CS815: AI for Finance

Assignment 2 - Algorithmic Trading

Olivier Delree - 202278679 Eilidh Pike - 202274131

12/04/2023

University of Strathclyde Glasgow

1 Setup

```
library(quantmod)
library(RSNNS)
library(pso)
library(neuralnet)
```

2 Background of the problem

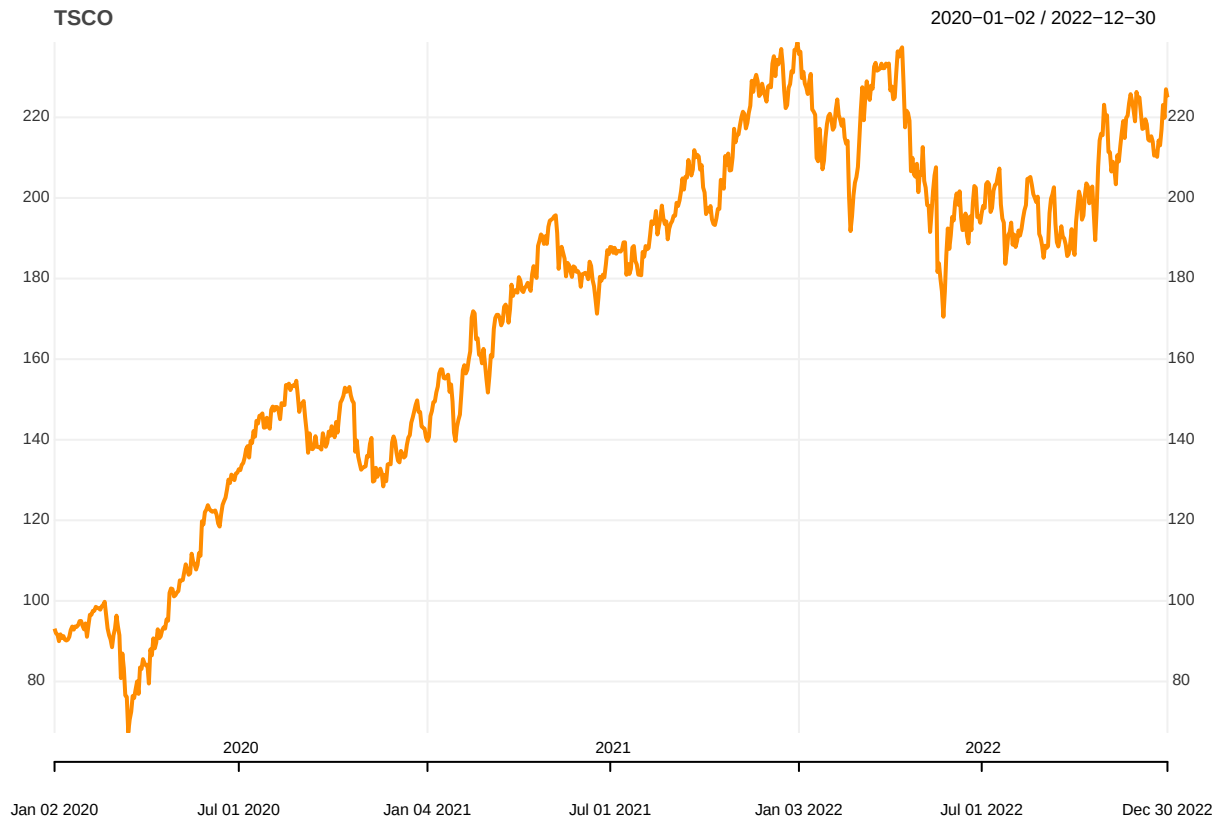
Algorithmic trading involves using financial market data to train a set of algorithms to execute buy and sell orders based on different types of market data. Price based prediction trading is a form of technical investment analysis which involves training algorithms to execute these orders based on indicator data such as historical price, volume, and volatility (Vinitnantharat et al. 2019). Investment analysis can be further refined using by using algorithms based on computational intelligence.

Two of these algorithms, PSO and a traditional artificial neural network, are employed and trained on a randomly selected stock using the above mentioned financial markers ranging two years. Particle Swarm Optimisation is an evolutionary computing algorithm based on population metaheuristics to optimize the trading strategies by iteratively improving the candidate solution. An ANN uses traditional computational intelligence methods to determine the most favorable weights based on the historical financial data. The most favourable market indicators are identified using a correlation matrix, and these are fed into both algorithms with their performance compared.

