CS971: AI for Finance Assignment 2

Stewart Macfarlane, Vladimir Lenkov, Alvee Kabir

11-04-2025

Background and Project Overview

Background and Description of the Problem

The goal of this project is to build a trading system that leverages advanced machine learning techniques to forecast asset prices and execute trading decisions. The system first selects an optimal asset from the S&P 500 by evaluating risk-adjusted historical performance using daily returns and Sharpe ratios. Once the asset is chosen, its price data are pre-processed with technical indicators such as the RSI, MACD, and volume moving averages to capture market dynamics. An LSTM neural network which is well-known for its ability to model temporal dependencies is then employed to predict next-day prices. The model's hyperparameters are then finely tuned using both grid search and genetic algorithms. Finally, trading rules are applied to convert predictions (alone or in combination with RSI signals) into buy, sell, or hold actions in a simulated trading environment.

Related Work

Recent work on machine learning-based trading strategies spans deep neural models, technical analysis, and evolutionary optimization. Recurrent architectures like LSTM networks have been widely applied to stock price prediction and trading signal generation, leveraging their ability to capture temporal patterns and often outperforming traditional statistical models [1]. Many studies enhance such models by incorporating popular technical indicators such as RSI and MACD as input features, effectively fusing signals with data-driven learning to improve predictive accuracy [2]. In addition, optimization techniques like genetic algorithms have been used to fine-tune both model hyperparameters and strategy parameters. For example, GAs optimizing LSTM settings have achieved better forecasting performance than untuned benchmarks and similarly have been applied to calibrate indicator-based trading rules to maximize metrics like the Sharpe ratio [1]. These combined approaches demonstrate that integrating LSTM-driven prediction with technical indicators and applying evolutionary optimization can yield more robust, profitable trading strategies in practice which is precisely what our project aims to do.

Asset Selection

The initial assets were gathered using the S&P 500 index, a stock market index that tracks the performance of 500 of the largest trading companies in the United States. In addition to having an extensive collection of assets, this index represents a wide range of sectors including but not limited to technology, healthcare and finance. This serves as a solid foundation for selecting a significant asset for the project.

```r
assets <- tq_index("SP500") # Load 500 assets from S&P
```

Furthermore, the daily returns for each asset are retrieved to calculate the Sharpe ratio.

```
load_daily_returns <- function(asset_symbols, startDate, endDate) { removed_assets <- c()
  assets_train <- lapply(asset_symbols, function(sym) {
    tryCatch(dailyReturn(getSymbols(sym, from = startDate, to = endDate, auto.assign = FALSE)),
      error = function(e) {removed_assets <<- append(removed_assets, sym); NULL})})
  asset_symbols <- setdiff(asset_symbols, removed_assets)
  df <- setNames(do.call(merge, c(assets_train, all = T)), asset_symbols)
  df <- na.omit(df); df <- df[, colSums(is.na(df)) < nrow(df)]; return(df)}
```

The start and end date for the period to be used to make next-day predictions has been set to two months. This is so that enough data is present to reflect vital patterns to make predictions, however, not a long enough time period whereby the large quantity of historic data will negatively skew results.

```
asset_symbols <- assets$symbol; startDate <- "2024-08-01"; endDate <- "2024-12-31"
df <- load_daily_returns(asset_symbols, startDate, endDate)
```

```
calc_sharpe_ratio <- function(returns, rf_rate) {mean_return <- mean(returns); risk <- sd(returns)
  sharpe_ratio <- ((mean_return - rf_rate) / risk) * sqrt(252); return(sharpe_ratio)}
```

The performance of all 500 assets is evaluated and compared to one another based on their Sharpe ratios. The Sharpe ratio serves as a valuable tool for measuring investment prospects for a specific asset as it enables the comparison of the expected return for the level of risk being taken (risk-adjusted return). In this case, a risk-free rate is dynamically retrieved and used within the Sharpe ratio calculation for each asset.

$$S_a = \frac{E[R_a - R_b]}{\sigma_a}$$

$$Where: S_a = \text{Sharpe Ratio } E = \text{Expected Return}$$

$$R_a = \text{Asset Return } R_b = \text{Risk Free Rate } \sigma_a = \text{Asset Risk}$$

```
rf_rate <- as.numeric(last(getSymbols("DGS3MO", src = "FRED", auto.assign = FALSE)))/100 /252
best_res <- calc_sharpe_ratio(df[, 1], rf_rate); best_asset <- NULL
for (col in colnames(df)) { curr_sharpe <- calc_sharpe_ratio(df[, col], rf_rate)
  if (curr_sharpe > best_res) { best_res <- curr_sharpe; best_asset <- col}}
```

