Project: SP500 Trend Prediction using Random Forest Classifier

Objective:

The objective of this project was to build a machine learning model to predict the daily trend of the S&P 500 stock market index. Specifically, the model aims to predict whether the next day's closing price will be higher or lower than the current day's closing price.

Data:

The historical daily price data for the S&P 500 index was downloaded from Yahoo Finance for the period from 1927 to present. The data includes open, close, high, low and volume information for each trading day.

Data Preprocessing:

A. Establishing a Target variable:

- We are trying to establish a target column where tomorrow's price is greater than today's price so that we will be able to predict the trend of the ETF going forward.
- If tomorrow's price is greater than today's price, then the Target column will show the value as 1 and if tomorrow's price is less than today's price then then it will show the value as 0.

B. Removed the data before 1990 so that the predictions won't be influenced by the data which is from a long time ago as the market conditions would have shifted drastically over these years.

Model Selection:

We are going to choose Random Forest Classifier as our ML model for a few reasons:

A. Random Forest works by training a bunch of individual decision trees with randomized parameters and then averaging the results from those decision trees.
B. Because of this process Random Forest are resistant to overfitting. They can overfit but it is harder for them to overfit compared to the other models.
C. They run relatively quickly.
D. They can pick up nonlinear tendencies in the data. for e.g.: the open price is not linearly correlated with the target. If the open price is higher, it doesn't mean the target is going to be higher.

Model Parameters:

1) n_estimators:

It is the number of individual decision Trees we want to train the higher this is generally the better the accuracy up to a limit. It is a trial-and-error method to run the model quickly. You can set it

lower, but you can manipulate it so that the accuracy is improved. The default number for this estimator is 100 in python .22.

2) min_samples_split:

This helps to protect us against overfitting. Decision trees have a tendency for overfitting. This experiment as well the higher this is the less likely the model will overfit, but the accuracy will be less.

3) random_state:

This model has some randomization built within settings this will help us to generate the same results when we run it again and again.

Initial Prediction:

Then predicted the values for the test set and calculated the precision score and the precision score was 57.9 %

Back Testing the Model:

Building a Back Testing System:

We only tested the model for the last 100 days, now let's test its performance by back testing around multiple years with the data available.

a) Predict Function:

- It basically wraps up all the steps we did above in a function. It takes train, test, predictors and model as its parameters then it fits the model and then it predicts for the test sets and predictions are stored in a variable called preds. preds is numpy array so it is converted to a panda's series and then the preds and test["Target"] are concatenated to a DataFrame called combined and that DataFrame is returned.

b) backtest Function:

- This function takes in the data, our machine learning model, our features (which are our predictors), start is hardcoded to 2500 and step is also hardcoded to 250.
- start value (here it is 2500):
- When you backtest you want to have a certain amount of data to train your model. Every trading year has about 250 days so here we are saying take 10 years of data and train it and try to predict the Target for the next whole year (which is step = 250). So, what happens here is we take the data for the first 10 years of data and predict the Target for the 11th year. Then we take the data for the first 11 years and try to predict the data for the 12th year and repeat this for the whole dataset.

backtest Function:

A. Here we define a empty list as all_predictors.
B. Then we create a for loop to loop across the dataset year by year [where for the range function start = 2500 till the last row of the dataset which is extracted with data.shape[0] and with an increment of step which is 250] .

C. Training set is created, and the test set is created as we defined above.
D. Then it recalls the predict function and stores the predicted values in predictions and then appended to the empty list and then for every loop it is concatenated to the list.

Result After Back Testing:

By doing the back testing for our model we can say that:

- Our model predicted that the sp500 will have a negative trend the next day on 3502 days and on 2596 days tomorrow's closing price will be greater than today's.
- While the market had a negative trend on 2837 days and a positive trend on 3261 days.
- In our back testing our model had a precision score of 52.8 %.

53.4% of the time the market went up and 46.5% of the time the market went down. So, over the period of our dataset our model has performed poorly because the precision score was 52.8 % and the market went up 53.4 % of the time. This means that even if you have traded every day assuming that the market is going to go up you would have been right more than the model's precision score of 52.8 %.

Model Improvement:

Now let's improve the Accuracy of our model:

By Adding additional parameters:

- Let's add a variety of rolling averages and take some ratios and add them to our predictors.
- So, we are looking at the trends of the stock price over certain periods which means that was the closing price today higher than 2 days ago, a week ago, 2 months ago, a year ago and 4 years ago.
- Then we will find the ratio between today's closing price and the closing price of these periods, which will help us to know if the market has gone up a ton because if so, maybe it is due for a downturn and vice versa.

Model Upgrade:

We upgraded the model by changing the parameters n_estimators = 200, min_samples_split = 50 and random_state = 1

Now let's rewrite the predict function sightly:

- We want to have some control over the prediction in terms of what becomes 1 and what becomes a 0 Target value.
- So we will use a predict_proba() function: which returns a probability that a row will be 0 or a 1 (which is our target).
- So, we will take the second column of this which is the probability of the stock price going up meaning the model predicting a Target of 1 and setting a custom threshold over it.

- The Custom Threshold is going to be at 60 % so if the model returns the probability of the price going up the next day more than 60 % then for that day the target will be 1 or else it will be 0.
- This will make the model more confident in its prediction meaning it will reduce the days when the model predicts the price will go up, but it will increase the accuracy of predictions where the prices will go up.

No, we are going to pass in our new _predictors as our features and predict the trend.

Results:

Now because we change the threshold the model is just predicting on 762 days where the price will go up. So, this means that we are going to be trading on just 762 days only.

After back testing with new features and mode the results were:

- This means that on 56.2 % of the time the model has accurately predicted that the price would go up.
- We have improved the accuracy of our model from 52.8 % to 56.2 %.