Q Learning for Stocks

Anusha Kumar, Rophence Ojiambo

We code a Q-learning algorithm for predicting closing prices.

Q-Learning Algorithm

The code is implementing the Q-learning algorithm to create a simple trading strategy for stocks. The Q-learning algorithm is a model-free reinforcement learning technique used to learn an optimal policy based on trial and error. It involves building a Q-table that maps the current state and action to the expected future reward. In this case, the state is the current stock price, and the action is either to buy or hold/sell.

The function 'Q_learning' takes a data frame of stock prices and applies the Q-learning algorithm to each stock to create a Q-matrix that stores the expected future rewards for each state-action pair. It also creates a list of Boolean Q-tables for each stock, which specify whether to buy or hold/sell at each time step based on the Q-matrix. Finally, it returns the Q-matrix, actions taken, and Q-table for each stock.

The code then loads stock data for four companies "AAPL", "AMZN", "JNJ", "NFLX", extracts the adjusted prices, and combines them into a matrix. It performs exploratory data analysis (EDA) by plotting the distribution of prices for each stock. It then applies the 'Q_learning' function to each stock to create a list of Q-matrices, actions taken, and Q-tables.

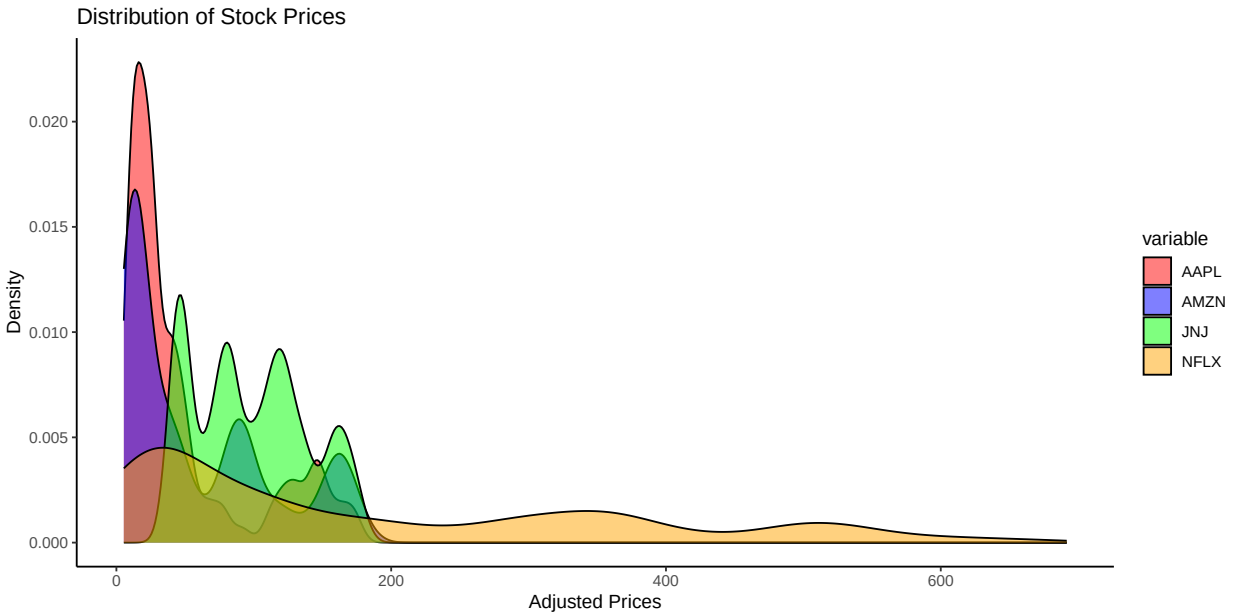
The 'Q_learning_summary' function generates a summary table of the actions taken for each stock, converts the action codes to action names, and splits the summary table by stock. It then creates tables for action for each stock and access the Q matrix for each stock. The 'predict_prices' function predicts the prices of a stock using the Q matrix and the current prices of the stock.

Summary statistics

From the table we observe that the prices for AAPL were highly skewed compared to prices from AMZN, JNJ and NFLX with skewness 1.332973, 0.852902, 0.233780, 0.850237 respectively.