Once all assets have been compared, the best-performing asset is selected to be used to make next-day predictions in alignment with a comprehensive trading rule. All relevant data is then retrieved, this includes opening, high, low and closing prices.

```
best_asset_data <- getSymbols(best_asset, from = startDate, to = endDate, auto.assign = FALSE)
```

Data Preprocessing

Before training the LSTM-based model, we first enrich our data with technical indicators (RSI, MACD, and others), then remove any missing values and normalize each feature. Normalization helps ensure that the ranges of different variables do not negatively impact model training. Afterwards, we structure the data as sequences for the network by selecting the features of interest, choosing an appropriate sequence length and splitting into training and test sets.

We then add these new indicators as columns in our main dataset and remove any rows with missing values.

```r
best_asset_data$RSI = rsi; best_asset_data$MACD = macd
best_asset_data$Volume_MA = volume_ma; best_asset_data = na.omit(best_asset_data)
```

Next, we normalize each column to the range [0,1] using a simple min-max scaling function to help the model converge more reliably during training.

```r
data <- data.frame(best_asset_data[,1:9])
min_max_normalize <- function(x) {(x - min(x)) / (max(x) - min(x))}
data_scaled <- as.data.frame(lapply(data, min_max_normalize))
```

We now define a custom splitting function for time-series data. The idea is to convert our continuous dataset into overlapping sequences of length `seq_length`.

```r
train_test_split <- function(asset, seq_length, target_feature, test_size = 0.2) {
  asset_matrix <- as.matrix(asset)
  num_seq <- nrow(asset_matrix) - seq_length + 1; num_features <- ncol(asset_matrix)
  seq_data <- array(dim = c(num_seq, seq_length, num_features))
  for (index in 1:(nrow(asset_matrix) - seq_length +1)) {
    seq_data[index, , ] <- asset_matrix[index:(index + seq_length - 1), ]}
  test_set_size <- round(test_size * nrow(seq_data)); train_set_size <- nrow(seq_data) - test_set_size
  x_train <- seq_data[1:train_set_size, 1:(seq_length - 1), , drop = FALSE]
  y_train <- seq_data[1:train_set_size, seq_length, target_feature, drop = FALSE]
  x_test  <- seq_data[(train_set_size + 1):nrow(seq_data), 1:(seq_length - 1), , drop = FALSE]
  y_test  <- seq_data[(train_set_size + 1):nrow(seq_data), seq_length, target_feature, drop = FALSE]
  return(list(x_train = x_train,y_train = y_train,x_test = x_test,y_test = y_test))}
```

With all preprocessing steps established, we can now select the columns to include and specify which feature to treat as our target for prediction. Below, we choose a sequence length of 8, meaning 7 steps for model inputs plus 1 step for the label.

```r
open <- paste(best_asset, "Open", sep = ".");close <- paste(best_asset, "Close", sep = ".")
high <- paste(best_asset, "High", sep = ".");low <- paste(best_asset, "Low", sep = ".")
rsi = "RSI"; macd = "MACD"; volume_ma = "Volume_MA"; seq_length <- 8
features <- data_scaled[, c(open, high, low, close, macd, volume_ma)]
split_data <- train_test_split(features, seq_length, target_feature=4)
x_train <- split_data$x_train; y_train <- split_data$y_train
x_test <- split_data$x_test; y_test <- split_data$y_test
```

Finally, we split part of the training set again for validation. This secondary split is helpful for hyperparameter tuning without contaminating our final test set.

```r
split_validation <- function(x, y, valid_prop = 0.2) { total <- dim(x)[1]
  valid_size <- round(valid_prop * total);train_size <- total - valid_size
  x_train_tune <- x[1:train_size, , , drop = FALSE]
  x_val <- x[(train_size + 1):total, , , drop = FALSE]; y <- as.matrix(y)
  y_train_tune <- y[1:train_size, , drop = FALSE]
  y_val <- y[(train_size + 1):total, , drop = FALSE]
  return(list(x_train_tune = x_train_tune,y_train_tune = y_train_tune,
    x_val = x_val,y_val = y_val))}
split_data <- split_validation(x_train, y_train, valid_prop = 0.2)
x_train_tune <- split_data$x_train_tune; y_train_tune <- split_data$y_train_tune
x_val <- split_data$x_val; y_val <- split_data$y_val
```

