CTBC Securities Research Project - Replicate the TAIEX

Yu-Ching Liao

MS in Financial Engineering, University of Illinois at Urbana-Champaign

Author Note

LinkedIn: https://www.linkedin.com/in/yu-ching-liao-a45321202/

Github: https://github.com/yu7yu7

Email: josephliao0127@gmail.com / ycliao3@illinois.edu

Table of Contents

Abstract

This research aims to create the portfolio of ETFs, which Adjusted Close Price can approximate the TAIEX. By shorting this portfolio, we can achieve the opportunity to arbitrage from the market. I initially picked 17 ETFs which are representative in Taiwanese stock market. I gathered the daily adjusted close price of each ETF, ranged from January 1st, 2020 to June 30th, 2022 as the features, and the adjusted TAIEX ranged from January 1st, 2020 to June 30th, 2022 as the label. I then extract first 80-percent of the features and label to train the model, mid-10-percent to evaluate the model, and last 10-percent to test if the model is valid.

I utilize linear regression to achieve the coefficient of each ETFs in the portfolio, and process the best subset selection via the out-sample Mean square error.

Eventually, the fitted value achieve via selected model has the correlation coefficient of 99.72% with the actual value, and 7116 of mean square error.

Keywords*:* Linear Regression, Best Subsets Selection, Mean square error, Correlation Coefficient.

Chapter 1 Introduction

To arbitrage from the stock market, to approximate the index as accurate as possible is even important, since if one have the better approach that can estimate the market, the one will as well have more potential to profit from the market inefficiency.

During my time in CTBC Securities, I was assigned to create a portfolio, which can approximate the TAIEX via its market value. An arbitrage is to short the portfolio, at once long the assets that underlies TAIEX, so that we can create the profit from doing so. The restriction of this arbitrage is that the portfolio will have to be even approximate to the TAIEX compared to the assert that underlies TAIEX, otherwise we will not achieve the positive return. As a result, to figure out the way to create the portfolio that is able to approximate the TAIEX as accurate as possible is the first step of this arbitrage strategy.

Though there is already an approach of best subset selection that is chosen via $R^2_{adjusted}$ or *Mallow's* C_p, none of them have included the randomness into account, since they do not choose the model via the data outside the sample. As a result, I decide to design this portfolio via combining the best subset selection with model evaluation technique of the Machine Learning, which is to decide the model via minimum out-sample MSE, so that we can avoid the model to be over fitting or regardless of randomness of the stock market.

I utilize the Python as my programming language, gain the raw data from Yahoo finance, and utilize the adjusted close price of ETFs and TAIEX as the features and label of the model.

Complete Python Code on Github.

Chapter2 Methods Review

2.1 Best Subset Selection

From 17 ETFs, we will get $2^{17} - 1 (= 131071)$ of combinations of features. I have exhaustively regressed all of those combinations and achieve their value out-sample MSE.

2.2 Linear Regression

Assume linear relationship between dependent variable Y and p features $X_1, \cdots, X_p$, $Y_i = \beta_1 X_{i1} + \beta_2 X_{i2} + \cdots + \beta_p X_{ip} + \varepsilon_i, 1 < i < n$, β_p is known as regression coefficient, the amount of change in the dependent variable when the pth regressor changes by 1 unit. We estimate βs via their Ordinary Least Squares (OLS) estimators. Since we are trying to create the optimal portfolio via ETFs, we do not include the intercept in this moment.

2.3 Ordinary Least Squares (OLS) Estimators

Ordinary Least Squares (OLS) Estimators of βs, $\widehat{\beta s}$, are the estimators that can minimize OLS residuals, $\widehat{\varepsilon_i}$. $\widehat{\varepsilon_i} = Y_i - \hat{Y}_i$, which is the difference between actual values, Y_i, and the fitted values, $\hat{Y}_i$,gained via linear regression.

2.4 Out-sample Mean-Square Error (MSE)

We achieve the fitted value via multiplying the coefficients gained from linear regression with out-sample features: $\hat{Y}_{out} = \widehat{\beta} \times X_{out}$. We then gain the out-sample MSE from the MSE of $\hat{Y}_{out}$ and Y_{actual}. $MSE = \frac{\Sigma(\theta - \widehat{\theta})^2}{n}$, where θ is the actual value, $\widehat{\theta}$ is the fitted value and n is the number of regressors.

