IE41201 - AI For Finance
Assignment 2
Portfolio optimization with machine learning prediction models

Student name: Nguyen Minh Duc
Student ID: 20202026

1 Main Task

1.1 Complete the function

In this first problem, we are asked to fill out the remaining code to complete the function that finds the mean-variance efficient portfolio. The answer can be seen below:

```python
def mean_variance_efficient_portfolio(mu, Sigma, min_return):
    """
    Args:
        mu (numpy.ndarray): Mean returns of the assets.
        Sigma (numpy.ndarray): Covariance matrix of the assets.
        min_return (float): Minimum expected return for the portfolio.

    Returns:
        numpy.ndarray: Portfolio asset weights corresponding to a mean-variance
    efficient portfolio.
    """
    w_3 = cp.Variable(5)
    risk_3 = cp.quad_form(w_3, Sigma)
    ret_3 = mu.T @ w_3
    ######################### Main Problem #################################
    # 1) Find the portfolio asset weights that mean-variance efficient portfolio.
    # Objective : mean-variance efficient portfolio
    # constraints 1 w_3: the sum of the asset weight is to 1
    # constraints 2 ret_3: The asset weights are non-negative
    # constraints 3: The expected return is greater than or equal to the min_return

    prob_3 = cp.Problem(cp.Minimize(risk_3),
                        [cp.sum(w_3) == 1,
                         w_3 >= 0,
                         ret_3 >= min_return])

    # More information is here! https://www.cvxpy.org/index.html
    # 2)  Obtain at least 5 different mean-variance efficient portfolios and plot
    them on the mean-standard deviation plane along with
    # the minimum volatility portfolio and the maximum return portfolio.

    ###################################################################
    prob_3.solve()
    return w_3.value
```

To obtain the answer, we need to transform the following Markowitz model into Python code

$$\begin{aligned} &\underset{w}{\text{minimize}}\ w\Sigma w^\top \\ &\text{subject to } \sum_{i=1}^{n} w_i = 1 \\ &\qquad\qquad w_i \geq 0\ \forall\ i \in [1, n] \\ &\qquad\qquad w^\top \mu \geq r \end{aligned}$$

1.2 Mean-Variance Efficient Portfolio Visualization

In this section, we are asked to obtain at least 5 different mean-variance efficient portfolios and plot them on the mean-standard deviation plane along with the minimum volatility portfolio and the maximum return portfolio. We are going to use two different techniques to obtain the portfolios: using historical data, and using `XGBoost`. The models will be evaluated on both validation and test sets.

1.2.1 Using historical data

Firstly, we will discuss the performance of using historical μ and Σ. Please refer to Figure 1 and Figure 2 for visualizations on the mean-standard deviation plane.

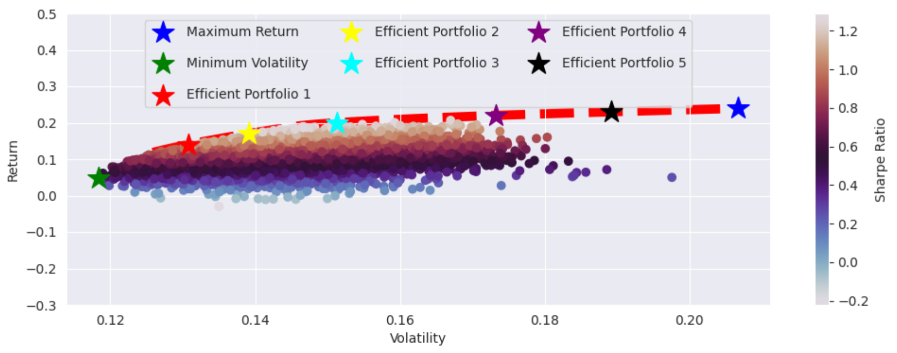

Figure 1: Efficient portfolios calculated using historical mu and Sigma that are evaluated using historical mu and Sigma

