MSDS 6306. Autodidactic trading robot

Ivan Charkin, Volodymyr Orlov

Introduction

Mastering equity trading might be a challenging and risky pursuit. It takes years of practice and some mistakes to forge a good trading strategy. Once a strategy is developed it should be constantly adjusted to stay current in the always changing environment of financial stock markets. While majority of the trades nowadays are done by computers, there is still a small portion of trades that are made by real-life humans deciding to buy or sell shares in one company or another[1]. If a strategy employed by a human has not been coded using a computer programming language from start it might be hard to automate it later or share it with others when the strategy has matured. In this work we want to see whether machine learning algorithms can be used to automatically learn given strategy from decisions made by a trader without requiring any additional input other than sell/buy signal provided by trader and stock market data. We experiment with three methods which can be used to learn to imitate trader's signal. We arrived at a conclusion that while these methods works for simple trading strategies, both are not generalizable enough. Lastly, we attempt to explain why this is a hard problem to solve using a single, generic machine learning algorithm.

Problem Statement and Data Description

For this work we've took historical market data from Yahoo Finance's website. A sample of typical 5 data rows are displayed below.

```
##         Date     Open     High      Low    Close  Volume Company
## 1 2005-01-03 49.54198 49.83083 48.84174 48.93802 5251500     AXP
## 2 2005-01-04 48.93802 49.00805 48.01896 48.20277 4642200     AXP
## 3 2005-01-05 48.18527 48.20277 47.58131 47.65133 4681400     AXP
## 4 2005-01-06 47.56380 47.72136 47.09114 47.40625 4947600     AXP
## 5 2005-01-07 47.51128 47.56380 46.86356 47.05613 4426200     AXP
## 6 2005-01-10 46.96860 47.40625 46.93359 47.15241 4945600     AXP
```

Please refer to Table 1 for a detailed description of all variables.

Our goal was to train a robot that learns to imitate a trader by looking only at a signal generated by his/her strategy. Strategies employed by a trader are subject to following constrains:

- A strategy should not get any other input except for daily stock/eft quotes.
- A strategy followed by a trader should not change over time.

Trading Strategies

Although we want to build a robot that can learn any strategy we selected a set of concrete trading algorithms to validate robot's performance. For our experiment we've chosen a number of strategies of over increasing complexity and used signals from these strategies to benchmark our robot.

Moving Average Crossover base strategies

First two strategies are based on an idea of Moving Average Crossover[2]. We've chosen 2 algorithms based on this principle:

[1] https://www.washingtonpost.com/news/wonk/wp/2018/02/07/the-robots-v-robots-trading-that-has-hijacked-the-stock-market

[2] https://en.wikipedia.org/wiki/Moving_average_crossover

1. A simple 5-day moving average is observed and a signal is generated when moving average goes above or drops below a price at that day.
2. Two simple moving averages (2-day and 8-day) are observed and a signal is generated when short MA crosses long MA.

Gap up after downtrend

The difference between traditional "investing" and "trading" is that people doing investing research individual companies and make trading decisions based on news, company reports, rumors, etc. The technical traders base their decisions on the previous price pattern of the stock. In turn the technical traders can be divided into two rough categories: mathematical model traders and psychology traders. The former ones attempt to devise a mathematical model predicting stock behavior using oscillators, various indicators etc. The latter are trying to predict emotions of traders who own or are considering trading the stock. In this paper we are trying to emulate trading patters of a person not basing his decisions on mathematical models

The simple strategy shown in Figure 1 is an oversimplified example of psychology style of technical trading. Please note that in real life trading much more parameters than described here are used to improve the odds of a successful trade.

Let us consider a stock which has been falling in price for multiple days. Continuous selling of the stock likely depleted the ranks of the people who want to sell the stock and potentially attracted short sellers who sold the stock short in hopes of further falling prices. Now let us imagine that one day after several days of falling prices the stock opens up at a higher value than the previous day close. This event will potentially attract buyers who will anticipate reversal of the direction of the stock and will be attracted by relatively cheap prices. At the same time the short sellers may panic that the stock is going to go up which mean losing money and will start covering their positions (buying) driving the stock price even higher.