Chapter 3 Experiment

In this chapter, we conduct numerical experiments to gain the appropriate portfolio, which can approximate the TAIEX via its Adjusted Close Price.

We divide our numerical experiments into three parts; first is utilizing linear regression to gain the β of each ETF, second is to achieve the best subset via out-sample error and third is to test whether this approach of best subset selection is valid via the samples that are not included in the first two parts. In the first part, since in this experiment, we are only using ETFs we selected to gain the model, we do not include intercept when regressing all 131071 combinations of features. The main objective we want to verify in the experiments is the effectiveness of out-sample model evaluation.

3.1 The environment of experiments

We construct our model by real data from Taiwan stock market with programming language Python. In the first part of gaining the coefficient without constraints, βs, we adopt Python package, *yfinance*, to obtain the Adjusted Close Price of each ETF and TAIEX. In the linear regression parts, we adopt Python package, *linear_model*, provided by sklearn. To regress on all 131071 combinations of features, we use the function, combinations provided by Python package, *itertools*. The experiment run on a Macbook Pro, the equipment of the testing environment is as below:

CPU: 2.4 GHz 8-Core Intel Core i9

OS: macOS Ventura Version 13.0.1

RAM: 32 GB 2667 MHz DDR4

3.2 The data of experiments

In this experiment, I have gathered the Adjusted Close Price of ETFs, 0050, 0051, 0052, 0053, 0054, 0055, 0056, 0057, 006203, 006204, 006208, 00633L, 00631L, 00632R, 00701, 00713 and 00731, as the features, and TAIEX as the labels, ranged from January 1st, 2020 to June 30th. The details of those ETFs are shown below in Table 1.

Table 1 Details of ETFs selected

Name of Fund	Stock Code	Underlying Index
Yuanta/P-shares Taiwan Top 50 ETF	0050	Taiwan 50 Index
Yuanta/P-shares Taiwan Mid-Cap 100 ETF	0051	Taiwan Mid-Cap 100 Index
Fubon Taiwan Technology Tracker Fund	0052	FTSE TWSE Taiwan Technology Index
Yuanta/P-shares Taiwan Electronics Tech ETF	0053	TAIEX Electronics Index
Yuanta/P-shares S&P Custom China Play 50 ETF	0054	S&P Custom China Play index
Yuanta/P-shares MSCI Taiwan Financials ETF	0055	MSCI Taiwan Financials Index
Yuanta/P-shares Taiwan Dividend Plus ETF	0056	Taiwan Dividend+ Index
Fubon MSCI® Taiwan ETF	0057	MSCI Taiwan Index
Yuanta/ P-shares MSCI Taiwan ETF	006203	MSCI® Taiwan Index
Sinopac TAIEX ETF	006204	Taiwan Stock Exchange Capitalization Weighted Stock Index（TAIEX）
Yuanta/P-shares Taiwan Top 50 ETF	006208	Taiwan 50 Index
Fubon SSE180 Leveraged 2X Index ETF	00633L	SSE180 Leveraged 2X Index
Yuanta Daily Taiwan 50 Bull 2X ETF	00631L	Taiwan 50 Index
Yuanta Daily Taiwan 50 Bear -1X ETF	00632R	Taiwan 50 Index
Cathay TIP TAIEX Low Volatility Select 30 ETF	00701	TIP TAIEX+ Low Volatility Dividend Plus Select 30 Index
Yuanta Taiwan High Dividend Low Volatility ETF	00713	TIP HDMV Index
Fu-Hwa FTSE Taiwan High Div Low Vol ETF	00731	FTSE Taiwan High Dividend Low Volatility Index

The reason of including those is determined by their volumes of being traded on the Taiwan stock market, which are large enough to be considered as representative ETFs on the market.

3.2.1 Description of data

For the features, there are 17 columns of different ETFs in the data set. There are totally 604 rows, means that we have included 604 daily adjusted close prices as the predictors.

For the labels, we then include adjusted close prices of TAIEX, totally 604 rows as well.

Brief Summary of data is shown below in Table 2.