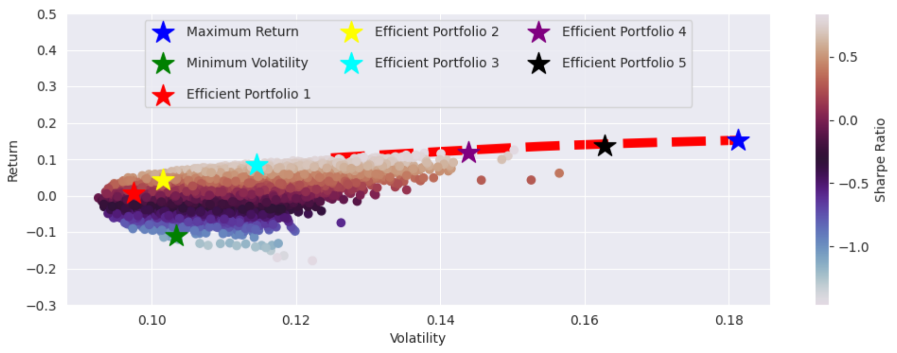

Figure 2: Efficient portfolios calculated using historical mu and Sigma that are evaluated using actual future mu and Sigma

As one could easily observe, the efficient portfolios found by using historical data follow closely to the efficient frontier when evaluating using the validation set. However, in the test set, i.e., the actual future values of μ and Σ, the first two portfolios with low `min_return` do not follow the frontier as close as those with high return. One interesting thing is that the minimum volatility portfolio seems to be in the wrong spot.

To investigate their performance further, let's take a look at their Sharpe Ratio. Please refer to Table 1 and Table 2 for a comprehensive performance comparison (all values are rounded up to 4 decimal places). As one can notice, the increase in return does not cause the increase in Sharpe Ratio as we also need to take care of the portfolio's volatility. On the validation set, the Efficient Portfolio 4 has the best value, while the Maximum Return Portfolio performs best on the test set.

Portfolio	**Return**	**Volatility**	**Sharpe Ratio**
Minimum Volatility	0.0471	0.1183	0.3980
Efficient Portfolio 1	0.2392	0.2067	1.1573
Efficient Portfolio 2	0.1400	0.1307	1.0710
Efficient Portfolio 3	0.1700	0.1391	1.2223
Efficient Portfolio 4	0.2000	0.1513	**1.3223**
Efficient Portfolio 5	0.2200	0.1732	1.2700
Maximum Return	0.2300	0.1892	1.2155

Table 1: Performance on the validation set using historical data

Portfolio	**Return**	**Volatility**	**Sharpe Ratio**
Minimum Volatility	-0.1115	0.1034	-1.0779
Efficient Portfolio 1	0.0052	0.0973	0.0536
Efficient Portfolio 2	0.0429	0.1015	0.4227
Efficient Portfolio 3	0.0844	0.1145	0.7369
Efficient Portfolio 4	0.1169	0.1439	0.8126
Efficient Portfolio 5	0.1351	0.1627	0.8303
Maximum Return	0.1519	0.1813	**0.8376**

Table 2: Performance on the test set using historical data

1.2.2 Using predicted price from XGBoost

Now, we will discuss the performance of using the predicted price from XGBoost. Please refer to Figure 3 and Figure 4 for visualizations on the mean-standard deviation plane.

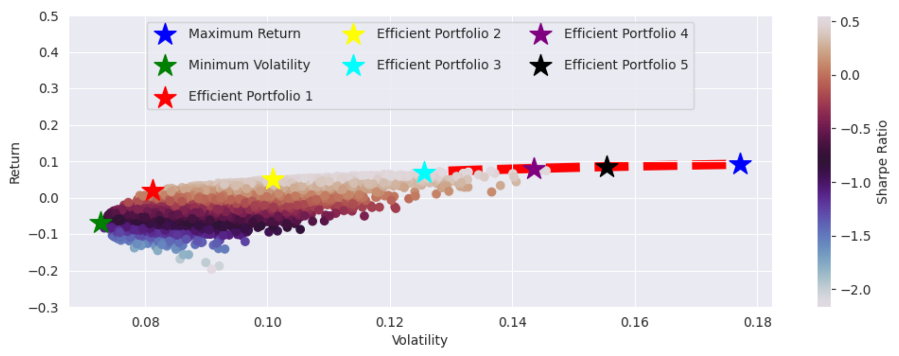

Figure 3: Efficient portfolios calculated using XGBoost predicted mu and Sigma that are evaluated using XGBoost predicted mu and Sigma