Optimising LSTM Parameters

The LSTM parameters are optimised using two techniques: grid search and genetic algorithms. This was done to compare the results from utilising traditional versus evolutionary approaches and conclude the pros and cons of each. Furthermore, the optimised parameters identified from this process are used by the LSTM to make predictions in conjunction with the proposed trading rule.

```
tune_lstm <- function(learningrate, hidden_dim, num_layers, numepochs, batch_size) {
  model <- trainr( Y = y_train_tune, X = x_train_tune, learningrate = learningrate,
    hidden_dim = hidden_dim, num_layers = num_layers, numepochs = numepochs,
    network_type = "lstm", seq_to_seq_unsync = TRUE,batch_size = batch_size)
  predictions <- predictr(model, x_val)
  mse <- mean((predictions - y_val)^2, na.rm = TRUE); return(mse)}
```

Grid Search

Grid search is a traditional approach to identifying optimal hyperparameter values for machine learning models. In this approach, the key hyperparameters to be tested are listed inside a vector, which the algorithm then systematically iterates over each combination and records the result. In this case, the mean squared error (MSE) is used on validation data to determine the current performance.

```
lr_vals <- c(0.001, 0.005, 0.01); hd_vals <- c(8, 16, 32, 64, 128) # Grid parameters
nl_vals <- c(1, 2, 3); ne_vals <- c(50, 100, 150, 200); bs_vals <- c(8, 16, 32, 64)
```

```
run_grid_search <- function(lr_vals, hd_vals, nl_vals, ne_vals, bs_vals){
  for (lr in lr_vals) {for (hd in hd_vals) {for (nl in nl_vals) {
    for (ne in ne_vals) {for (bs in bs_vals) { current_mse <- tune_lstm(lr,hd,nl,ne,bs)
            log_results(lr, hd, nl, ne, bs, current_mse)}}}}}}
#run_grid_search(lr_vals, hd_vals, nl_vals, ne_vals, bs_vals)
#best_params_GS <- results[which.min(results$mse), ]
```