The training period spans the period between 01/01/2020 and 31/12/2020, with the last 50 days being used as a small testing period. The main backtesting period spans between 01/01/2021 and 31/12/2022.

3 Overview of the data

Tractor Supply Company is a publicly traded company on the NASDAQ. It is currently ranked 294th on the Fortune 500 list. The brand sells home improvement, agriculture, lawn and garden maintenance, livestock, equine and pet care for the general public. The stock price from 2020 to 2023 follows a generally upwards movement meaning the approach does not necessarily need to be too robust with regards to changing trends. 2020 to 2023 follows a generally upwards movement meaning the approach does not necessarily need to be too robust with regards to changing trends.



4 Details of the trading strategy

The chosen strategy relies simply on the EMA12 and EMA28, and on the predicted closing price from our model. This dictates at market opening whether it is a good time to buy, hold, or sell.

The approach taken is a bit naive and would most likely not be recommended in practice, especially on a stock which has been known to be relatively unpredictable. The simulation assume that buying and selling should be done all at once, meaning that a buy order invests all of the available capital into the asset while a sell order sells all available shares. This leaves the investor at a high risk when buying as the entirety of their capital is invested at once.

Although this approach is not too realistic, it allows us to evaluate the relative strength of the strategy and, to a certain extent, how reliable the predictions are when combined with other indicators.

4.1 Simulation function

```
simulate_investment <- function(opening_prices, position) {
  capital <- 1e4         # Starting capital of US$10,000
  invested_funds <- 0    # No starting investment

  # Because the entire capital/investment is used for buy/sell orders,
  # the positions need to be processed so that multiple similar orders
  # are reduced to a single buy/sell order followed by hold orders until
```

```
  # an opposite order is reached.
  processed_position <- xts(rep(0, length(position)), order.by = index(position))
  previous <- -1        # Hardcoding the first order to be a buy
  for (index in 1:length(position)) {
    item <- as.numeric(position[index])
    if (item != previous) {
      processed_position[index] <- item
      previous <- item
    }
  }

  # Simulate the buy/sell orders from the processed position
  previous_open <- 0
  for (index in 1:length(processed_position)) {
    order <- as.numeric(processed_position[index])
    current_open <- as.numeric(opening_prices[index])

    # Simulate investment growth
    if (index > 1) {
      growth <- current_open / previous_open
      invested_funds <- invested_funds * growth
    }

    if (order == -1) { # SELL
      sold_value <- invested_funds - (invested_funds %% current_open)
      capital <- capital + sold_value
      invested_funds <- invested_funds - sold_value
    }
    else if (order == 1) { # BUY
      bought_value <- capital - (capital %% current_open)
      capital <- capital - bought_value
      invested_funds <- invested_funds + bought_value
    }

    previous_open <- current_open
  }

  # return(processed_position)
  return(c(capital, invested_funds))
}
```

4.2 Particle Swarm Optimisation

4.2.1 PSO Fitness function

This fitness function is fed to the PSO. It takes in the weights of the current particle, normalise them, multiplies them with the features of the dataset, and compares the result to the real closing value. The returned value is the squared differences, meaning that particles are punished exponentially the further off their predictions are. Squaring also has the added benefit that all returned values are positive and that the PSO can safely try to minimise the cost.

```r
fitness_function <- function(weights, train_data) {
  weights <- weights / sum(weights)

  weighted_sums <- apply(train_data[, -ncol(train_data)] * weights, 1, sum)
  targets <- train_data[, ncol(train_data)]

  deltas <- targets - weighted_sums

  return(mean(deltas^2))
}
```

4.2.2 PSO Parameters

We chose particle swarm optimisation as our model. This allows us to optimise a set of weights so that it quickly converges on a good solution.