Breakout strategy

This strategy (oversimplified version) is using the concept that if the price of a stock stays relatively flat for an extended period of time there is a higher probability of this stock to continue moving in the same direction after the stock price left the "flat" zone. There are multiple underlying reasons for stocks staying at roughly the same price. For example if the the price of the stock reaches a round number such as 50, a lot of owners of the stock will be selling the stock because that is a round number they decided they were waiting for the stock to reach before selling it. Even if there are many people willing to buy the stock, the pressure from the sellers will keep the stock down.

To test the performance of the robot, we chose a short flat period to generate more signal. Figure 2 illustrates the sample strategy.

Robot Models

One way to think about this problem is to imagine a robot that attempts to learn a general form of a mapping $\phi : X \to Y$ from a d-dimensional vector of the real numbers, features, $X = (x1, x2, ..., x_L)$, where $x_l \in \mathbb{R}^d$ to a categorical output Y, signal, where $y \in Y = \{y_1, y_2, ..., y_K\}$.

Every day our robot takes a set of features derived from a fixed number of historical quotes preceding this day and predicts a category which we map to a buy or sell action. Thus, our robot is a function

$$\hat{y} = f(x, \theta).$$

where θ are model parameters, which maximizes a likelihood of an observed outcome produced by a human trader.

One important difference of this problem from a typical classifier is that a set of input features, X are unknown. If we need to build a mapping for a specific trading strategy we could potentially find a best subset of features x that maximizes a likelihood of an observed signal by either engineering features manually or doing an automated search over feature space.

$$\underset{y \in Y}{argmax}\{p(Y=y)\prod_{i=1}^{d} p(X_i|Y=y)\}.$$

To build a generic robot that is capable to imitate any trading strategy we need not only a flexible function f but also to automate search for a best subset of features that maximizes likelihood of an observed signal.

Evolving Solution for a Turing-Complete Machine

Our first approach is based on two evolutionary algorithms and a theory of Turing-complete machines[3].

First, lets imagine that there is a machine which is capable to run any computable strategy from a fixed set of instructions, supported by this machine. In fact, we don't have to imagine such a machine because all modern computer architectures are Turing-complete[4] and capable to do exactly that.

Modern computer architectures are based on a complex set of instruction that is optimized to deliver a good performance for any possible task a computer might do. For purpose of our goal we don't need all instructions. Instead, we've created a simple virtual machine which supports a limited set of instructions but is still turing-complete, which means it can, in theory, execute any computable algorithm. Please refer to Table 2 to see a set of instructions our machine supports.

Now all we need is to find an ordered set of instructions, A, which, when delivered to this machine, produces desired observed signal Y for any sequence of daily stock/eft quotes.

If we have infinite amount of time and energy we will, in theory[5], arrive to a right set of instructions which match observed signal.

Of course, we don't have infinite amount of time and energy thus we need a better way to search for an optimal set of instructions.

$\mu + \lambda$ Genetic Algorithm

Genetic algorithms provide an effective alternative to a random search. The basic idea of this algorithm is simple and is demonstrated in Figure 3. The set of instructions, θ are translated to a binary string, an individual. A population of individuals is then randomly generated ((1) in Figure 3). A fitness function $g(f(X,\theta))$ is then applied to each individual ((2) in Figure 3). The more fit an individual is, the more likely it is to be selected to be part of the next generation. Next generation is created from a sample of best individuals by applying crossover and mutation ((3 and 4) in Figure 3) to pairs of best individuals from a previous generation. Then the cycle repeats. Eventually, an individual is found that encodes an optimal solution.

Covariance Matrix Adaptation Evolution Strategy

Besides Genetic Algorithm we tried a more modern derivative-free method for numerical optimization, CMA-ES[6]. We did not change the algorithm. Since in CMA-ES candidate solutions are sampled according to a multivariate normal distribution in $\mathbb{R}^n$ we had to turn a vector of real numbers that represents an individual into a binary set of instruction to our virtual machine using rounding to a nearest integer method.

[3] https://www.cs.virginia.edu/~robins/Turing_Paper_1936.pdf
[4] https://en.wikipedia.org/wiki/Turing_completeness
[5] https://en.wikipedia.org/wiki/Infinite_monkey_theorem
[6] https://arxiv.org/pdf/1604.00772.pdf

LSTM Network

Our third model is based on Long Short-Term Memory Recurrent Neural Network with enough cells to learn any strategy.