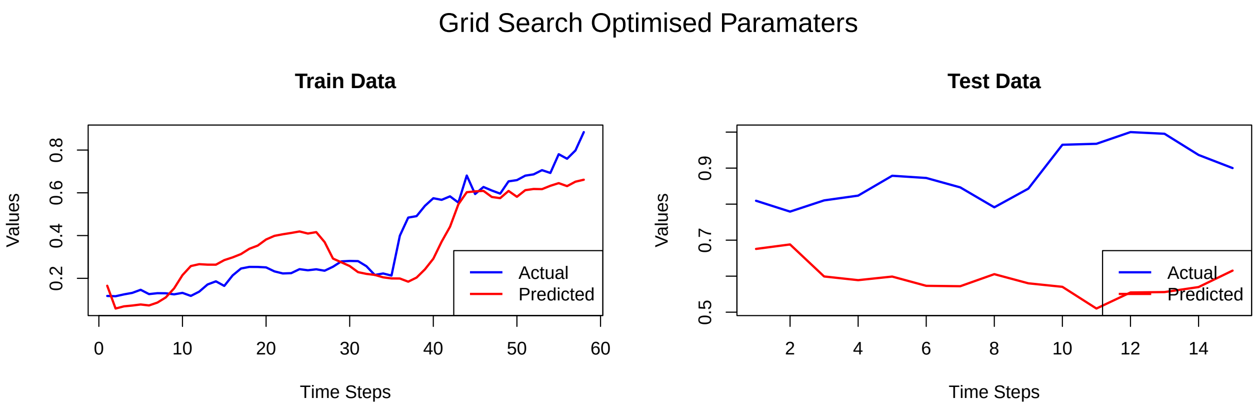
Genetic Algorithm

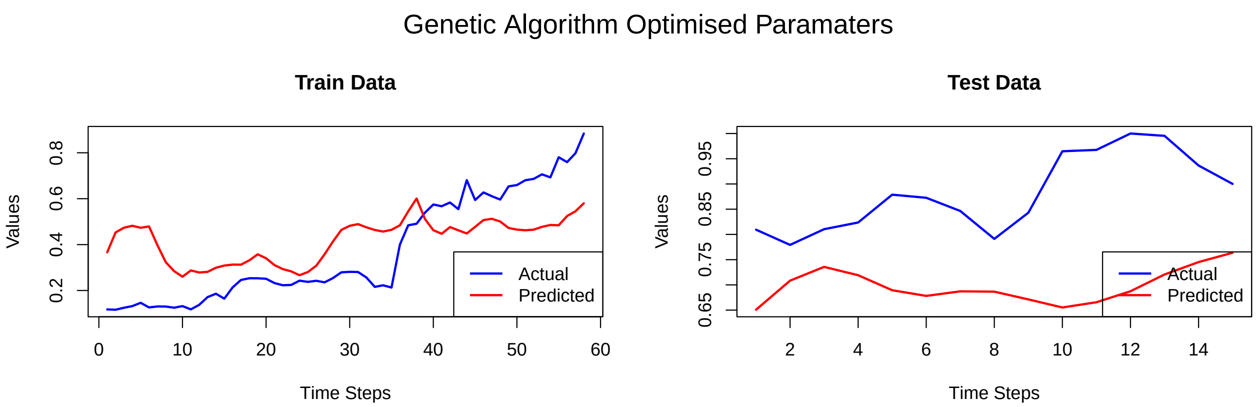
A genetic algorithm is an evolutionary process that mimics natural selection and genetics. This algorithm has been used to identify optimal hyperparameters within specified ranges (lower and upper). This implementation has a maximum of 100 iterations and will stop executing if the fitness does not improve after 20 iterations. The fitness is determined using the fitness function which evaluates performance against the MSE value.

```
fitness_function <- function(params) {
  lr <- params[1]; hd <- round(params[2]); nl <- round(params[3])
  ne <- round(params[4]); bs <- round(params[5])
  mse <- tune_lstm(lr, hd, nl, ne, bs); return(-mse)}
run_ga <- function(){ ga_result <- ga(type = "real-valued",fitness = fitness_function,
    lower = c(0.0001, 8, 1, 50, 8),upper = c(0.01, 128, 3, 200, 64),
    popSize = 20,maxiter = 100,run = 20); return(ga_result)}
#ga_result <- run_ga(); best_params_GA <- ga_result@solution
```

Optimisation Comparisons

Through experimenting with both of the above approaches key benefits and downfalls of each have been identified. First, Grid search is strictly limited to searching the specified hyperparameters whereas the GA solution can navigate the search space more effectively only being restricted to lower and upper bounds. Furthermore, both algorithms are computationally expensive, although, genetic algorithms have an edge as they can effectively terminate execution if the performance has not improved over a specified number of iterations, whereas grid search must evaluate all combinations. Finally, this difference between the two approaches is what sets them apart as a GA can get stuck in a local maximum and never converge to the optimal solution, on the other hand, grid search will evaluate all provided combinations guaranteeing the most optimal from the provided is found. Overall, both methods gain a similar performance using MSE. However, grid search slightly outperforms the GA result with an MSE score of 0.0161161 compared to 0.03577853 on the test (unseen) data, for this reason, the LSTM trained with optimised parameters from grid search will be used for algorithmic trading.





TensorFlow LSTM

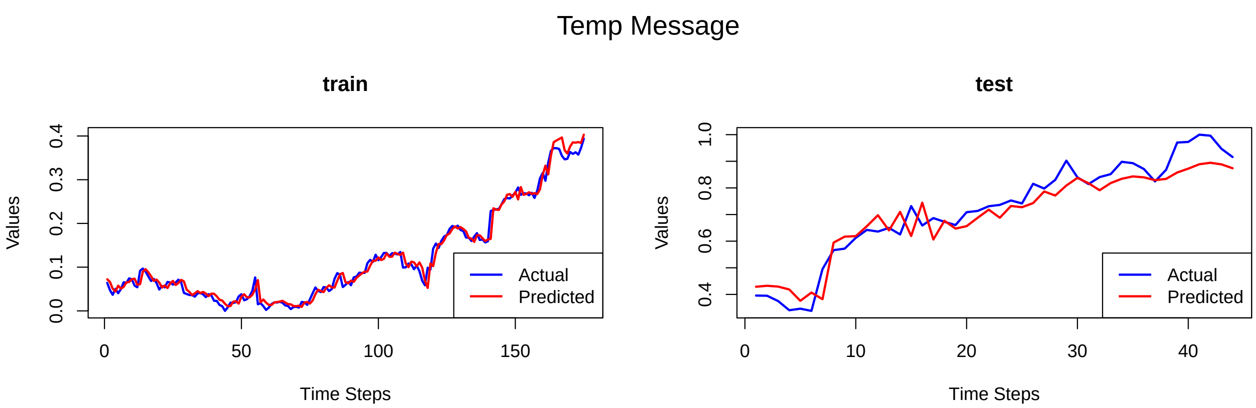
After observing suboptimal performance with our initial approach, we decided to utilise the TensorFlow framework to build and train a deeper LSTM network. The R interface to TensorFlow provides a higher-level API and greater flexibility in model design, allowing us to stack multiple LSTM layers and customise hyperparameters such as the hidden units and learning rate. Additionally, this setup supports advanced optimisations and GPU acceleration, which can significantly improve training speed and predictive performance. As a result, the deeper LSTM architecture built with TensorFlow was able to capture more complex temporal dynamics in the data and deliver significantly more accurate predictions.