Table 2 Brief Summary of raw data

Date	0050.TW	0051.TW	0052.TW	0053.TW	0054.TW	0055.TW
1/2/2020	97.6500015	35.4500008	73.3000031	42.7000008	24.75	19.0200005
1/3/2020	97.6500015	35.2999992	72.8499985	42.5800018	25.1200008	19.0300007
6/28/2022	119.800003	53.9500008	98.5500031	56.7999992	28.25	23.5400009
6/29/2022	118.800003	53.7000008	97.4499969	56.5499992	28.25	23.3299999

0056.TW	0057.TW	006203.TW	006204.TW	006208.TW	00631L.TW	00632R.TW
29.0900002	62.0999985	45.3400002	60.0499992	54.0999985	54.4000015	9.93000031
29.1800003	61.6500015	45.6300011	60	54.0499992	54.2000008	9.94999981
28.5300007	83.5500031	57.9000015	77.6999969	69	106	5.80000019
28.1399994	82.8499985	57.5999985	76.9000015	68.3000031	104.050003	5.86000013

00633L.TW	00701.TW	00713.TW	00731.TW	Adj.TAIEX
51.0999985	23.7099991	32.2599983	48.5900002	12100.4805
50.6500015	23.7600002	32.1800003	48.6500015	12110.4297
49.5400009	24.4699993	41.2000008	58.3499985	15439.9199
49.3499985	24.3799992	40.9500008	58.1500015	15240.1299

3.2.2 Data pre-processing

We separate the data with proportion 80%, 10% and 10%. We will then utilize the first 80% to train the model and get the coefficient of each combination of features, mid-10% to evaluate which combination of features provides a best prediction of TAIEX, last 10% to testify whether the model we achieve is valid for prediction.

3.3 Design of experiments

We will show how we design the experiments in this section. As above, the main experiments will be divided into three parts, gaining the β of each ETF, achieving the best subset and testing the model. In the first part, we gather the data which we processed to gain the insights.

3.3.1 Importing the packages

We adopt certain packages which are needed to complete this experiment. The packages we utilize are shown below in Table 3.

Table 3 Imported Packages

Package	Utilization
yfinance	To get the raw ETF and TAIEX data.
itertools	To get the combinations of features.
pandas	To construct the dataframe which we gather information in.
linear_model	To apply linear regression. Imported from sklearn.
mean_squared_error	To get MSE of a fitted value. Imported from sklearn.metrics.
matplotlib.pyplot	To visualize the data.

The programming of importing those packages is shown below.

```
> import yfinance as yf
> import itertools
> import pandas as pd
> import matplotlib.pyplot as plt
> from sklearn import linear_model
> from sklearn.metrics import mean_squared_error
```

3.3.2 Gathering and pre-processing the raw data

To apply *yfinance*, we have to create a string contains all stock codes separated by blank. For example, “0050.TW 0051.TW 0052.TW”. To achieve so, I created a list of stocks and add them into the string with blank to avoid error. The programming to create the string is shown below.

```
> stock_list = ["0050.TW", "0051.TW", "0052.TW",
              "0053.TW", "0054.TW", "0055.TW",
              "0056.TW", "0057.TW", "006203.TW",
              "006204.TW", "006208.TW", "00633L.TW",
              "00631L.TW", "00632R.TW", "00701.TW",
              "00713.TW", "00731.TW"]
> stocks = ""
> for stock in stock_list:
        stocks += stock + " "
```

After gaining the stocks string, we then utilize yfinance to get the raw ETFs data and that of TAIEX. The function in the package *yfinance* we used to get raw data is *download*. The programming of achieving so is shown below.

```
> X = yf.download(stocks, start="2020-01-01", end="2022-06-30")
> y = yf.download("^TWII", start="2020-01-01", end="2022-06-30")
```

We then extract adjust close price from *X* and *y*, divided each of them into three different dataframe with the proportion 80%, 10% and 10%; first 80% as the training dataset, mid-10% as

the evaluation dataset and last 10% as the testing dataset. The programming of achieving so is shown below.

```
> X_train = X.iloc[:int(len(X)*0.8)]['Adj Close']
> X_val = X.iloc[int(len(X)*0.8):int(len(X)*0.9)]['Adj Close']
> X_test = X.iloc[int(len(X)*0.9):]['Adj Close']
> y_train = y.iloc[:int(len(X)*0.8)]['Adj Close']
> y_val = y.iloc[int(len(X)*0.8):int(len(X)*0.9)]['Adj Close']
> y_test = y.iloc[int(len(X)*0.9):]['Adj Close']
```

