip install numpy matplotlib pandas tensorflow scikit-learn
  pandas-datareader yfinance TA-lib statsmodels plotly pmdarima fredapi
  xgboost
  ```

- Another step is to list all required libraries into a requirements.txt file, and using the following command to install the required dependencies:

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

Figure 1: Example of requirements.txt

4 Understanding the Integration of Macroeconomic Indicators

4.1 Macroeconomic Variables Processing

4.1.1 Codebase

```python
import pandas as pd
from .env import env
from fredapi import Fred
from src.processing import start, end


class FredHelper:
    def __init__(self) -> None:
        # For the API key, create a .env file in the root directory (same level as main.py file), and add the following line:
        # FRED_API_KEY='ad59ab772d6ea3ca06ef9b9188ece7eb'
        self.fred = Fred(api_key=env.fred_api_key)

    def get_macroeconomic_data(
        self, start_date: str = start, end_date: str = end
    ) -> pd.DataFrame:

        gdp = self.fred.get_series("GDP", start_date=start_date, end_date=end_date)
        unemployment = self.fred.get_series(
            "UNRATE", start_date=start_date, end_date=end_date
        )
        inflation = self.fred.get_series(
            "CPIAUCSL", start_date=start_date, end_date=end_date
        )

        return pd.DataFrame(
            {
                "GDP": gdp,
                "Unemployment": unemployment,
                "Inflation": inflation,
            }
        )
```

Figure 2: Codebase for importing and processing data from FRED

4.1.2 Functionalities

In general, the given code will be used for loading the macroeconomic indicators from **FRED** and processing them to be used for the prediction model. The given code will perform the following procedures:

- **Loading Data**: The function will first load the required series, which are **GDP, Unemployment Rate and Inflation Rate** from FRED using the `get_series()` function from `fredapi` library.
- **Processing Data**: Once the data is loaded, the function will return a dataframe that contains the loaded data.

4.2 Model Preparation and Integration

4.2.1 Codebase

```python
from numpy import ndarray
from .IModel import IModel
from xgboost import XGBRegressor


class XGBoostModel(IModel):
    def __init__(
        self,
        n_estimators: int = 500,
        learning_rate: float = 0.01,
        max_depth: int = 6,
        verbosity: int = 1,
    ) -> None:
        self.xgb_model = XGBRegressor(
            n_estimators=n_estimators,
            learning_rate=learning_rate,
            max_depth=max_depth,
            verbosity=verbosity,
        )

    def train(self, x_train: ndarray, y_train: ndarray) -> XGBRegressor:
        self.xgb_model.fit(x_train, y_train)
        return self.xgb_model

    def predict(self, x_test: ndarray) -> ndarray:
        return self.xgb_model.predict(x_test)

    def score(self, x_test: ndarray, y_test: ndarray) -> float:
        return self.xgb_model.score(x_test, y_test)
```

Figure 3: Codebase to create XGBoost model

```python
from numpy import ndarray
from .IModel import IModel
from sklearn.ensemble import RandomForestRegressor


class RandomForestModel(IModel):
    def __init__(
        self,
        n_estimators: int = 500,
        max_depth: int = 6,
        min_samples_split: int = 2,
    ) -> None:
        self.rf_model = RandomForestRegressor(
            n_estimators=n_estimators,
            max_depth=max_depth,
            min_samples_split=min_samples_split,
        )

    def train(self, x_train: ndarray, y_train: ndarray) -> RandomForestRegressor:
        self.rf_model.fit(x_train, y_train)
        return self.rf_model

    def predict(self, x_test: ndarray) -> ndarray:
        return self.rf_model.predict(x_test)