Below, we define a deeper LSTM architecture using TensorFlow. The network consists of three LSTM layers stacked on top of each other, this is then followed by a dense layer that outputs a single value. Stacking multiple LSTM layers helps the model capture more complex temporal patterns in our time series data. We compile the model with the Adam optimiser and use MSE as the primary loss function. Finally, we train the model for 200 epochs while feeding samples in batches of size 32 at each iteration. Given the success of grid search on the previous LSTM, this method has been employed again to optimise the following parameters: learning rate, hidden dimensions, number of epochs and batch size.

```
#lr <- best_params$learningrate; hd <- best_params$hidden_dim
#ne <- best_params$numepochs; bs <- best_params$batch_size
train_model <- function(lr, hd, ne, bs){
  model <- keras_model_sequential() %>%
  layer_lstm(units = hd, input_shape = c(7, 6), return_sequences = TRUE) %>%
  layer_lstm(units = hd, return_sequences = TRUE) %>%
  layer_lstm(units = hd) %>%
  layer_dense(units = 1, activation = "tanh")
  model %>% compile( optimizer = optimizer_adam(learning_rate = lr),
    loss = "mse", metrics = c("mse"))
  history <- model %>% fit( x_train, y_train, epochs = ne, batch_size = bs,
    validation_split = 0.2, verbose = 0); return(model)}
#model <- train_model(lr, hd, ne, bs)
#save_model_hdf5(model, "tensorflow_lstm.keras")
model <- load_model_hdf5("tensorflow_lstm.keras")
```

```
## 6/6 - 1s - 559ms/epoch - 93ms/step
```

```
## 2/2 - 0s - 15ms/epoch - 7ms/step
```