3.3.3 Get βs of all subsets

We first define a function *fit_linear_reg(X, Y)* to run the linear regression and to gain the parameters of the linear regression at once. Since we are not going to include the intercept (since we are going to create the portfolio only by ETFs, without risk-free rate bond), in the function *linear_model.LinearRegression*, we let *fit_intercept=False*. The programming of achieving so is shown below.

```
> def fit_linear_reg(X, Y)
>     model_k = linear_model.LinearRegression(fit_intercept=False)
>     model_k.fit(X, Y)
>     Params = [*model_k.coef_]
>     return Params
```

We then loop the function *fit_linear_reg(X, Y)* so that we can get the coefficients of each subsets. Here we can simply utilize *range* to get all the number of features. We however instead

use *tnrange* from *tqdm*, which provide us a visualization of the regression process. The programming of achieving so is shown below.

```
> from tqdm import tnrange, tqdm_notebook
> p = len(stock_list)
> feature_list, Param_list, numb_features = [], [], []
> for p in tnrange(1, len(X_train.columns) + 1, desc='Loop...'):
>         for combo in itertools.combinations(X_train.columns, p):
>                 tmp_result = fit_linear_reg(X_train[list(combo)], y_train)
>                 Param_list.append(tmp_result)
>                 feature_list.append(combo)
>                 numb_features.append(len(combo))
> fit = pd.DataFrame({“numb_features”: numb_features,
                        “features”: feature_list,
                        "Parameters": Param_list})
```

3.3.4 Get out-sample MSEs of all subsets

From looping the linear regression above, we obtain all the coefficient of all subsets. We are then able to gain all the *out-sample MSEs* of all the subsets. We first collect all the MSEs into the list, then add the list into the dataframe we obtain above. The programming of achieving so is shown below, and the dataframe we obtain is shown in Table 4.

```
> error = []
> for i in range(len(fit)):
>error.append(mean_squared_error
```

(y_val,X_val[list(fit["features"][i])]@fit["Parameters"][i]))

> fit["Out-sample MSE"] = error

> fit

Table 4 Dataframe of all the subsets (Brief)

	numb_features	features	Parameters	Out-sample MSE
0	1	('0050.TW',)	[123.78829570443297]	179931.9596
15955	6	('0051.TW', '0054.TW', '006208.TW', '00632R.TW', '00633L.TW', '00701.TW')	[85.68230480511245, 52.22929637165934, 124.64475639688298, 8.855677444686059, 1.0667747186820724, 30.521191537882586]	16533.55706

3.3.5 Best subset selection

We now can gain the best subset, which is the one with the minimal *out-sample MSE*. The programming of achieving so is shown below. The best subset we achieve is shown in Table 5 below.

> best_model = fit.loc[fit['Out-sample MSE']

== fit['Out-sample MSE'][fit['Out-sample MSE'].idxmin()]]

> best_model

Table 5 Best subset Selected

	numb_features	features	Parameters	Out-sample MSE
95484	10	('0050.TW', '0051.TW', '0054.TW', '0056.TW', '0057.TW', '006203.TW', '006204.TW', '00632R.TW', '00701.TW', '00731.TW')	[4.949246910508601, 41.080263812246116, 28.00828031731126, 53.5834983033481, 138.19364710408726, -65.16368543940416, 30.357685953822582, 5.6936082481971635, 80.38359235192969, -33.3018665634231]	1108.309675

3.3.6 Get the fitted value of best subset

Now we can achieve the MSE of testing datasets by multiplying the testing dataset of ETFs that is chosen via best subset selection with the parameters chosen via best subset selection. Since the features is not callable due to its data type, we will have to transform features and parameters to list so that they can multiply with each. The programming of achieving so is shown below.

