Group 1: Justin Mannar, Alex Van Der Merwe, Fletcher Mcmeans

CS3820 Semester Project Artifacts

12/2/24

**Source code is in the zip file, but this report will label each block of documentation with its corresponding source code filename. This documentation and all source code files are also on GitHub:** [**https://github.com/jmannar03/CS3820\_MidTerm\_Presentation**](https://github.com/jmannar03/CS3820_MidTerm_Presentation)

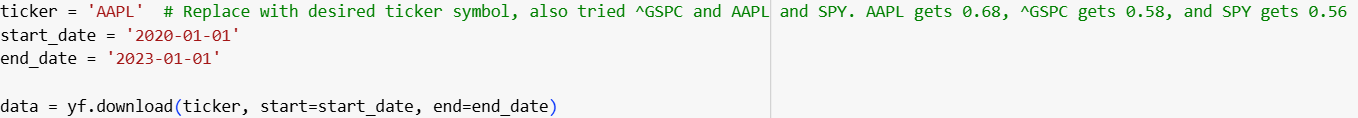
1. **Algorithm\_using\_SMOTE:**

First, we need to install the necessary dependencies, which are A close-up of a white background

Description automatically generated

Pandas and NumPy are used for data handling, while imblearn is used for class imbalance, and scikit-learn is used for Modeling and Evaluation.

We then fetch the data we will be using in our ML model using yfinance in the following code snippet:



One can edit the amount of data brought in by changing the start date and end date variables. Also, note the commented line discussing the 3 different market stocks that our models were trained on. The decimals associated with each stock are the accuracies that we achieved using this model. (^GSPC is the S&P500 index fund, AAPL is Apple, and SPY is the S&P500 ETF stock.



This code snippet shows data manipulation in creating a target column in the dataset to act as an indicator. By using the percent change between days, and whether it was positive or negative, we are able to create binary classification column using 1’s and 0’s.

A close-up of words

Description automatically generated

Here we are creating our feature set and dropping all N/A values so as to clean our dataset from empty rows.

A white background with black and green text

Description automatically generated

Now we are splitting our dataset into training and testing sets, with the testing set being populated by 20% of the full dataset, and the training dataset being given the other 80%.

We then use SMOTE to assist in the class imbalance that we found in other models. It helps by synthesizing new examples for the minority class to balance the dataset.

A close-up of a computer code

Description automatically generated

After balancing the classes, we used the balancedRandomForestclassifier to handle the dataset effectively. n\_estimators is the number of trees in the forest, while max\_depth is the maximum depth of a tree, and min\_samples\_split is minimum amount of samples required to split a node.

A screenshot of a computer program

Description automatically generated

At this point, it is time to test the model. We do this using the first line in this code snippet, and then print out a classification report using the testing dataset and the predictions made against it previously.

After this, we utilized cross validation to check the model’s stability over multiple folds.

1. **Algorithm\_using\_Regression:**

We again begin with downloading the necessary libraries and dependencies, which consist of:

A computer code with black text

Description automatically generated

Pandas and NumPy handle structured data and numerical computations, while sklearn modules create and evaluate the machine learning regression model. We also used matplotlib.pyplot to plot and visualize the results, and yfinance to fetch the historical stock price data.

A computer code with text

Description automatically generated

This function is very important in this model because it calculates on of the key metrics used, the Relative Strength Index (RSI). We first calculate daily price changes using the variable “delta” and segregate the gains/losses. Next, we compute the average gains and losses using rolling average windows. Finally, we are to compute the RSI to indicate overbought/oversold market conditions to use in our model.

A white background with black text

Description automatically generated

Now, like the last Algorithm in this document, we must get our data from yfinance. The user can once again get their data from any market stock, but the documentation will follow the use of the SPY (refer to first algorithm to see what SPY is).

A screenshot of a computer program

Description automatically generated

The “Target” column is a column made up of the next days closing price, so it is the current days closing price shifted down one in the array, and it will be used for predictions later in the Algorithm. We then clean the dataset by dropping unnecessary columns, with an exception if the column we want dropped does not exist. This will not usually happen since most of the data coming from yfinance has the same structure, but if the user ever got a dataset that did not have the columns that our algorithm is trying to drop, the program would not break.