For general-purpose sequence modeling, LSTM networks has proven stable and powerful tool for modeling long-range dependencies[7].

We used a simple architecture displayed in Figure 4. We took 60 quotes, preceding each trading day and turned this sequence into a 60x4 tensor that serves as an input to our network. Next the network applies a series of linear and non-linear transformations at each layer and the final layer produces a vector of 3 real numbers. We turn this vector into probabilities by applying a softmax function[8]. Finally we use argmax function to choose an action with highest probability, one of:

- 0 - Sell
- 1 - Buy
- 2 - Do nothing

To optimize our model we've used Adam[9] algorithm with a cross-entropy loss function[10].

Results

To benchmark robots we trained each model against all our test strategies, from easiest to the hardest and looked at how far each model can get us. We assume that if a model cannot discover patterns governing a simple strategy it will not be able to learn a more complex one.

A table with results summary can be found in Table 3. To calculate scores we've ran each model against 100 randomly chosen time frames and used scoring function to measure difference between true signal and a simulated one:

$$recall(\eta) = \frac{length((actual\_signal = \eta) \cap (simulated\_signal = \eta))}{length(actual\_signal = \eta)}$$

$$difference = \frac{3}{\frac{1}{recall(0)} + \frac{1}{recall(1)} + \frac{1}{recall(2)}}$$

where $actual\_signal$ and $simulated\_signal$ are both sequences of $y \in Y = \{0, 1, 2\}$, a true signal produced by a strategy and a simulated signal generated by a model, correspondingly. Then we averaged differences across all time frames

As you can see from Figure 5 and Figure 6 both, $\mu + \lambda$ and CMA-ES models ware not capable to learn the simplest strategies.

The LSTM model was much better in matching a signal of a simplest strategy (Figure 7) but failed to get us through a more complex Gap up strategy (Figure 8).

Conclusion

We have not found a single machine learning approach that can be used to simulata all possible trading strategies. We think that this might be a NP-hard problem, because searching for the solution that matches the signal can exceed polynomial time. If a right solution exist we can verify it in polynomial time though. To better demonstrate our conclusion we decided to include Figure 9 in our report. It shows a simple search

[7] https://arxiv.org/pdf/1308.0850.pdf
[8] https://en.wikipedia.org/wiki/Softmax_function
[9] https://arxiv.org/pdf/1412.6980.pdf
[10] https://en.wikipedia.org/wiki/Cross_entropy

space that consist of 3 dimensions only. Each cell is a potential solution that is encoded using 3 bytes. A solution might represent a set of instructions to our virtual machine or a set of parameters of our neural network. Only one solution will produce a signal that matches trader's strategy ("111" in Figure 9). If we simply enumerate all possible solutions it will take 2^3 evaluations to get to a right solution in the worst-case scenario. A genetic algorithm reduce search time but we have not found this reduction sufficient enough to turn this problem into a P-hard. While in Figure 9 we show only 3 dimensions, most real-life strategies require parameters of much higher dimentionality. For example, an algorithms of the first trading strategy can be coded for our virtual machine using about 100 bytes. It will take $4 \cdot 10^{22}$ years for a random search to arrive to this solution in the worst-case scenario if we need 1 second to evaluate each solution.

Our LSTM model uses a better optimization technique, that is based on a gradient descent. While it is a good alternative to an undirected search over parameter space that genetic algorith uses, this method has its own weaknesses. One of the key challenges of this approach is minimizing highly non-convex error functions and avoiding getting trapped in numerous suboptimal local minima[11]. Search space of this problem is non-linear, highly non-convex and rugged.

Code

All code used to generate models, plots and report related to this work can be found in https://github.com/VolodymyrOrlov/MSDS6306_project2

Figures and Tables

Variable Name	Description
Date	Date when data was recorded
Open	The first trading price recorded when the market opened on the day
High	Highest price at which a stock has traded that day
Low	Lowest price at which a stock has traded that day
Close	The last trading price recorded when the market closed on the day
Volume	Total number of shares traded for the day, listed in hundreds
Company	A unique alphabetic name which identifies the stock

Table 1: List of variables.