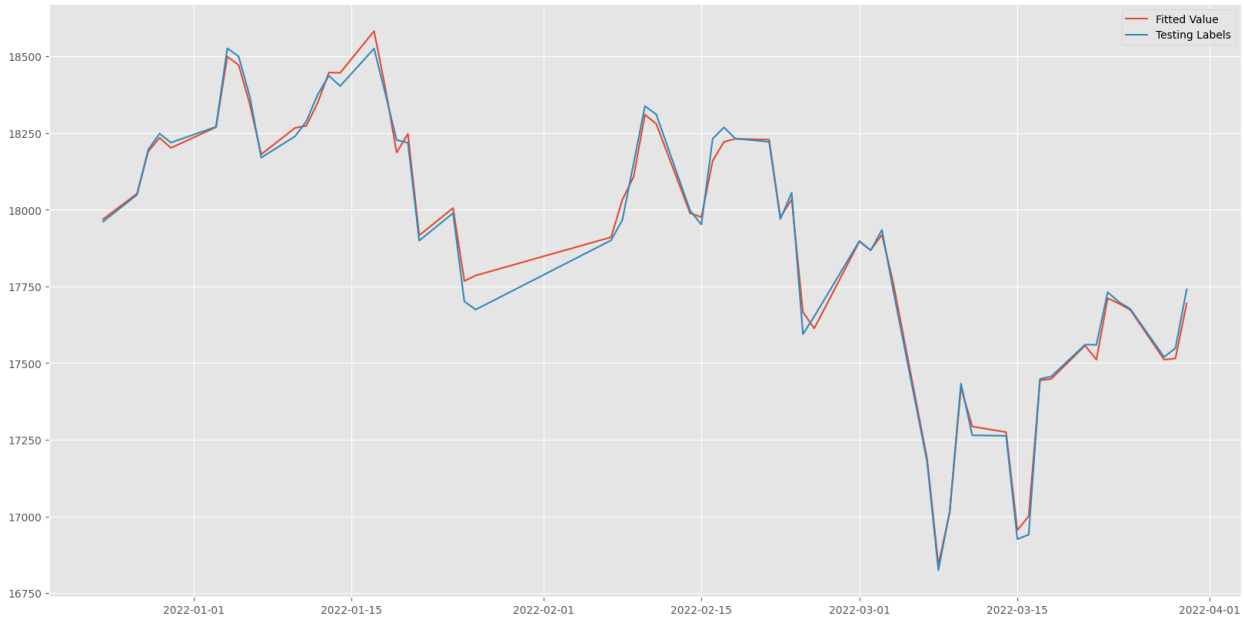
```
> index = fit['Out-sample MSE'].idxmin()
> best_features = []
> best_parameters = []
> for feature in best_model['features'][index]:
>     best_features.append(feature)
> for parameter in best_model['Parameters'][index]:
>     best_parameters.append(parameter)
> fitv = X_test[best_features] @ best_parameters
```

We gain the fitted value that can approximate the testing TAIEX, we will then calculate the MSE and correlation coefficient in the follow chapter as the numerical results. The Table 6 below is the fitted value we obtain, and the Figure 1 shows how our fitted value of evaluation approximate the evaluation datasets of TAIEX.

Table 6 Fitted Value of Testing Features

Date	Fitted Value
2022-03-31	17717.322373
2022-04-01	17566.244293
2022-06-28	15317.910994
2022-06-29	15180.088508

Figure 1 Comparation of Best Fitted Value with Evaluation label of TAIEX



Chapter 4 Numerical Result

In this section, we will display the numerical results of our experiment for validating its effectiveness. We will compare our result of fitted value with the actual testing labels of TAIEX via their MSE and correlation coefficient. We will then see if our approach of prediction is valid.

4.1 Mean Square Error (MSE)

Via MSE of testing labels of TAIEX and the fitted value we achieve, we can notice how distant the fitted value and the TAIEX are. From other method I have tried (e.g. Best subset selection via $R^2_{adjusted}$ or Machine Learning model with Stochastic Gradient Decent), I had achieved the MSE value with approximately 30000 to 50000, which is quite high and does not show the enough accuracy. We then compare them with the MSE value we get from this experiment. The programming of achieving the MSE is shown below.

> mean_squared_error(y_test, fitv)

Output: 7116.324465742683

The MSE between fitted value and the testing TAIEX we achieve in this experiment is approximately *7116*, which is quite low compared to other two method I have tried.

4.2 Correlation Coefficient

We can as well gain the correlation coefficient of fitted values and testing labels to see if they are highly correlated. We use the first output element of function *stats.pearsonr* provided from package *scipy* to get the correlation coefficient. The programming of achieving so is shown below.