	AAPL	AMZN	JNJ	NFLX
nobs	3.272000e+03	3272.000000	3272.000000	3272.000000
Minimum	5.837758e+00	5.430500	39.409145	7.018571
Maximum	1.806839e+02	186.570496	181.108795	691.690002
1Quartile	1.660096e+01	13.327125	62.507419	34.668928
3Quartile	5.486968e+01	93.131001	126.193081	327.874992
Mean	4.939430e+01	58.912957	97.960695	188.284766
Median	2.734463e+01	36.393750	96.012989	110.114998
Sum	1.616182e+05	192763.194847	320527.394296	616067.753871
SE Mean	8.347890e-01	0.945406	0.708342	3.126838
LCL Mean	4.775754e+01	57.059309	96.571857	182.154008
UCL Mean	5.103106e+01	60.766604	99.349533	194.415524
Variance	2.280165e+03	2924.489098	1641.719601	31990.720302
Stdev	4.775107e+01	54.078546	40.518139	178.859499
Skewness	1.332973e+00	0.852902	0.233780	0.850237
Kurtosis	3.709760e-01	-0.625129	-1.084334	-0.480602

	Stock			
Action Taken	AAPL	AMZN	JNJ	NFLX
Buy	2958	2958	2958	2958
Sell	157	157	157	157
Hold	156	156	156	156



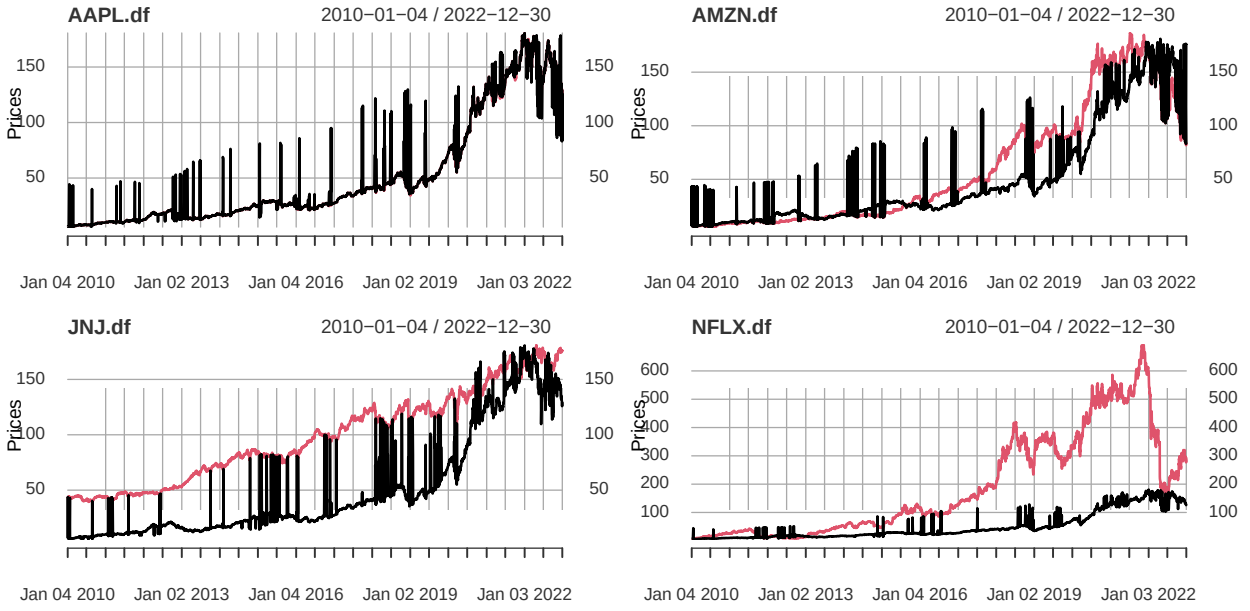
Summary tables for action taken

The table shows the frequency of each action ("Buy", "Sell", "Hold") taken for each stock. For instance, for the stock "AAPL", the algorithm took the action "Buy" 2958 times, "Sell" 157 times, and "Hold" 156 times during the simulation. The same information is presented for the other three stocks ("AMZN", "JNJ", and "NFLX").

	AAPL	AMZN	JNJ	NFLX
MSE	161.2266	1006.65	3317.487	91061.72

Plot for predicted closing prices and Adjusted prices

The plots shows that the algorithm was able to predict closing prices with significant accuracy since the predicted closing prices are not deviating too much from the adjusted prices.



Performance Check

The table above shows the performance of the predictions using mean squared error for four stocks (AAPL, AMZN, JNJ, and NFLX). For each stock, the code first predicts the prices using the Q matrices trained on the training data (2010 to 2016) and then calculates the mean squared error between the predicted prices and the actual prices in the testing data (2017 to 2022).