The lower and upper bounds for the values are -1 and 1. The maximum number of iterations has been increased from 1000 to 2000 to try and get a better model. Although this could lead to overfitting, because the trend of the data does not change much over the training and testing, it seems acceptable to train the model to assume that trend.

```r
lower <- -1
upper <- 1
control <- list(maxit = 2000)
```

4.3 Neural Network

4.3.1 NN Parameters

```r
nn_formula <- true_close ~ open + low_lag1 + high_lag1 + close_lag1 + ema12_lag1 +
        ema26_lag1 + day_of_year
hidden_layers <- 10
activation_function <- "logistic"
```

5 Presentation of the results from the training

5.1 Data preparation

5.1.1 Feature Engineering

To allow for easier preparation of training and backtesting data, we wrote this function to extract the features from the initial xts variable into a format that can be applied to our PSO solution.

```r
prepare_data <- function(raw_data) {
  open <- raw_data$TSCO.Open
  low_lag1 <- lag(raw_data$TSCO.Low)
  high_lag1 <- lag(raw_data$TSCO.High)

  close_lag1 <- lag(raw_data$TSCO.Close)
```

```
  ema12_lag1 <- lag(EMA(raw_data$TSCO.Close, n = 12))
  ema26_lag1 <- lag(EMA(raw_data$TSCO.Close, n = 26))

  day_of_year <- lubridate::yday(index(raw_data))

  prepared_data <- cbind(open,
                         low_lag1,
                         high_lag1,
                         close_lag1,
                         ema12_lag1,
                         ema26_lag1,
                         day_of_year,
                         raw_data$TSCO.Close)
  colnames(prepared_data) <- c("open",
                               "low_lag1",
                               "high_lag1",
                               "close_lag1",
                               "ema12_lag1",
                               "ema26_lag1",
                               "day_of_year",
                               "true_close")
  prepared_data <- as.xts(prepared_data)

  return(prepared_data)
}
```

The feature selection was done from a correlation matrix. The most correlated features from the pool of open, low_lag1, high_lag1, volume_lag1, close_lag1 to close_lag8, rsi_lag1, ema12_lag1, ema26_lag1, macd_lag1, smi_lag1, adx_lag1, and day_of_year were selected

The chosen features are the opening price, the low, high, and close prices of the previous day, as well as the EMA12 and EMA26 values of the previous day, and finally the day of the year. This last feature is there to try to account for seasonal changes over the year and produces better results than simply including the month of the year.

5.1.2 Data normalisation

Because not all the features are in the same range of values, the model needs the features to be normalised in order to efficiently predict the true closing price.

```
normalise_data <- function(prepared_data) {
  normalised_data <- normalizeData(prepared_data)
  normalisation_parameters <- getNormParameters(normalised_data)
  normalised_data <- xts(normalised_data, order.by = index(prepared_data))
  colnames(normalised_data) <- colnames(prepared_data)

  return(list(normalised_data = normalised_data,
              normalisation_parameters = normalisation_parameters))
}
```