Trading

```
starting_funds = 10000
investment = starting_funds
cash_on_hand = starting_funds
shares = 0

inverse_scale <- function(scaled_value, unscaled_min, unscaled_max) {
  scaled_value * (unscaled_max - unscaled_min) + unscaled_min
}

predictions_scaled = model %>% predict(x_test)
```

```
## 2/2 - 0s - 15ms/epoch - 8ms/step
```

```r
unscaled_min_close = min(data[, paste(best_asset, "Close", sep = ".")])
unscaled_max_close = max(data[, paste(best_asset, "Close", sep = ".")])

predictions_unscaled = inverse_scale(predictions_scaled, unscaled_min_close, unscaled_max_close)
actual_unscaled = inverse_scale(y_test, unscaled_min_close, unscaled_max_close)
#predictions_unscaled
#actual_unscaled
```

```r
trading_rule = data.frame(
  Date = index(tail(best_asset_data, nrow(y_test))),
  actual_price = rep(NA, nrow(y_test)),
  predicted_price = rep(NA, nrow(y_test)),
  action = character(nrow(y_test)),
  asset_value = numeric(nrow(y_test)),
  shares_held = numeric(nrow(y_test)),
  cash_held = numeric(nrow(y_test)),
  daily_profit_loss = numeric(nrow(y_test)) # New column for daily P/L
)

trading_rule$asset_value[1] = starting_funds
trading_rule$shares_held[1] = shares
trading_rule$cash_held[1] = cash_on_hand
trading_rule$daily_profit_loss[1] = 0
trading_rule$actual_price = actual_unscaled
trading_rule$predicted_price = predictions_unscaled

threshold_buy = 0.01
threshold_sell = -0.01
loss_minimisation_threshold = -0.05
last_buy_price = 0

next_day_action = character(nrow(trading_rule))
next_day_action[1] = "HOLD"

for(i in 1:(nrow(trading_rule) - 1)){
  current_price = trading_rule$actual_price[i]
  predicted_price = trading_rule$predicted_price[i]
  action = "HOLD"

  if(!is.na(predicted_price) && !is.na(current_price)){
    predicted_change_percentage = (predicted_price - current_price) / current_price
    if(predicted_change_percentage > threshold_buy && cash_on_hand > predicted_price){
      action = "BUY"
    } else if(predicted_change_percentage < threshold_sell && shares > 0){
      action = "SELL"
    }
  }
  next_day_action[i + 1] = action
}

for(i in 1:nrow(trading_rule)){
  previous_asset_value = trading_rule$asset_value[i]
```

```
  if(i > 1){
    cash_on_hand = trading_rule$cash_held[i-1]
    shares = trading_rule$shares_held[i-1]
    previous_asset_value = trading_rule$asset_value[i-1]
  }

  trade_action = next_day_action[i]
  current_price = trading_rule$actual_price[i]

  if(trade_action == "BUY" && cash_on_hand > 0){
    buy_quantity = floor(cash_on_hand / current_price)
    if(buy_quantity > 0){
      shares = shares + buy_quantity
      cash_on_hand = cash_on_hand - (buy_quantity * current_price)
      last_buy_price = current_price
    }
  } else if(trade_action == "SELL" && shares > 0){
    sell_value = shares * current_price
    # Loss minimisation sell
    if (last_buy_price > 0 && (current_price - last_buy_price) / last_buy_price < loss_minimisation_thre
      cash_on_hand = cash_on_hand + sell_value
      shares = 0
      last_buy_price = 0
      trade_action = "SELL OUT"
    } else {
      cash_on_hand = cash_on_hand + sell_value
      shares = 0
      last_buy_price = 0
    }
  }

  trading_rule$action[i] = trade_action
  trading_rule$asset_value[i] = cash_on_hand + (shares * current_price)
  trading_rule$shares_held[i] = shares
  trading_rule$cash_held[i] = cash_on_hand

  # Calculate daily profit/loss
  if (i > 1) {
    trading_rule$daily_profit_loss[i] = trading_rule$asset_value[i] - previous_asset_value
  }

  # Sell all on the final day
  if (i == nrow(trading_rule) && trading_rule$shares_held[i] > 0) {
    final_sell_value = trading_rule$shares_held[i] * current_price
    trading_rule$asset_value[i] = trading_rule$cash_held[i] + final_sell_value
    trading_rule$cash_held[i] = trading_rule$cash_held[i] + final_sell_value
    trading_rule$shares_held[i] = 0
    trading_rule$action[i] = "SELL"
  }
}
```

```
final_asset_value = tail(trading_rule$asset_value, 1)
initial_investment = starting_funds
```

```r
profit_loss = final_asset_value - initial_investment
roi = (profit_loss / initial_investment) * 100

cat("\nFinal Asset Value: $", round(final_asset_value, 2), "\n")
```

```
##
## Final Asset Value: $ 17159.5
```

```r
cat("Profit/Loss: $", round(profit_loss, 2), "\n")
```

```
## Profit/Loss: $ 7159.5
```

```r
cat("Return on Investment (ROI): ", round(roi, 2), "%\n")
```