A close-up of a computer code

Description automatically generated

With this feature engineering, “EMA\_50” is an exponential moving average going over 50 days to identify trends. “SMA\_50” is a simple moving average going over 50 days for trend smoothing. The “RSI: is the technical indicator calculated earlier in the described function.

A white screen with red text

Description automatically generated

Here, we are defining the input features as X and the output target as Y. We are then splitting the dataset into training and testing groups, with the training holding 80% and the testing holding the other 20%. While we split up the data, we are also dropping our N/A values before so that we do not have to do deal with blank rows skewing our data.

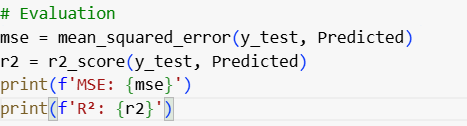
A close-up of a computer code

Description automatically generated

We are now ready to train our model using a RandomForestRegressor, which is an ensemble model that reduces overfitting and improves performance. As before, n\_estimators is the number of trees in the forest, max\_depth is the maximum depth of each tree, and min\_samples\_split is the minimum samples required to split a node.



This line Generates predictions for the test set using the trained model.



Using some of the libraries we imported at the beginning of this algorithm, we can now calculate and print the mean squared error (MSE) and the r2\_score (R^2). The MSE Quantifies the average squared difference between actual and predicted values, while the R^2 Measures the proportion of variance in the target variable explained by the features.

A close-up of a computer code

Description automatically generated

We now can Identify and rank features by their importance in influencing predictions. This can help us to tune our model later by seeing if we were successful in our feature engineering, or if we need to pivot to better our Model.

A screen shot of a computer code

Description automatically generated

Using the actual vs predicted values, we can visualize the models performance by creating a scatterplot and seeing how close the predicted values are to the actual values based off closing price.

1. **Algorithm\_using\_RandomForestClassifier:**

As with the first 2, this algorithm begins with importing the necessary libraries and dependencies:

A close-up of a computer code

Description automatically generated

We can then move into loading and cleaning the dataset by doing:

A screenshot of a computer code

Description automatically generated

This code snippet also adds new columns to the dataset to be used later. “Tomorrow” is just the closing prices shifted down 1 day, to make a column representing the next days closing price. WE then used this column to create the “Target” column which goes to 1 if the closing price goes up and 0 if the closing price goes down.



Here we initialize a new EMA that spans 50 days and will be used as a technical indicator to capture smooth trends in price.

A screenshot of a computer

Description automatically generated

The purpose of “predictors” is to use a list of features to predict the target variable, and the purpose of “train” and “test” is to give all but the last 100 entries to the training dataset, and giving the last 100 to the testing dataset.

A screenshot of a computer program

Description automatically generated

In using this method, we can perform cross validation to identify the best combination of parameters based on precision. The parameters needed are n\_estimators, min\_samples\_split and max\_depth, which have been explained in detail in previous algorithms.

A screenshot of a computer code

Description automatically generated

This snippet of code shows the power of the grid search method. In finding the best permutation of the parameters above, we can now use it to train our model off the “Target” column in our dataset. In usual ML training methods, you must use your best judgment and just plug and play until you find your best parameters. However, this grid search does it for you. One thing that should be said is that this took about 10-15 minutes to run my groups computers, so it is computationally expensive.

A close-up of a test

Description automatically generated

We can now test our model off of the best model that we found and trained above, displaying the precision score and the classification report when done.

A screenshot of a computer program

Description automatically generated

These functions are what our Algorithm uses to both predict and back test our model using the grid search method and RandomForestClassifier. In the “predict” function, we are relying on probability to determine whether the “target” column will be populated with a 1 or a 0 for that day. As in previous run ins with the “target” column, it is basically just seeing whether or not the stocks closing price increased or decreased from the previous day to the current one. This Algorithm uses a slightly different method, as we go solely off of whether or not the probability is >=0.6 or <0.6. The back test function is quite general, as it is basically just looping through the dataset and then training/testing and making predictions based off of it.