    def score(self, x_test: ndarray, y_test: ndarray) -> float:
        return self.rf_model.score(x_test, y_test)

```

Figure 4: Codebase to create RandomForest model

4.2.2 Functionalities

In general, the major purpose of both **XGBoostModel** and **RandomForestModel** class is to initialize, run and predict the stock prices using the **XGBoost** and **RandomForest** models. The following functionalities are implemented in both classes:

- **Initialization**: The class will initialize the **XGBoost** (**RandomForest**) model with the required parameters. By default, the **XGBoost** model will have 500 trees, a learning rate of 0.01, a maximum depth of 6 and a verbosity of 1, and the **RandomForest** model will have 100 trees, a maximum depth of 6 and a verbosity of 1.
- **Training**: The class will train the model with the training data.
- **Prediction**: The class will predict the stock prices with the test data.
- **Scoring**: The class will score the model with the testing data.

4.3 Data Processing

4.3.1 Codebase

Figure 5: Codebase to process data

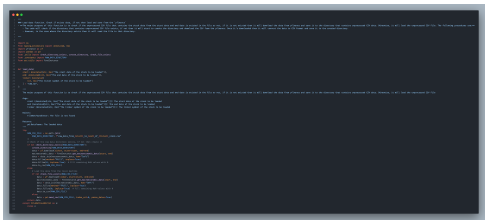

Figure 6: Codebase to load data

Figure 7: Codebase to prepare data

4.3.2 Functionalities

In general, the major purpose of these three functions is to process the data and prepare it for the prediction model. The following procedures are implemented in these three functions:

- **Loading Data**: The function will load the data from the CSV file into a pandas dataframe.
- **Processing Data**: The function will process the data by dropping the unnecessary columns and normalizing the data.
- **Splitting Data**: The function will split the data into training and testing sets using `train_test_split()` function from `sklearn` library.

5 Deploying and Testing the Codebase

5.1 Codebase used for testing

```python
if __name__ == "__main__":
    from src.processing.index_ext import prepare_data
    from src.model import XGBoostInstance, RandomForestInstance
    from src.processing.constants import FEATURE_COLS
    from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
    from src.charts import PredictionResultInstance, RSIandMACDInstance

    data, x_train, x_test, y_train, y_test, scalers = prepare_data()

    # XGBoostInstance is a singleton instance of XGBoostModel
    xgb_train = XGBoostInstance.train(
        x_train.reshape(x_train.shape[0], x_train.shape[1] * len(FEATURE_COLS)), y_train
    )
    xgb_predict = XGBoostInstance.predict(
        x_test.reshape(x_test.shape[0], x_test.shape[1] * len(FEATURE_COLS))
    )
    # Check NaN
    xgb_mse = mean_squared_error(y_test, xgb_predict)
    xgb_r2 = r2_score(y_test, xgb_predict)
    xgb_mae = mean_absolute_error(y_test, xgb_predict)
    # print("XGBoost Model Trained: ", xgb_predict)
    print("XGBoost Model MSE: ", xgb_mse)
    print("XGBoost Model R2: ", xgb_r2)
    print("XGBoost Model MAE: ", xgb_mae)

    # RandomForestInstance is a singleton instance of RandomForestModel
    rf_train = RandomForestInstance.train(
        x_train.reshape(x_train.shape[0], x_train.shape[1] * len(FEATURE_COLS)), y_train
    )
    rf_predict = RandomForestInstance.predict(
        x_test.reshape(x_test.shape[0], x_test.shape[1] * len(FEATURE_COLS))
    )
    rf_mse = mean_squared_error(y_test, rf_predict)
    rf_r2 = r2_score(y_test, rf_predict)
    rf_mae = mean_absolute_error(y_test, rf_predict)
    # print("Random Forest Model Trained: ", rf_predict)
    print("Random Forest Model MSE: ", rf_mse)
    print("Random Forest Model R2: ", rf_r2)
    print("Random Forest Model MAE: ", rf_mae)

    xgb_predict_inverse = scalers["TargetNextClose"].inverse_transform(
        xgb_predict.reshape(-1, 1)
    )
    rf_predict_inverse = scalers["TargetNextClose"].inverse_transform(
        rf_predict.reshape(-1, 1)
    )

    y_test_inverse = scalers["TargetNextClose"].inverse_transform(y_test.reshape(-1, 1))

    PredictionResultInstance.plot(
        y_test_inverse, xgb_predict_inverse, rf_predict_inverse
    )
    RSIandMACDInstance.plot(data)

```

Figure 8: Codebase for testing

To display the prediction results and the RSI and MACD plots, the given codebase below will be used:

```python
import pandas as pd
from .IChart import IChart
import numpy as np
import matplotlib.pyplot as plt


class RSIandMACDChart(IChart):
    def plot(self, data: pd.DataFrame) -> None:
        rsi_val = data["Close_RSI"]
        macd_val = data["Close_MACD"]
        theta_val = np.linspace(0, 2 * np.pi, len(rsi_val))

        plt.figure(figsize=(10, 5))
        ax = plt.subplot(111, projection="polar")
        plt.plot(theta_val, rsi_val, label="RSI", color="red")
        plt.plot(theta_val, macd_val, label="MACD", color="blue")
        plt.title("RSI and MACD")
        plt.legend()
        plt.show()

```

Figure 9: Codebase for RSI and MACD plots

