Stock Trading Algorithm

Documentation

Xing Yi Liu

Last Updated October 31, 2020

Contents

1 Method

The aim, for each stock, is to use a neural network to incorporate various factors that may affect the next day's stock price. The factors included are:

- Jensen's alpha
- Price today as predicted by Dividend Discount Model

1.1 Jensen's Alpha

Jensen's alpha, α, is given by:

$$\alpha = \text{Actual Rate} - \text{CAPM Rate}$$

, where Actual Rate is the actual daily return of the stock, and CAPM Rate is the rate of return as determined by the Capital Asset Pricing Model.

$$\text{CAPM Rate} = R_{\text{free}} + \beta(\bar{R}_{\text{mkt}} - R_{\text{free}}).$$

Here, R_{free} is the risk-free return, β is the stock's beta, and $\bar{R}_{\text{mkt}}$ is the market return. For further details, see function documentation below.

1.2 Dividend Discount Model

$$P_0 = \frac{\text{DIV}_1}{i_e - g}$$

, where P_0 is the price of the stock today, DIV_1 is the expected dividend 1 year from now, i_e is the total yield, and g is the dividend growth rate.

i_e is computed by the Capital Asset Pricing Model.

$$i_e = R_{\text{free}} + \beta(\bar{R}_{\text{mkt}} - R_{\text{free}})$$

, where R_{free} is obtained by using the `quantmod` package in R to retrieve the U.S. 10-year treasury yield.

2 Future Changes

The calculation of CAPM rate yields many negative values, which in turn affects the calculation of Jensen's alpha and DDM price. This will be patched.

3 Code Documentation

3.1 Global Parameters

- `startDate`: The date of the first data point used to train neural networks

3.2 `nn_train.R`

3.2.1 `attachAlpha`

Description: Takes a list as outputted by `attachCAPMRate` and adds a column with stock's Jensen's alpha over time.

Arguments:

- `stocks_info`: a list of stocks as outputted by `attachCAPMRate`, where each component is a data frame with date, price, return, decision, beta, and CAPM rate

Details: Jensen's alpha is computed by subtracting the actual return by the CAPM rate.

$$\alpha = \text{Actual Rate} - \text{CAPM Rate}$$

, where Actual Rate is given by the variable `return` within each stock's data frame.

Value: `stocks_info`, but with each stock's data frame component of the lsit having an added (Jensen's) alpha variable.

3.2.2 `attachBeta`

Description: Takes a list as outputted by `getStockPrices` and adds a column with stock's β over time.

Arguments:

- `stocks_info`: a list of stocks as outputted by `getStockInfo`, where each component is a data frame with date, price, and return
- `mkt_info`: list as outputted by `getStockInfo`, with the first component being the data frame that represents market information (e.g. S&P 500 index). The number of rows in this data frame must be the same as that of the data frames in `stocks_info`, and it is assumed that corresponding rows in `stocks_info`'s data frames and `mkt_info`'s data frames have the same dates. Components of `mkt_info` other than the first are not used.

Details: In general, β is calculated using one year of historical, daily data. For the first year of observations, all the preceding data points are used to calculate β, but for observations thereafter, only the previous 350 observations are used. With the lack of data on some days, these 350 previous observations are approximately one year of historical, recent data.

β is given by

$$\beta = \frac{\text{Cov}_{i,\text{mkt}}}{\sigma^2_{\text{mkt}}}$$

, where $\text{Cov}_{i,\text{mkt}}$ is the covariance (`cov()`) of daily returns between stock i and the market (e.g. S&P 500 returns), and σ^2_{mkt} is the variance (`var()`) of daily market (S&P 500) returns.

Value: `stocks_info`, but with each stock's data frame component of the list having an added `beta` variable.

3.2.3 `attachCAPMRate`

Description: Takes a list as outputted by `attachBeta` and adds a column with stock's CAPM rate i_e over time.

Arguments:

- `stocks_info`: a list of stocks as outputted by `getStockInfo`, where each component is a data frame with date, price, return, and beta
- `mkt_info`: list as outputted by `getStockInfo`, with the first component being the data frame that represents market information (e.g. S&P 500 index), and the second component being the data frame that represents information on the risk-free asset in CAPM (e.g. 10-year treasury bill). List components other than the first and second are not used.

Details: The new variable, `CAPM.rate`, is computed by the Capital Asset Pricing Model at each date.

$$i_e = R_{\text{free}} + \beta(\bar{R}_{\text{mkt}} - R_{\text{free}})$$

Value: `stock_info`, but with each stock's data frame component of the list having an added `CAPM.rate` variable (i_e).

3.2.4 `attachDecision`

Description: Takes a list of stocks (data frames) and adds a column to each component with the correct trade decision on that day. If the next day's stock price became lower than the current day's, the correct decision is `"sell"`. If the next day's stock price became higher than the current day's, the correct decision is `"buy"`.

Arguments:

- `stocks_info`: A list of stocks, where each component is a data frame, and these data frames have a `price` column

Value: `stocks_info`, except each list component now has an added `decision` column.

3.2.5 `attachDividendGrowth`

Description: Takes a list of stocks (data frames) and adds a column to each component representing the annual dividend growth rate.

Arguments:

- `stocks_info`: A list of stocks as outputted by `calculateExpDiv1Yr`, where each component is a data frame

Details: For any date `d`, annual dividend growth rate is computed by calculating the net growth rate between the value of `exp_div_1yr` on `d` and the value of `exp_div_1yr` one year before `d`. If `exp_div_1yr` is not available one year before `d`, the result is `NA`.