> import scipy

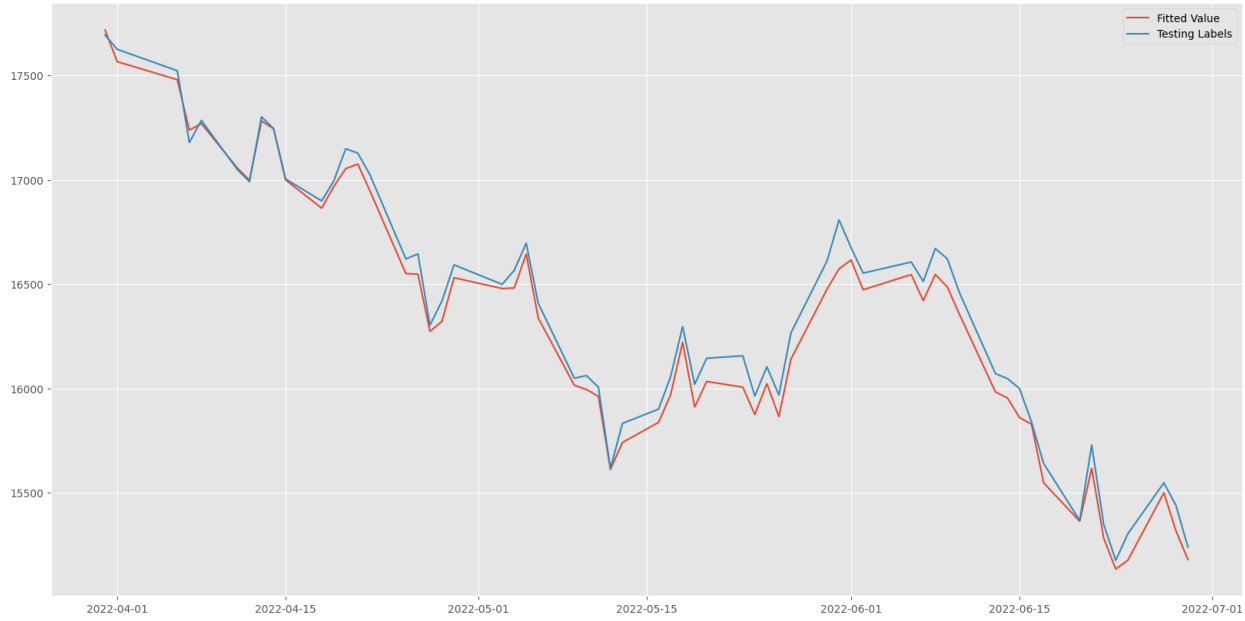
> r,_ = scipy.stats.pearsonr(fitv, y_test)

> print(r)

Output: 0.9972214139772638

The fitted value is approximately 99.72% correlated with the actual testing labels, which we considered as highly correlated. The Figure 2 below shows how fitted values correlated with actual testing labels.

Figure 2 Comparation of Fitted Value and Testing Labels



Chapter 5 Conclusion and Future Direction

In this research, we conduct the numerical experiment of a replicating the TAIEX via best subset selection that is processed via minimizing the out-sample error. We discover that, via this approach, we can effectively approximate the given index we desire to achieve so. However, since all of the optimization technique we utilize rely on the Python package, it is still possible that, with extra steps of gradient decent or other optimization technique, we can achieve the minimal error and thus enhance the accuracy. In short, without noticing the details of the optimization technique inside the Python packages, it is not possible to conclude that "we did our best".

Moreover, for this experiment, we only include fewer amount of funds in to account, which reduce its potential of lowering the error. For example, it is highly possible that with the risk-free rate bond included in the features, the more accurate approximation we can achieve, as well as other funds, such as stocks. As a result, with extra funds, not just specific ETFs included, it is highly possible to achieve the result that is even more decent, though with more features included, it might dramatically increase the calculation cost, as well as higher the trading fee might be charged when the amount of funds gets higher.

Lastly, we did not include the time factor in to account in this experiment. It is highly possible that, with the time factor included, we can achieve even more accurate outcome. For example, utilize the sum of the errors of a total five days instead of daily errors, since the price of a day might be highly correlated with the former days. As a result, it is also valuable if we check if the price of a given day is correlated with prices of former days via ARIMA model.