```python
from .IChart import IChart
import numpy as np
import matplotlib.pyplot as plt
from src.processing import ticker


class PredictionResultChart(IChart):
    def plot(
        self,
        y_test: np.ndarray,
        xgBoostPrediction: np.ndarray,
        rfPrediction: np.ndarray,
    ) -> None:
        plt.figure(figsize=(10, 5))
        plt.plot(y_test, label="Actual", color="red")
        plt.plot(xgBoostPrediction, label="XGBoost", color="blue")
        plt.plot(rfPrediction, label="Random Forest", color="green")
        plt.title(f"Share Price Prediction of {ticker}")
        plt.xlabel("Time")
        plt.ylabel("Share Price")
        plt.legend(loc="best")
        plt.grid(True)

        plt.ylim(
            [
                min(y_test.min(), xgBoostPrediction.min(), rfPrediction.min()),
                max(y_test.max(), xgBoostPrediction.max(), rfPrediction.max()),
            ]
        )
        plt.show()

```

Figure 10: Codebase for drawing prediction result chart compared to actual stock prices

5.2 Testing Results

After running the codebase for testing, we could see that the prediction resultsare quite close to the actual stock prices. The RSI and MACD plots are also showing the interpolated values of the RSI and MACD, which could assist us to predict where the price is likely to go in the future.

```
XGBoost Model MSE:  938.886283593279
XGBoost Model R2:  0.7335489604479835
XGBoost Model MAE:  24.678306842672413
Random Forest Model MSE:  449.8290898418183
Random Forest Model R2:  0.8723408460603185
Random Forest Model MAE:  14.4709678814942
```

Figure 11: Prediction metrics (MSE, R2, MAE) of the XGBoost and RandomForest models

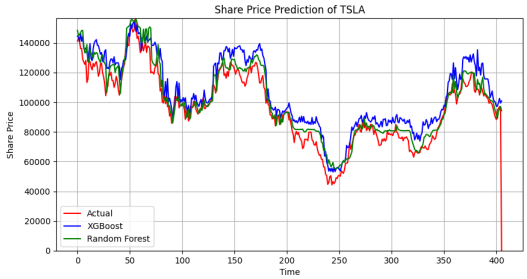

Figure 12: Prediction result of XGBoost and RandomForest models compared to actual stock prices

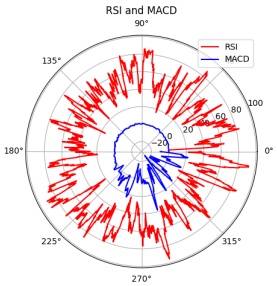

Figure 13: RSI and MACD plots

References

[1] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. `https://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf`, 2012. Accessed: 2024-10-25.

[2] Shri Bharathi and A Geetha. Sentiment analysis for effective stock market prediction. *Int. J. Intelligent Engineering and Systems*, 2018. Accessed: 2024-10-25.

[3] Milan Cutkovic. Best trading indicators: Most popular technical indicators / axi AU. `https://www.axi.com/au/blog/education/trading-indicators`. Accessed: 2024-10-25.

[4] Md Sabbirul Haque, Md Shahedul Amin, Jonayet Miah, Duc Minh Cao, and Ashiqul Haque Ahmed. Boosting stock price prediction with anticipated macro policy changes. *arXiv*, October 2023.

[5] Hardikkumar. Stock market forecasting using time series analysis with ARIMA model. `https://www.analyticsvidhya.com/blog/2021/07/stock-market-forecasting-using-time-series-analysis-with-arima-model/`, July 2021. Accessed: 2024-10-25.

[6] Minqi Jiang, Jiapeng Liu, Lu Zhang, and Chunyu Liu. An improved stacking framework for stock index prediction by leveraging tree-based ensemble models and deep learning algorithms. *Physica A*, 541(122272):122272, March 2020.

[7] Razib Hayat Khan, Jonayet Miah, Md Minhazur Rahman, Md Maruf Hasan, and Muntasir Mamun. A study of forecasting stocks price by using deep reinforcement learning. In *2023 IEEE World AI IoT Congress (AIIoT)*, pages 0250–0255. IEEE, June 2023.