Instruction Code	Description
0010	set a value to a specified address in memory
0011	add two numbers
0100	divide one number by another
0110	unconditional jump to a specified position in code
0111	jgz jump to a specified position in code if value is less than 0
1000	compare two values, choose the maximum (max operation)
1001	compare two values, choose the minimum (min operation)
1010	increment a number by 1
1011	decrement a number by 1
1100	move a value specified by one address to another address
1101	compare two values, set 1 to an address of the biggest one
1110	compare two values, set 1 to an address of the smallest one
1111	halt the program and exit

Table 2: Instructions set, supported by our virtual machine.

[11] https://arxiv.org/pdf/1609.04747.pdf

Model	Trading Strategy	Test Score
$\mu + \lambda$ GA	Simple Moving Average Crossover	0.22
CMA-ES	SimpleMoving Average Crossover	0.40
LSTM	Simple Moving Average Crossover	0.88
LSTM	Gap up strategy	0.37
LSTM	Breakout strategy	0.56

Table 3: Test scores of all three models.

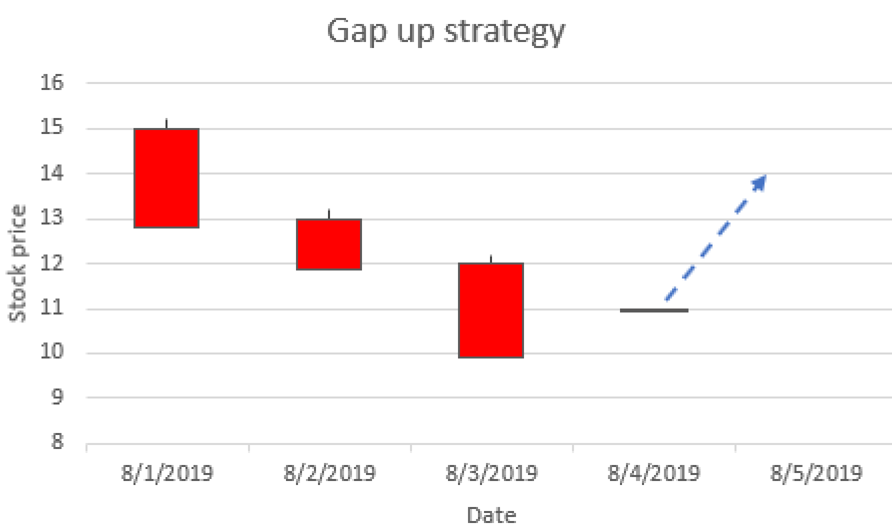

Figure 1: Gap up strategy.

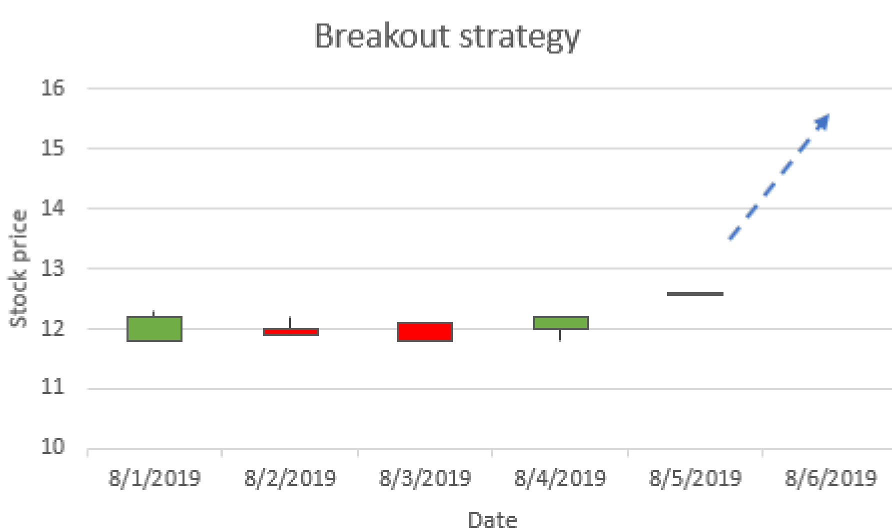

Figure 2: Breakout strategy.