The mean squared error for AAPL is 161.2266, for AMZN it is 1006.65, for JNJ it is 3317.487, and for NFLX it is 91061.72. A lower mean squared error indicates better performance, therefore the predictions for AAPL are the most accurate among the four stocks.

Appendix

```r
knitr::opts_chunk$set(echo = TRUE)
# Load the required libraries
library(quantmod)
library(fBasics)
library(ReinforcementLearning)
library(ggplot2)
library(gridExtra)
library(reshape2)
library(knitr)
library(dplyr)
library(cowplot)
#Q_learning function
Q_learning <- function(prices) {
  # Define the initial values of the Q-matrix and the learning parameters
  n <- nrow(prices)
  n_stocks <- ncol(prices)
  Q_list <- vector("list", n_stocks)
  for (j in 1:n_stocks) {
    Q_list[[j]] <- matrix(0, nrow = n, ncol = 3) # Q-matrix for stock j with 3 actions
  }
  alpha <- 0.2 # Learning rate
  gamma <- 0.9 # Discount factor

  # Initialize an empty vector to store the actions taken
  actions_taken <- vector("numeric", n-1)

  # Loop through each time step
  for (i in 2:n) {
    # Loop through each stock
    for (j in 1:n_stocks) {
      # Define the state at time t-1
      state <- prices[i-1, j]

      # Select the action with the highest Q-value
      q_values <- Q_list[[j]][i-1, ]
      action <- sample(which(q_values == max(q_values)), 1)

      # Store the action taken at time t-1
      actions_taken[i-1] <- action

      # Define the reward and the next state at time t
      reward <- prices[i, j] - prices[i-1, j] # The reward is the change in the price of stock j
      next_state <- prices[i, j]

      # Update the Q-matrix using the Q-learning algorithm
      Q_list[[j]][i, action] <- Q_list[[j]][i-1, action] + alpha * (reward + gamma * max(Q_list[[j]][i-
    }
  }

  # Create a list of Q-tables for each stock
  Q_table <- lapply(Q_list, function(x) {
```

```
    buy_action <- x[, 1] > x[, 2] & x[, 1] > x[, 3] # Buy if Q-value of buy is greater than Q-value of
    sell_action <- x[, 2] > x[, 1] & x[, 2] > x[, 3] # Sell if Q-value of sell is greater than Q-value
    hold_action <- !buy_action & !sell_action # Hold if none of the above is true
    data.frame(Buy = buy_action, Sell = sell_action, Hold = hold_action)
  })

  # Return the list of Q-matrices (one for each stock), actions taken, and Q-tables
  return(list(Q_list = Q_list, actions_taken = actions_taken, Q_table = Q_table))
}
# Load the stock data
data <- getSymbols(c("AAPL", "AMZN", "JNJ", "NFLX"), src = "yahoo", from = "2010-01-01", to = "2022-12-
# Extract adjusted prices
AAPL_prices <- Ad(AAPL)
AMZN_prices <- Ad(AMZN)
JNJ_prices <- Ad(JNJ)
NFLX_prices <- Ad(NFLX)
# Combine the stock price vectors into a matrix
prices <- cbind(AAPL_prices, AMZN_prices, JNJ_prices,NFLX_prices)
colnames(prices) <- gsub("\\..*", "", colnames(prices))
prices <- na.omit(prices)
prices_df <- as.data.frame(prices)
# generate summary statistics for the prices data
summary_prices <- as.data.frame(basicStats(prices_df))
# create the table using kable
results <- kable(summary_prices, format = "latex", align = "c")

# print the table using kable
cat(results)
# Load the stock data
library(quantmod)
data <- getSymbols(c("AAPL", "AMZN", "JNJ", "NFLX"), src = "yahoo", from = "2010-01-01", to = "2022-12-
# Extract adjusted prices
AAPL_prices <- Ad(AAPL)
AMZN_prices <- Ad(AMZN)
JNJ_prices <- Ad(JNJ)
NFLX_prices <- Ad(NFLX)
# Combine the stock price vectors into a matrix
prices <- cbind(AAPL_prices, AMZN_prices, JNJ_prices,NFLX_prices)
colnames(prices) <- gsub("\\..*", "", colnames(prices))
prices <- na.omit(prices)
prices_df <- as.data.frame(prices)
library(ggplot2)
library(reshape2)
# Plot the distribution of prices for each stock
ggplot(melt(prices_df), aes(value, fill = variable)) +