This last part simply prints out the classification report and the precision for the back tested predictions. In the case of our model, these were often better than the previous predictions, as they should be, but we did not move forward with this algorithm too much as it took too long to run and it did not return good enough results.

1. **Algorithm\_using\_Confusion\_Matrix:**

Once again, we began with importing the necessary libraries and dependencies:

A computer code with text

Description automatically generated with medium confidence

Pandas and Numpy will be used for data manipulation and numerical computations, and yfinance will be used for its usual fetching of historical data. The sklearn modules RandomforestClassifier, accuracy\_score, precision\_score, recall\_score,f1\_score, and confusion\_matrix will all be used in the algorithm as well as performance metrics and analyzing model predictions. Finally, seaborn and matplotlib will be used to visualize the results

A screenshot of a computer program

Description automatically generated

We then moved forward with calculating the Relative Strength Index (RSI) with the following function. In the function, we first calculate daily price changes, and then separate the gains and losses. Next, we compute the rolling averages of the gains and losses over the specified period (14 days in the context of our algorithm). Finally, we calculate the Relative Strength (RS) using avg\_gain and avg\_loss, ultimately finding the RSI.

A screen shot of a computer code

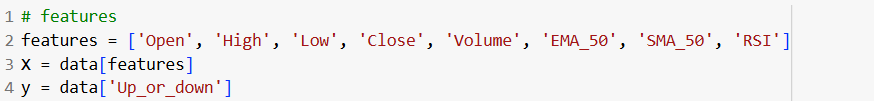
Description automatically generated

Now, we can use this code snippet to download and clean the data that we want. As with the other algorithms, the user can input any stock that they want to keep track of. My group and I followed the ^GSPC, AAPL, and SPY closely in our project, but there are many more out there.

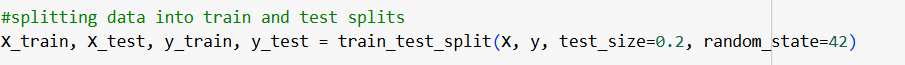
A computer code with text

Description automatically generated with medium confidence

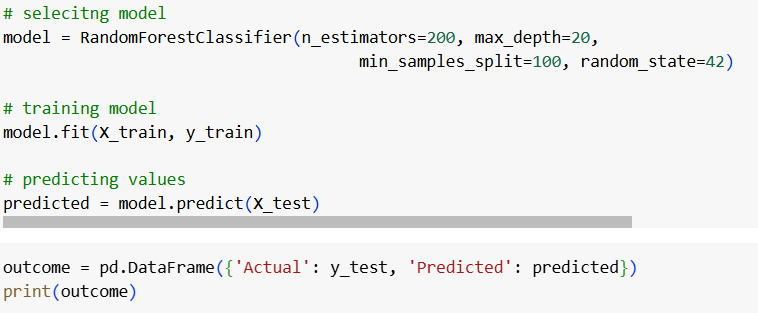
In this code snippet, we again utilized both EMA’s and SMA’s spanning 50 days to capture short-term trends. We also incorporated the calculated RSI from earlier to indicate stock momentum and overbought/oversold conditions. Finally, we drop N/A values to ensure clean data for modeling.



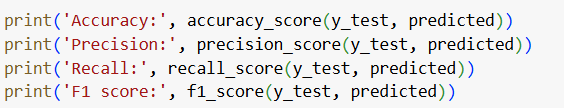
In this code snippet, my group and I created the features for our model, setting X to the features, and setting Y to a target variable indicating upward/downward price movement



We then split the dataset, as usual, giving 80% to the training dataset and 20% to the testing dataset.



With this code snippet, the model was made into a RandomForestClassifier and was trained and tested on the previously made datasets. We then have a simple print statement showing the outcome of predicted values vs the actual values.



Now that we have our results displayed, it was time to print the classification metrics, as seen above.



The final code snippet involved creating the confusion matrix and plotting it using matplotlib and seaborn.