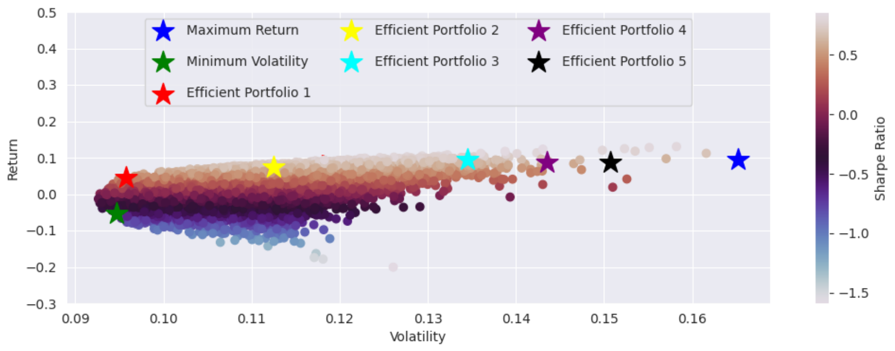

Figure 4: Efficient Portfolios calculated using XGBoost predicted mu and Sigma that are evaluated using actual future mu and Sigma

The same interpretation can also be applied to this case. The efficient portfolios found by using historical data follow closely to the efficient frontier on both validation and test sets. However, the positions of the portfolio on the test set do not seem to align as close to the frontier as the validation set, which is expected. Also, the position of the minimum volatility on the test set seems to make more sense compared to the one using historical data.

To investigate their performance further, let's take a look at their Sharpe Ratio. Please refer to Table 1 and Table 2 for a comprehensive performance comparison (all values are rounded up to 4 decimal places). As one can notice, the increase in return does not cause the increase in Sharpe Ratio as we also need to take care of the portfolio's volatility. On the validation set, Efficient Portfolio 4 has the best value, while Efficient Portfolio 3 performs best on the test set.

Portfolio	**Return**	**Volatility**	**Sharpe Ratio**
Minimum Volatility	-0.0696	0.0727	-0.957
Efficient Portfolio 1	0.0200	0.0813	0.2461
Efficient Portfolio 2	0.0500	0.1009	0.4956
Efficient Portfolio 3	0.0700	0.1256	0.5574
Efficient Portfolio 4	0.0800	0.1435	**0.5574**
Efficient Portfolio 5	0.0850	0.1555	0.5467
Maximum Return	0.0916	0.1772	0.5166

Table 3: Performance on the validation set using predicted price from XGBoost

Portfolio	**Return**	**Volatility**	**Sharpe Ratio**
Minimum Volatility	-0.0521	0.0946	-0.5503
Efficient Portfolio 1	0.0443	0.0957	0.4627
Efficient Portfolio 2	0.0748	0.1125	0.6650
Efficient Portfolio 3	0.0951	0.1345	**0.7070**
Efficient Portfolio 4	0.0878	0.1435	0.6117
Efficient Portfolio 5	0.0873	0.1507	0.5796
Maximum Return	0.0950	0.1652	0.5753

Table 4: Performance on the test set using predicted price from XGBoost

2 Quizzes

2.1 Why we use shift(-1) in the following code

```python
def fetch_stock_data(symbol):
    """
    Fetch the stock data for a given symbol using Yahoo Finance API.

    Args:
        symbol (str): Stock symbol to fetch data for.

    Returns:
        pandas.DataFrame: Stock data for the given symbol.
    """
    ticker = yf.Ticker(symbol)
    data = ticker.history(period="5y", interval="1d").iloc[:, :5].reset_index()
    print(data)
    data['Close'] = data['Close'].shift(-1) # Quiz 1. Why use shift -1?
    data.dropna(inplace=True)
    data.drop(columns=['Volume'], inplace=True)
    return data
```

The `shift` method of a `DataFrame` in `pandas` can be used when we want to shift a column for some steps forward (when the input is positive) or backward (when the input is negative) [4]. By introducing -1 to the parameter, the `close` column of the `data` will be shifted "backward" by 1 step, i.e., the next day's close price will become today's close price. Since we want to predict tomorrow's closing price, `shift(-1)` effectively aligns today's input data with the ground truth.