  geom_density(alpha = 0.5) +
  scale_fill_manual(values = c("red", "blue", "green", "orange")) +
  labs(x = "Adjusted Prices", y = "Density", title = "Distribution of Stock Prices")+theme_classic()
library(kableExtra)
#Q_learning function
Q_learning <- function(prices) {
  # Define the initial values of the Q-matrix and the learning parameters
  n <- nrow(prices)
```

```r
  n_stocks <- ncol(prices)
  Q_list <- vector("list", n_stocks)
  for (j in 1:n_stocks) {
    Q_list[[j]] <- matrix(0, nrow = n, ncol = 3) # Q-matrix for stock j with 3 actions
  }
  alpha <- 0.2 # Learning rate
  gamma <- 0.9 # Discount factor

  # Initialize an empty vector to store the actions taken
  actions_taken <- vector("numeric", n-1)

  # Loop through each time step
  for (i in 2:n) {
    # Loop through each stock
    for (j in 1:n_stocks) {
      # Define the state at time t-1
      state <- prices[i-1, j]

      # Select the action with the highest Q-value
      q_values <- Q_list[[j]][i-1, ]
      action <- sample(which(q_values == max(q_values)), 1)

      # Store the action taken at time t-1
      actions_taken[i-1] <- action

      # Define the reward and the next state at time t
      reward <- prices[i, j] - prices[i-1, j] # The reward is the change in the price of stock j
      next_state <- prices[i, j]

      # Update the Q-matrix using the Q-learning algorithm
      Q_list[[j]][i, action] <- Q_list[[j]][i-1, action] + alpha * (reward + gamma * max(Q_list[[j]][i-
    }
  }

  # Create a list of Q-tables for each stock
  Q_table <- lapply(Q_list, function(x) {
    buy_action <- x[, 1] > x[, 2] & x[, 1] > x[, 3] # Buy if Q-value of buy is greater than Q-value of
    sell_action <- x[, 2] > x[, 1] & x[, 2] > x[, 3] # Sell if Q-value of sell is greater than Q-value
    hold_action <- !buy_action & !sell_action # Hold if none of the above is true
    data.frame(Buy = buy_action, Sell = sell_action, Hold = hold_action)
  })

  # Return the list of Q-matrices (one for each stock), actions taken, and Q-tables
  return(list(Q_list = Q_list, actions_taken = actions_taken, Q_table = Q_table))
}
# Apply the Q-learning function to each stock
Q_list <- Q_learning(prices_df)
#================================================
#presentation of actions taken
#================================================
# Generate summary Action table
Q_learning_summary <- function(prices) {
  # Define the names of the actions
```

```r
  action_names <- c("Buy","Sell","Hold")

  # Run the Q_learning function and extract the actions taken
  Q_result <- Q_learning(prices)
  actions_taken <- Q_result$actions_taken

  # Convert the action codes to action names
  actions_taken <- factor(actions_taken, levels = 1:3, labels = action_names)

  # Create a summary table of the actions taken for each stock
  summary_table <- data.frame(
    Stock = rep(colnames(prices), each = nrow(prices)-1),
    Action_Taken = actions_taken,
    stringsAsFactors = FALSE
  )

  # Return the summary table
  return(summary_table)
}
summary_table <- Q_learning_summary(prices_df)
summary_table

# Split summary_table by stock
stock_summary <- split(summary_table, summary_table$Stock)

#tables for action for each stock
table(stock_summary$AAPL)
table(stock_summary$AMZN)
table(stock_summary$JNJ)
table(stock_summary$NFLX)

results1 <- kable(table(stock_summary$AAPL), format = "latex", align = "c")
results2 <- kable(table(stock_summary$AMZN), format = "latex", align = "c")
results3 <- kable(table(stock_summary$JNJ), format = "latex", align = "c")
results4 <- kable(table(stock_summary$NFLX), format = "latex", align = "c")

# print the table using kable
cat(results1)
cat(results2)
cat(results3)
cat(results4)
predict_prices <- function(Q, stock_prices) {