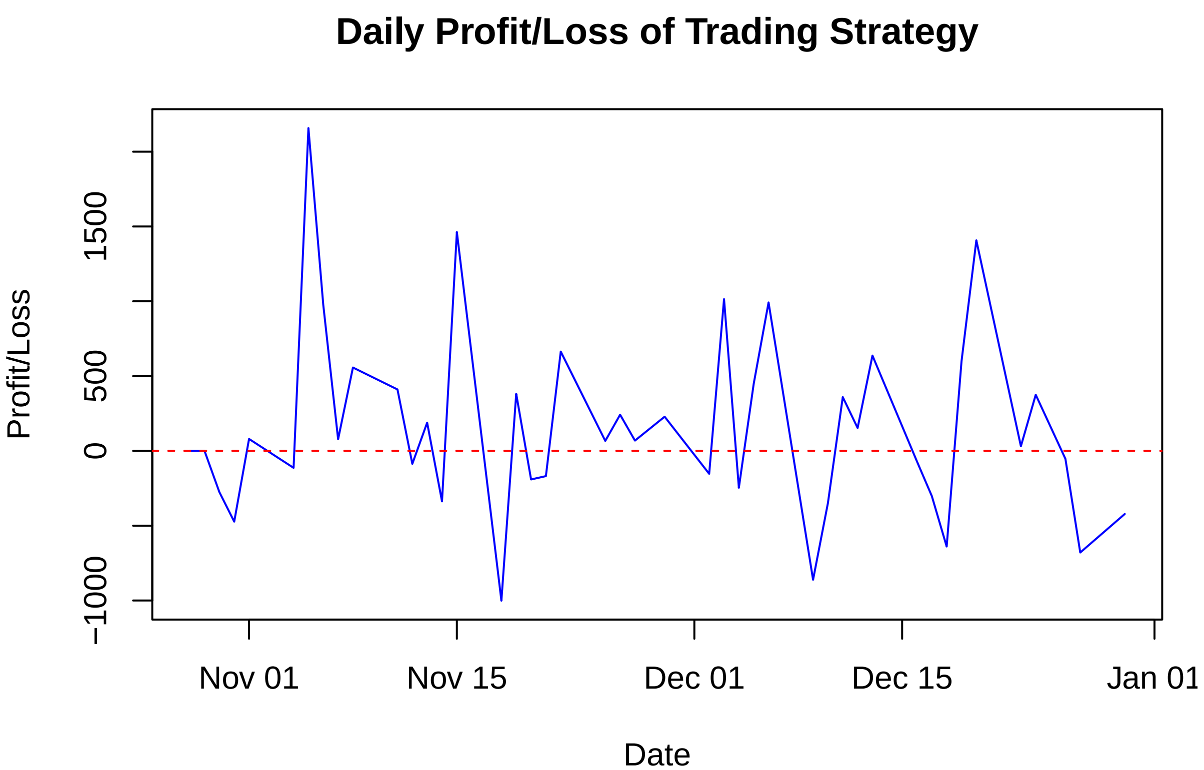
```
## Return on Investment (ROI):  71.6 %
```

```r
plot_trading_simulation = function(trade_log) {
  plot(trading_rule$Date, trading_rule$asset_value, type = "l", col = "green",
       xlab = "Date", ylab = "Asset Value ($)",
       main = "Trading Strategy Performance")
  grid()
}

plot_trading_simulation(trade_log)
```

```r
plot(trading_rule$Date, trading_rule$daily_profit_loss,
     type = "l",  # "l" for line plot
     xlab = "Date",
     ylab = "Profit/Loss",
     main = "Daily Profit/Loss of Trading Strategy",
     col = "blue")

# Add a horizontal line at zero for reference
abline(h = 0, lty = "dashed", col = "red")
```




```r
print(trading_rule)
```

```
##          Date actual_price predicted_price action asset_value shares_held
## 1  2024-10-28        44.97        47.01913   HOLD    10000.00           0
## 2  2024-10-29        44.93        47.24749    BUY    10000.00         222
## 3  2024-10-30        43.69        47.05579    BUY     9724.72         222
## 4  2024-10-31        41.56        46.38199    BUY     9251.86         222
## 5  2024-11-01        41.92        43.74469    BUY     9331.78         222
## 6  2024-11-04        41.41        45.66875    BUY     9218.56         222
## 7  2024-11-05        51.13        44.12522    BUY    11376.40         222
## 8  2024-11-06        55.53        57.28683   HOLD    12353.20         222
## 9  2024-11-07        55.88        58.66763    BUY    12430.90         222
## 10 2024-11-08        58.39        58.78885    BUY    12988.12         222
## 11 2024-11-11        60.24        61.09233   HOLD    13398.82         222
## 12 2024-11-12        59.85        63.65119    BUY    13312.24         222
## 13 2024-11-13        60.70        60.12133    BUY    13500.94         222
## 14 2024-11-14        59.18        64.42734   HOLD    13163.50         222
## 15 2024-11-15        65.77        58.81952    BUY    14626.48         222
```

```
## 16 2024-11-18        61.26        66.57191   HOLD    13625.26          222
## 17 2024-11-19        62.98        58.00428    BUY    14007.10          222
## 18 2024-11-20        62.12        62.35580   HOLD    13816.18          222
## 19 2024-11-21        61.36        60.54299   HOLD    13647.46          222
## 20 2024-11-22        64.35        61.10736   HOLD    14311.24          222
## 21 2024-11-25        64.65        63.05785   HOLD    14377.84          222
## 22 2024-11-26        65.74        64.93962   HOLD    14619.82          222
## 23 2024-11-27        66.05        63.06273   HOLD    14688.64          222
## 24 2024-11-29        67.08        65.79993   HOLD    14917.30          222
## 25 2024-12-02        66.39        65.51249   HOLD    14764.12          222
## 26 2024-12-03        70.96        66.46480   HOLD    15778.66          222
## 27 2024-12-04        69.85        69.16815   HOLD    15532.24          222
## 28 2024-12-05        71.87        68.22824   HOLD    15980.68          222
## 29 2024-12-06        76.34        70.55065   HOLD    16973.02          222
## 30 2024-12-09        72.46        72.32747   HOLD    16111.66          222
## 31 2024-12-10        70.89        71.09394   HOLD    