Value: `stocks_info`, except each stock's data frame component of the list now has an added `div_growth` column.

3.2.6 `attachDividends`

Description: Takes a list of stocks (data frames) and adds a column to each component representing the amount of dividends paid on a particular date.

Arguments:

- `stocks_info`: A list of stocks, where each component is a data frame

Details: If a particular dividend payment date is not found as a row in `stocks_info`, this function will return an error.

Value: `stocks_info`, except each list component now has an added `dividend` column.

3.2.7 `attachDDM`

Description: Takes a list of stocks (data frames) and adds a column to each component representing the price of the stock predicted by the Dividend Discount Model.

Arguments:

- `stocks_info`: A list of stocks, where each component is a data frame

Value: `stocks_info`, except each list component now has an added `DDM_price` column.

3.2.8 `calculateExpDiv1Yr`

Description: Takes a list of stocks (data frames) with dividend information and adds a column to each component, which represents the expected dividend paid by the stock over the next year.

Arguments:

- `stocks_info`: A list of stocks, where each component is a data frame with a `dividend` column (e.g. as outputted by `attachDividends`)

Details: First, "expected dividend in one year" on a particular date means how much dividend we expect that stock to pay (in total) between today and one year from now, inclusive.

Expected dividend in one year is calculated by adding the sum of divideneds paid in the last 365 days, inclusive. This number of days covers all dividend payment dates in the past year, but for a few dates of the year, this estimation may overcount.

Here is an example of why using the last 365 days could be slightly problematic. Apple paid a dividend on August 7, 2020. For the variable "expected dividend in 1 year" on August 8, 2020, using the last 365 days would mean that we sum the dividend payments which occurred on:

- August 7, 2020
- May 8, 2020
- February 7, 2020
- November 7, 2019
- August 9, 2019

Notice how Apple pays quarterly dividends, yet we counted the August payment twice in estimating the expected dividend in one year. However, after much consideration, summing dividends in the last 365 days to estimate the expected dividend in one year was the best solution. For example, only summing dividends in the last 360 days may also be problematic, because for certain dates this may undercount expected dividend in one year by an entire quarter's amount. It was decided that overcounting was less risky than undercounting, because if a company sticks to schedule in making dividend payments, overcounting the amount on a few dates may be considered as adding a "bonus score" for their payment consistency.

Also, summing the last 365 days to estimate expected dividend in one year avoids any riskier problems which arise from irregular dividend payment schedules. One may consider providing the algorithm with the knowledge that Apple pays *quarterly* dividends, and so we should only sum the previous four dividend payments on each date. However, this strategy collapses when dealing with NYSE: MAIN, for example. Take a look at their highly irregular payment schedule, especially in their first few years of payments. They switch from quarterly to monthly payments, and sometimes even pays sporadically.

Thus, for any date `d`, the value of the "expected dividend in 1 year" variable for `d` is estimated by summing all dividend payment amounts in the 365 days prior to and including `d`.

Value: `stocks_info`, except each list component now has an added `exp_div_1yr` column.

3.2.9 `getStockInfo`

Description: `getStockInfo` obtains one or more stocks' information.

Arguments:

- `stocks`: a vector of strings containing stock symbols

Value: A list where each component represents and is named as a stock. Each component is a data frame with the following variables:

- Date
- Price
- Daily return

3.2.10 `getStockPrices`

Description: `getStockPrices` obtains one or more stocks' prices from `library(quantmod)`.

Arguments:

- `stocks`: a vector of strings containing stock symbols

- `commonDates`: logical value to indicate if only dates for which all stocks have data should be kept

Value: A list where each component represents and is named as a stock. Each component is a data frame, where the first variable contains dates, and the second variable contains the stock price on that date.

3.2.11 `removeNA`

Description: Removes rows where any of the variables `NN_vars` have NA values. The variables `NN_vars` can be seen in the Global Parameters section. The purpose is to prepare the list of stocks for neural network training.

Arguments:

- `stocks_info`: list of stocks as intended to train neural network, where each component is a data frame

Value: `stocks_info`, but with each stock's data frame component of the list having certain rows removed. The rows removed are ones with `NA` in at least one of the `NN_vars` variables.

3.2.12 `returns`

Description: `returns` calculates the net growth rate (or in the case of stock prices, returns) of successive elements in a numeric variable.

Arguments:

- `x`: the numeric variable containing stock prices for which to calculate returns

Value: A vector of the same length as `x`, with each element containing the net growth rate from the previous index of `x`. The first element is `NA`.

3.3 `nn.predict.R`

3.3.1 `makePredictions`

Description: Predicts a trade decision based on previously trained neural network and new information provided

Arguments:

- `networks`: list of neural networks as resulted from `nn_train.R`, where each component is a neural network that corresponds to a stock. Each neural network is used to predict for that stock.
- `stocks_info_predict`: list of stocks, where each component is a data frame corresponding to a stock. This data frame must have the variables: `date`, `alpha`, and `DDM_price`.

Value: A list where each component is a data frame representing a stock. All the dates that were in `stocks_info_predict` have been predicted. The first variable in each data frame is the dates. The second, third, and fourth variables are predicted probabilities of a correct "buy", "neutral", and "sell" decision, repectively. The fifth variable states the decision which had the maximum predicted probability on that date, which should be interpreted as the **neural network's predicted trade decision**.