  # Define the initial values of the state and the predicted prices vector
  state <- stock_prices[1, ]
  predicted_prices <- numeric(nrow(stock_prices))

  # Loop through each time step
  for (i in 1:nrow(stock_prices)) {
    # Define the action as the one with the highest Q-value for the current state
    action <- which.max(Q[i, ])

    # Define the predicted price as the price of the chosen action
    predicted_prices[i] <- stock_prices[i, action]
```

```
    # Update the state to the next state based on the chosen action
    state <- stock_prices[i, ]
  }

  # Return the predicted prices vector
  return(predicted_prices)
}
# Predict the closing prices for each stock using the corresponding Q-matrix
AAPL_predicted_prices <- predict_prices(Q_list$Q_list[[1]], prices_df)
AMZN_predicted_prices <- predict_prices(Q_list$Q_list[[2]], prices_df)
JNJ_predicted_prices <- predict_prices(Q_list$Q_list[[3]], prices_df)
NFLX_predicted_prices <- predict_prices(Q_list$Q_list[[4]], prices_df)
#Plots for the predicted closing and Adjusted prices
par(mfrow = c(2, 2))

# AAPL Plot
AAPL.df <- cbind(AAPL_predicted_prices, AAPL_prices[,1])
names(AAPL.df)[2] <- "Adjusted Prices"
names(AAPL.df)[1] <- "Closing_Price"

plot(AAPL.df, main = "AAPL.df", xlab = "Date", ylab = "Prices")

# AMZN Plot
AMZN.df <- cbind(AMZN_predicted_prices, AMZN_prices[,1])
names(AMZN.df)[1] <- "Close"
names(AMZN.df)[2] <- "Adjusted"

plot(AMZN.df, main = "AMZN.df", xlab = "Date", ylab = "Prices")

# JNJ Plot
JNJ.df <- cbind(JNJ_predicted_prices, JNJ_prices[,1])
names(JNJ.df)[1] <- "Close"
names(JNJ.df)[2] <- "Adjusted"

plot(JNJ.df, main = "JNJ.df", xlab = "Date", ylab = "Prices")

# NFLX Plot
NFLX.df <- cbind(NFLX_predicted_prices, NFLX_prices[,1])
names(NFLX.df)[1] <- "Close"
names(NFLX.df)[2] <- "Adjusted"

plot(NFLX.df, main = "NFLX.df", xlab = "Date", ylab = "Prices")
#=================================================================
#Train/test
#=================================================================
# Split data into training and testing sets
# Convert rownames to a column 'Date'
prices_df$Date <- rownames(prices_df)
# Convert 'Date' column to Date format
prices_df$Date <- as.Date(prices_df$Date)
# Subset the data for the training set (2010-2016)
train_prices <- subset(prices_df, Date >= as.Date("2010-01-04") & Date <= as.Date("2016-12-31"))
```

```r
# Subset the data for the test set (2017-2022)
test_prices <- subset(prices_df, Date >= as.Date("2017-01-01") & Date <= as.Date("2022-05-07"))

# Apply the Q-learning function to each stock using the training data
Q_list_train <- Q_learning(train_prices)

# Access Q matrix for each stock
AAPL_Q_train <- Q_list_train$Q_list[[1]]
AMZN_Q_train <- Q_list_train$Q_list[[2]]
JNJ_Q_train <- Q_list_train$Q_list[[3]]
NFLX_Q_train <- Q_list_train$Q_list[[4]]

# Use the Q matrices to predict prices for each stock using the testing data
AAPL_predicted_prices <- predict_prices(AAPL_Q_train, test_prices)
AMZN_predicted_prices <- predict_prices(AMZN_Q_train, test_prices)
JNJ_predicted_prices <- predict_prices(JNJ_Q_train, test_prices)
NFLX_predicted_prices <- predict_prices(NFLX_Q_train, test_prices)

# Evaluate the performance of the predictions using mean squared error
AAPL_mse <- mean((AAPL_predicted_prices - test_prices[, "AAPL"])^2)
AMZN_mse <- mean((AMZN_predicted_prices - test_prices[, "AMZN"])^2)
JNJ_mse <- mean((JNJ_predicted_prices - test_prices[, "JNJ"])^2)
NFLX_mse <- mean((NFLX_predicted_prices - test_prices[, "NFLX"])^2)

# Print the mean squared error for each stock
cat("AAPL MSE:", AAPL_mse, "\n")
cat("AMZN MSE:", AMZN_mse, "\n")
cat("JNJ MSE:", JNJ_mse, "\n")
cat("NFLX MSE:", NFLX_mse, "\n")
#================================================================
```