15763.12          222
## 32 2024-12-11        72.51        69.45679   HOLD    16122.76          222
## 33 2024-12-12        73.20        71.12399   HOLD    16275.94          222
## 34 2024-12-13        76.07        72.12299   HOLD    16913.08          222
## 35 2024-12-16        75.75        72.66921   HOLD    16842.04          222
## 36 2024-12-17        74.39        72.45251   HOLD    16540.12          222
## 37 2024-12-18        71.51        71.79705   HOLD    15900.76          222
## 38 2024-12-19        74.21        72.10196   HOLD    16500.16          222
## 39 2024-12-20        80.55        73.57722   HOLD    17907.64          222
## 40 2024-12-23        80.69        74.48263   HOLD    17938.72          222
## 41 2024-12-24        82.38        75.50291   HOLD    18313.90          222
## 42 2024-12-26        82.14        75.84655   HOLD    18260.62          222
## 43 2024-12-27        79.08        75.46748   HOLD    17581.30          222
## 44 2024-12-30        77.18        74.57321   SELL    17159.50            0
##       cash_held daily_profit_loss
## 1  10000.00000           0.00000
## 2     25.53993           0.00000
## 3     25.53993        -275.28037
## 4     25.53993        -472.85939
## 5     25.53993          79.91929
## 6     25.53993        -113.21963
## 7     25.53993        2157.84027
## 8     25.53993         976.79949
## 9     25.53993          77.70051
## 10    25.53993         557.21963
## 11    25.53993         410.70051
## 12    25.53993         -86.58071
## 13    25.53993         188.70051
## 14    25.53993        -337.44010
## 15    25.53993        1462.97919
## 16    25.53993       -1001.21963
## 17    25.53993         381.84027
## 18    25.53993        -190.92014
## 19    25.53993        -168.71963
## 20    25.53993         663.77953
## 21    25.53993          66.60068
## 22    25.53993         241.97919
## 23    25.53993          68.82115
## 24    25.53993         228.65973
```

```
## 25    25.53993        -153.18054
## 26    25.53993        1014.53993
## 27    25.53993        -246.42014
## 28    25.53993         448.44095
## 29    25.53993         992.33858
## 30    25.53993        -861.35939
## 31    25.53993        -348.53993
## 32    25.53993         359.64061
## 33    25.53993         153.17885
## 34    25.53993         637.14061
## 35    25.53993         -71.03993
## 36    25.53993        -301.92014
## 37    25.53993        -639.35939
## 38    25.53993         599.39932
## 39    25.53993        1407.48088
## 40    25.53993          31.07986
## 41    25.53993         375.17885
## 42    25.53993         -53.27953
## 43    25.53993        -679.31946
## 44 17159.50000        -421.80034
```

References

[1] A. Dangi, “Optimizing LSTM Network using Genetic Algorithm for Stock Market Price Prediction,” 24 April 2023. [Online]. Available: https://www.linkedin.com/pulse/optimizing-lstm-network-using-genetic-algorithm-stock-akash-dangi/. [Accessed 10 April 2025].

[2] R. M. Dhokane and S. Agarwal, “LSTM Deep Learning Based Stock Price Prediction with Bollinger Band, RSI, MACD, and OHLC Features,” International Journal of Intelligent Systems and Applications in Engineering, vol. 12, no. 3, p. 1169–1176, 2024.