2.2 Why we should use cross-validation

As we have discussed in the tutorial slides, validation helps to prevent over-fitting by giving the model new unseen data, which is a good metric to compare between different usages of hyperparameters or between different architectures. Cross-validation is a technique that extends the concept of validation by dividing the data into multiple subsets (or folds). For each validation step, this technique will use different subsets for training and different subsets for validation, which further ensure the model's ability to generalize well with unseen data as the model is evaluated on different unseen data on each epoch, not just the same one as in the traditional validation. Therefore, it is a more reliable estimation of a model's performance by leveraging the entire data set for both training and evaluation.

2.3 How can we cross-validate a model for time-series prediction tasks?

As we have discussed, cross-validation is a powerful technique to monitor a model's ability to generalize. However, we cannot directly use the traditional cross-validation technique (K-folds) as the data we are dealing with is sequential, which means training or validation data should be contiguous, hence, we cannot divide the data into arbitrary subsets. Fortunately, there are some techniques that can apply the idea of cross-validation to time-series analysis. One technique that we can use is gradually appending the data into our train-valid set. Please refer to Figure 5 [1] for more details. To do this efficiently, Mohammed et al. [3] proposed a novel parallel algorithm to speed up the cross-validation. Another approach is using Sliding Window Validation, where the size of the data used in the train-valid set is determined, and then slide that window across the entire data. Please refer to Figure 6 [2] for a visualization.

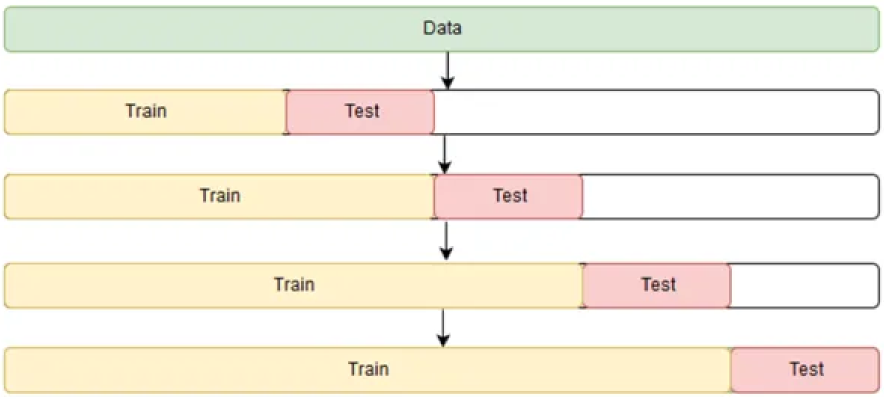

Figure 5: Gradually appending data into train-valid set

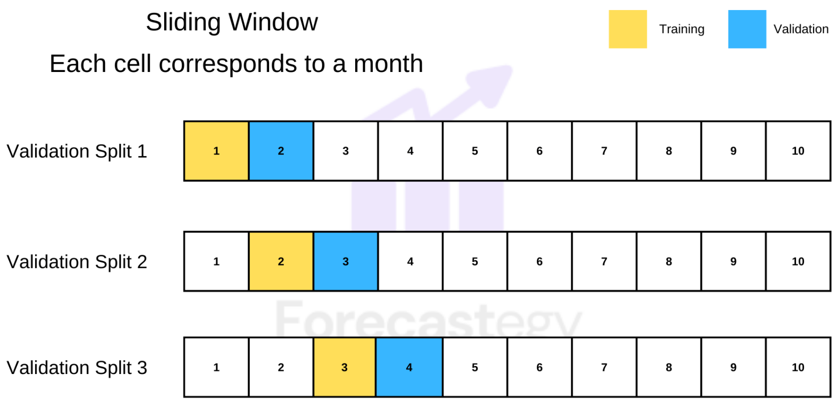

Figure 6: Sliding Window Validation

References

[1] Cross Validation in Time Series. https://medium.com/@soumyachess1496/cross-validation-in-time-series-566ae4981ce4. Accessed: 2023-05-24.

[2] How To Do Time Series Cross-Validation In Python. https://forecastegy.com/posts/time-series-cross-validation-python/. Accessed: 2023-05-24.

[3] Abdulrahim Mohammed, Ahmed Y. Khedr, Duaa AlHaj, Reem Al Khalifa, and Abdulla Alqaddoumi. Time-series cross-validation parallel programming using mpi. *2021 International Conference on Data Analytics for Business and Industry (ICDABI)*, pages 553–556, 2021.

[4] Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56 – 61, 2010.