CS971: AI for Finance Assignment 2

Stewart Macfarlane, Vladimir Lenkov, Alvee Kabir

2025-04-09

Data Selection

```
assets <- tq_index("SP500")
```

```
## Getting holdings for SP500
```

```
head(assets)
```

```
## # A tibble: 6 x 8
##   symbol company       identifier sedol weight sector shares_held local_currency
##   <chr>  <chr>         <chr>      <chr>  <dbl> <chr>        <dbl> <chr>
## 1 MSFT   MICROSOFT CO~ 594918104  2588~ 0.0622 -         91325520 USD
## 2 AAPL   APPLE INC     037833100  2046~ 0.0611 -        184544185 USD
## 3 NVDA   NVIDIA CORP   67066G104  2379~ 0.0557 -        300856483 USD
## 4 AMZN   AMAZON.COM I~ 023135106  2000~ 0.0380 -        115870655 USD
## 5 META   META PLATFOR~ 30303M102  B7TL~ 0.0264 -         26902602 USD
## 6 BRK-B  BERKSHIRE HA~ 084670702  2073~ 0.0213 -         22517536 USD
```

```
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)
        cat("\nSkipping asset:", sym, "\n")
      }
    )
  })

  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)
}
```

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

```
## Warning: Failed to open
## 'https://query2.finance.yahoo.com/v8/finance/chart/-?period1=1704067200&period2=1735603200&interval=
## The requested URL returned error: 404

##
## Skipping asset: -
```

```r
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)
}
```

```r
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
  }
}
```

```r
best_asset
```

```
## [1] "PLTR"
```

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

Data Preprocessing

```r
rsi = TTR::RSI(Cl(best_asset_data), n = 14)
ema_short = TTR::EMA(Cl(best_asset_data), n = 12)
ema_long = TTR::EMA(Cl(best_asset_data), n = 26)
macd = ema_short - ema_long
volume_ma = TTR::SMA(Vo(best_asset_data), n = 20)
```

```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)
#best_asset_data
```

```r
data <- data.frame(best_asset_data[,1], best_asset_data[,2], best_asset_data[,3], best_asset_data[,4], 
min_max_normalize <- function(x) {
  (x - min(x)) / (max(x) - min(x))
}

data_scaled <- as.data.frame(lapply(data, min_max_normalize))
#data_scaled
```

```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))
}
```

```r
seq_length <- 8
open <- paste(best_asset, "Open", sep = ".")
high <- paste(best_asset, "High", sep = ".")
low <- paste(best_asset, "Low", sep = ".")
close <- paste(best_asset, "Close", sep = ".")
rsi = "RSI"
macd = "MACD"
volume_ma = "Volume_MA"
features <- data_scaled[, c(open, high, low, close, rsi, macd, volume_ma)]

split_data <- train_test_split(features, seq_length, ncol(features))
x_train <- split_data$x_train
y_train <- split_data$y_train
x_test <- split_data$x_test
y_test <- split_data$y_test
```

```r
str(features)
```

```
## 'data.frame':    226 obs. of  7 variables:
```

```
##  $ PLTR.Open : num  0.016 0.0537 0.071 0.0529 0.0468 ...
##  $ PLTR.High : num  0.0451 0.0638 0.0646 0.0711 0.0575 ...
##  $ PLTR.Low  : num  0.0172 0.049 0.0531 0.0572 0.0489 ...
##  $ PLTR.Close: num  0.0506 0.0653 0.0632 0.074 0.0572 ...
##  $ RSI       : num  0.917 0.947 0.928 0.951 0.805 ...
##  $ MACD      : num  0.296 0.354 0.395 0.431 0.443 ...
##  $ Volume_MA : num  0.558 0.664 0.732 0.778 0.821 ...
```

```r
# For hyperparameter tuning, we split part of x_train/y_train to act as a validation set
# For example, we use 80% for training and 20% for validation
split_validation <- function(x, y, valid_prop = 0.2) {
  total <- dim(x)[1]
  valid_size <- round(valid_prop * total)
  train_size <- total - valid_size

  # Subset x without dropping dimensions
  x_train_tune <- x[1:train_size, , , drop = FALSE]
  x_val <- x[(train_size + 1):total, , , drop = FALSE]

  # Force y to be a matrix to ensure two dimensions
  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 the training data for tuning
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
```

Optimisation Stuff

```r
# Define a tuning function that trains the LSTM and returns the mean squared error on the validation se
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,
```

```r
    numepochs = numepochs,
    network_type = "lstm",
    seq_to_seq_unsync = TRUE,
    batch_size = batch_size
  )
  # Generate predictions on the validation set
  predictions <- predictr(model, x_val)
  mse <- mean((predictions - y_val)^2, na.rm = TRUE)
  return(mse)
}
```

Grid Search

```r
# Set up grid search parameters (you can adjust or expand the grid as needed)
learningrate_vals <- c(0.001, 0.01)
hidden_dim_vals   <- c(16, 32)
num_layers_vals   <- c(1, 2)
numepochs_vals    <- c(1, 2)
batch_size_vals   <- c(16, 32)

# Initialize a data frame to store results
results <- data.frame(
  learningrate = numeric(0),
  hidden_dim = numeric(0),
  num_layers = numeric(0),
  numepochs = numeric(0),
  batch_size = numeric(0),
  mse = numeric(0)
)
```

```r
# Grid search
for (lr in learningrate_vals) {
  for (hd in hidden_dim_vals) {
    for (nl in num_layers_vals) {
      for (ne in numepochs_vals) {
        for (bs in batch_size_vals) {
          current_mse <- tune_lstm(learningrate = lr,
                                   hidden_dim = hd,
                                   num_layers = nl,
                                   numepochs = ne,
                                   batch_size = bs)
          results <- rbind(results, data.frame(
            learningrate = lr,
            hidden_dim = hd,
            num_layers = nl,
            numepochs = ne,
            batch_size = bs,
            mse = current_mse
          ))
          #cat("Tested: lr=", lr, ", hd=", hd, ", nl=", nl, ", ne=", ne, ", bs=", bs,
          #    "-> MSE=", current_mse, "\n")
```

```r
        }
      }
    }
  }
}

# Identify the best parameter set (lowest MSE)
best_params <- results[which.min(results$mse), ]
cat("Best Hyperparameters:\n")
```

```
## Best Hyperparameters:
```

```r
print(best_params)
```

```
##    learningrate hidden_dim num_layers numepochs batch_size        mse
## 15        0.001         32          2         2         16 0.02282863
```