5.2 Training

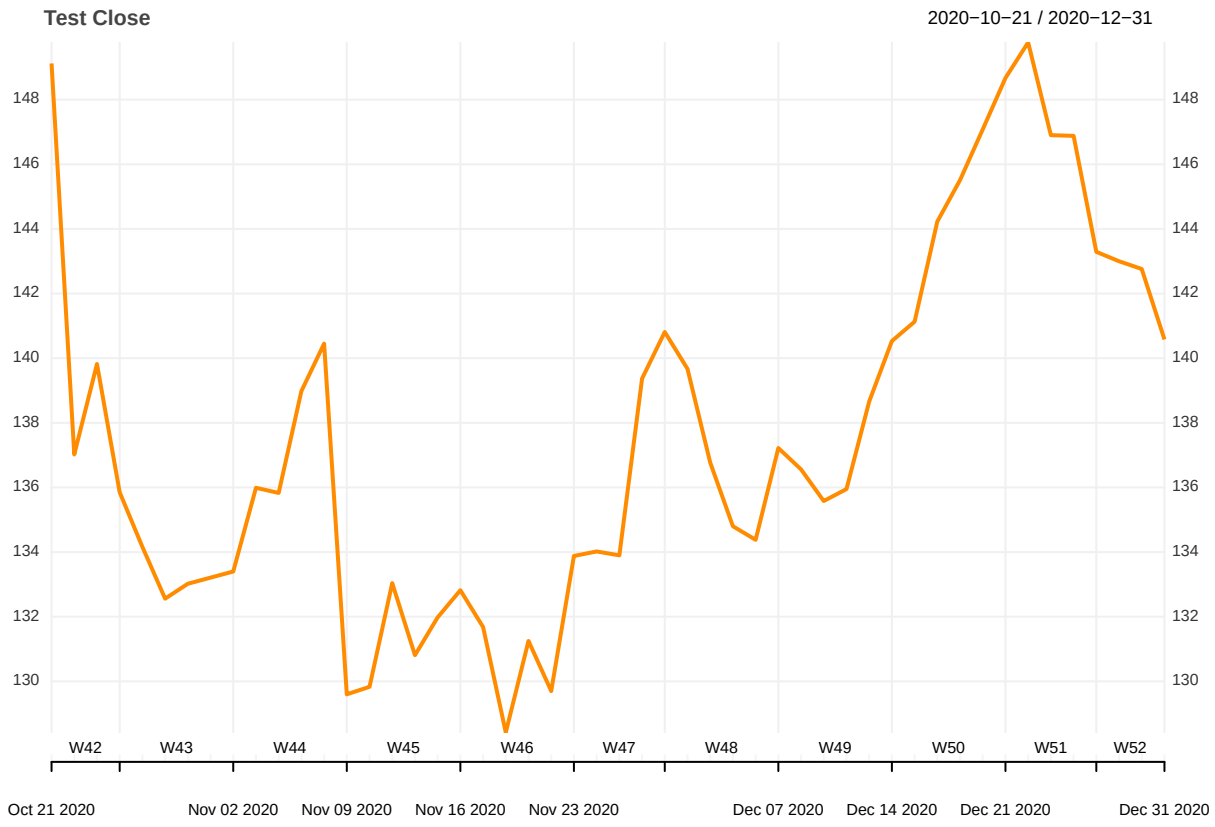
The year chosen for the training is 2020.

```r
# Note that the year 2019 is also included in the initial retrieval of
# prices. This is to allow the calculation of EMAs for the first 26 days
# of 2020. Once the data has been prepared, only 2020 is kept.
tsco <- getSymbols("TSCO", src = "yahoo",
                   from = "2019-01-01", to = "2022-01-01",
                   auto.assign = FALSE)

tsco_prepared <- prepare_data(tsco)[c("2020", "2021")]
tsco_normalisation <- normalise_data(tsco_prepared)
tsco_normalised <- tsco_normalisation$normalised_data["2020"]
tsco_normalisation_parameters <- tsco_normalisation$normalisation_parameters

# The last 50 days of the trading year are used for testing and to get
# the RMSE of our model.
train_data <- tsco_normalised[1:(nrow(tsco_normalised) - 50),]
test_data <- tsco_normalised[(nrow(tsco_normalised) - 49):nrow(tsco_normalised),]

# Training the PSO
pso <- psoptim(rep(NA, (ncol(train_data) - 1)),
               lower = lower,
               upper = upper,
               control = control,
               fn = fitness_function,
               gr = NULL,
               train_data)

solution <- pso$par / sum(pso$par)

# Training the NN
nn <- neuralnet(nn_formula,
                hidden = hidden_layers,
                act.fct = activation_function,
                linear.output = TRUE,
                data = train_data,
                algorithm = "backprop",
                learningrate = 0.0001,
                stepmax = 1e8)
```

```
## PSO test RMSE 2020: 12.36074
## NN test RMSE 2020: 6.435212
```

```
## From an investment of US$10,000, using a PSO-base strategy yields at total of US$9349.431.
## From an investment of US$10,000, using an NN-based strategy yields at total of US$9872.761.
```

Although this doesn't look like a great return over a 50-day period, it is worth noting that the stock followed a relatively erratic pattern during that time. In the grand scheme of things, not suffering great losses overall seems like an acceptable outcome.



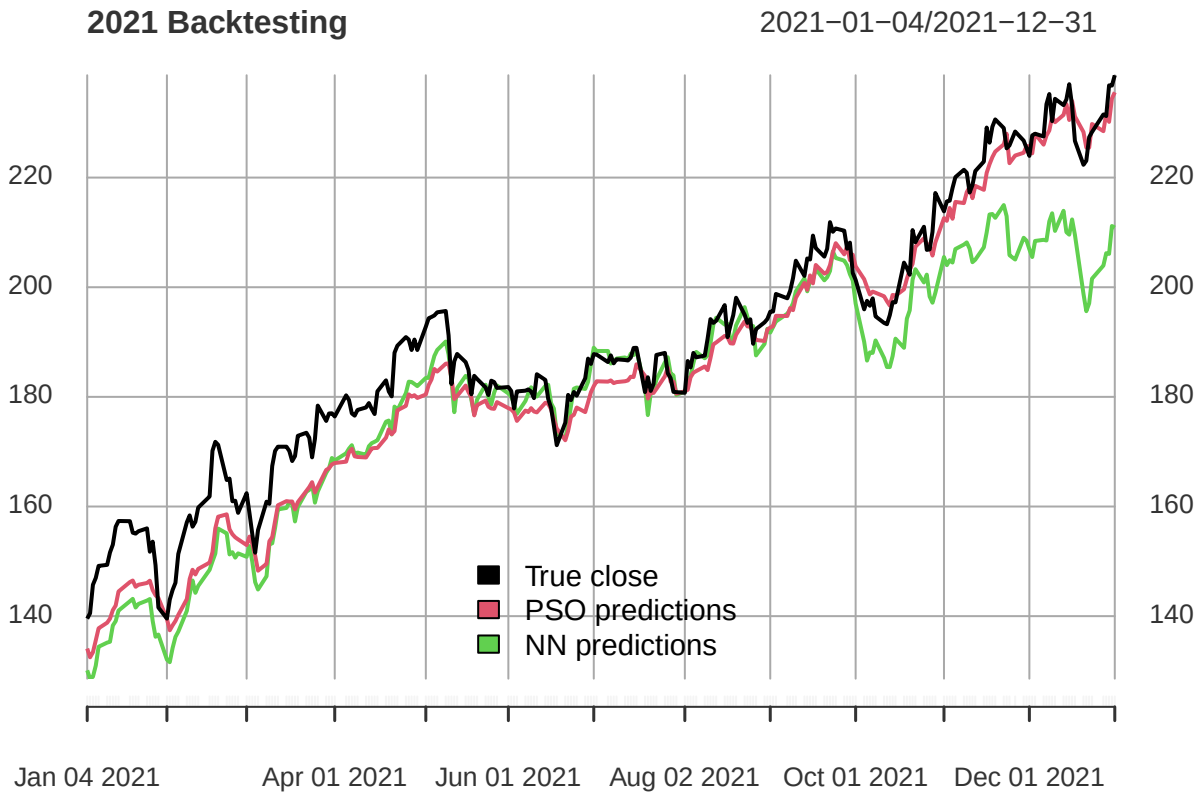
6 Presentation of the results from the backtesting

6.1 RMSE 2021

```
## PSO RMSE 2021: 6.67993
## NN RMSE 2021: 11.22654
```

```
## Working with data from all of 2021:
## From an investment of US$10,000, using a PSO-base strategy yields at total of US$11099.91.
## From an investment of US$10,000, using an NN-based strategy yields at total of US$10625.22.
```

6.2 Charting



7 Comparison against other approaches

The PSO-based simulation resulted in a loss over the last 50 days of 2020, while the NN-based simulation led to a gain in capital. This is expected as the PSO RMSE for the testing period was roughly triple that of the NN. However, this ratio is quickly reversed when looking at the backtesting period of 2021. Over that year, the PSO RMSE remains roughly 3 times smaller than for the NN. This worsening of predictions of the neural network could be due to various factor such as overfitting on the training set, an inability to use input values higher than that of the training set, or on the contrary underfitting and therefore a lack of predictive power. Overall, it appears that PSO is a better candidate for trading for longer time-periods.

8 Conclusions

A PSO model and NN model were compared with regards to their performance in price prediction based trading. A correlation matrix was employed to find the best features and technical indicators to use for training. Overall, PSO had the lowest rate of error in the long run, but NN performed better when used on the last 50 days of the training year. There may be other factors at play, such as the complexity of the calculations, differences in hyper-parameter tuning, as well as the complex fluctuations in the stock price. Overall, PSO would be a better model to use when looking to trade over a long period of time with a stock that keeps following the same overall trend. Meanwhile, NNs might be better suited to predict values at a smaller range but seem to be more robust to changes in overall trend.

9 References

Source: Wikipedia

Vinitnantharat et al. (2019) "Quantitative trading machine learning using differential evolution algorithm," 2019 16th International Joint Conference on Computer Science and Software Engineering (JCSSE) [Preprint]. Available at: https://doi.org/10.1109/jcsse.2019.8864226.

Mohamed, I. and Otero, F.E. (2022) "A performance study of multiobjective particle swarm optimization algorithms for market timing," 2022 IEEE Symposium on Computational Intelligence for Financial Engineering and Economics (CIFEr) [Preprint]. Available at: https://doi.org/10.1109/cifer52523.2022.9776019.