Genetic Algorithm

```r
fitness_function <- function(params) {
  learningrate <- params[1]
  hidden_dim <- round(params[2])
  num_layers <- round(params[3])
  numepochs <- round(params[4])
  batch_size <- round(params[5])

  mse <- tune_lstm(
    learningrate = learningrate,
    hidden_dim = hidden_dim,
    num_layers = num_layers,
    numepochs = numepochs,
    batch_size = batch_size
  )
  return(-mse)
}

ga_result <- ga(
  type = "real-valued",
  fitness = fitness_function,
  lower = c(0.001, 1, 1, 1, 1),
  upper = c(0.01, 64, 5, 1, 64),
  popSize = 20,
  maxiter = 100,
  run = 20
)

best_params <- ga_result@solution
cat("Best Hyperparameters:\n")
```

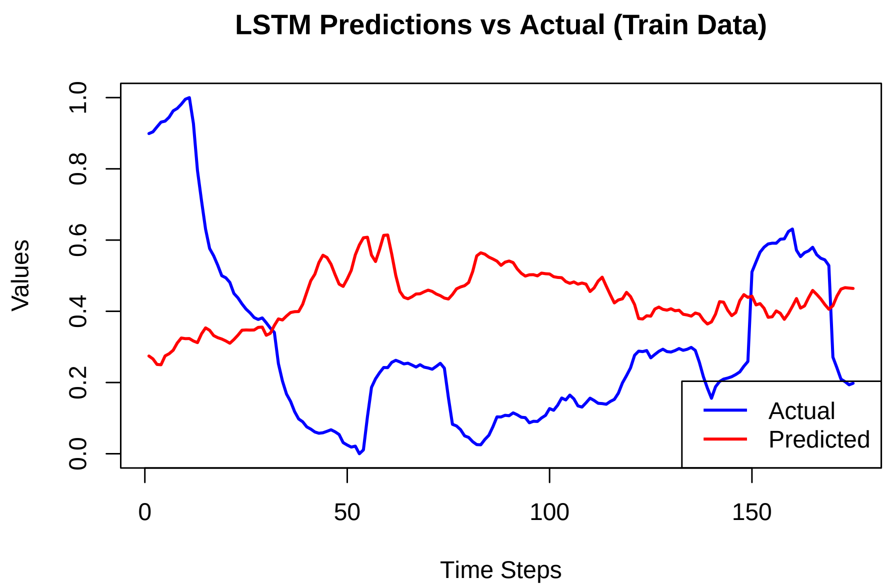
```
## Best Hyperparameters:
```

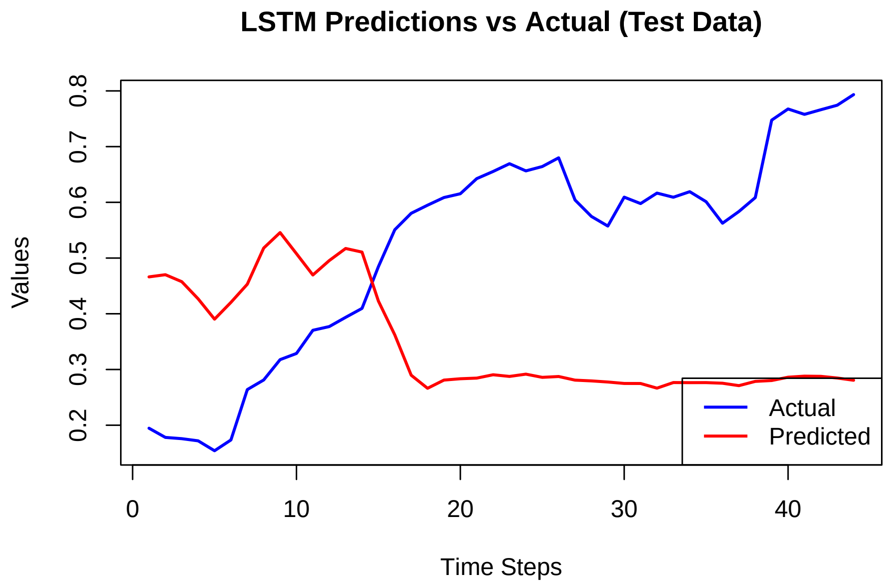
```
print(best_params)
```

```
##                x1       x2       x3 x4       x5
## [1,] 0.006256953 21.35258 1.917781  1 34.65891
```

LSTM

```
model <- trainr(
  Y = y_train,
  X = x_train,
  learningrate = 0.001,
  hidden_dim = 32,
  num_layers = 1,
  numepochs = 1,
  network_type = "lstm",
  seq_to_seq_unsync = T,
  bactch_size = 32
)
```




LSTM Predictions vs Actual (Test Data)
Values
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
10
20
30
40
Time Steps
Actual
Predicted