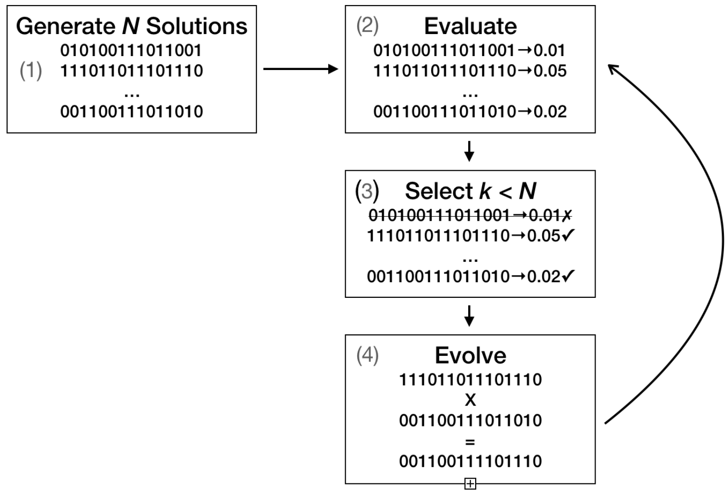

Figure 3: Outline of a simple genetic algorithm.

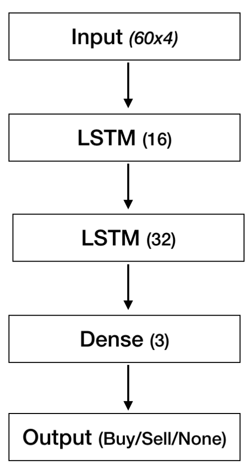

Figure 4: An architecture of the RNN network.

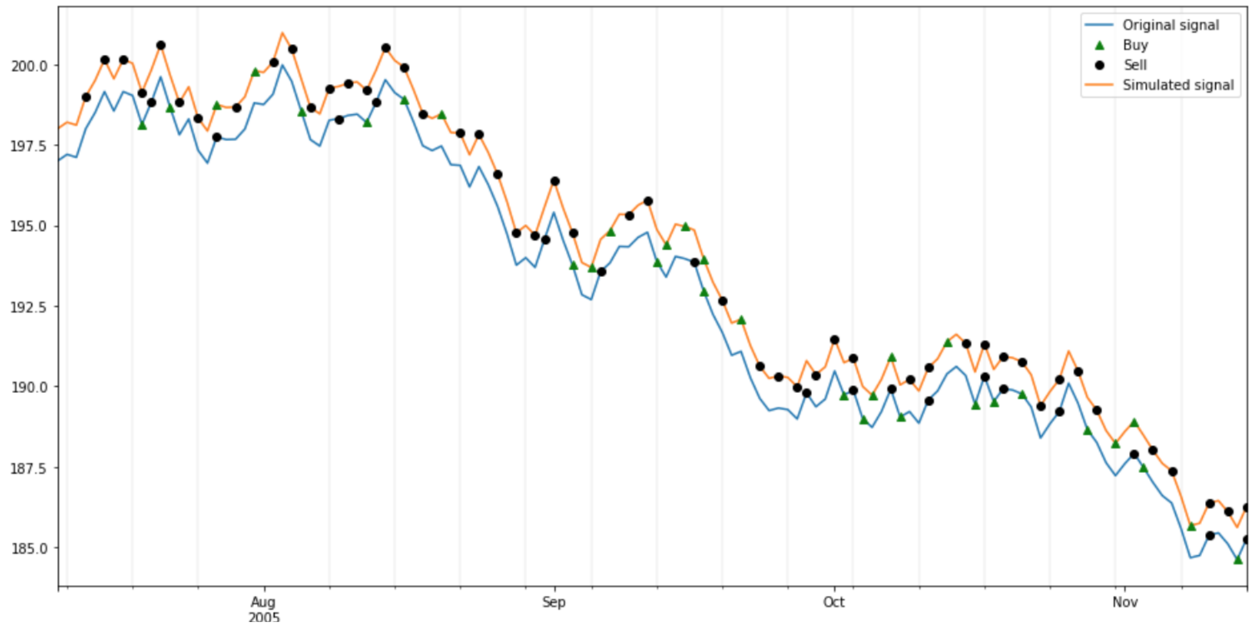

Figure 5: Signal generated by $\mu + \lambda$ GA model compared to a signal produced by a simple Moving Average Crossover strategy.

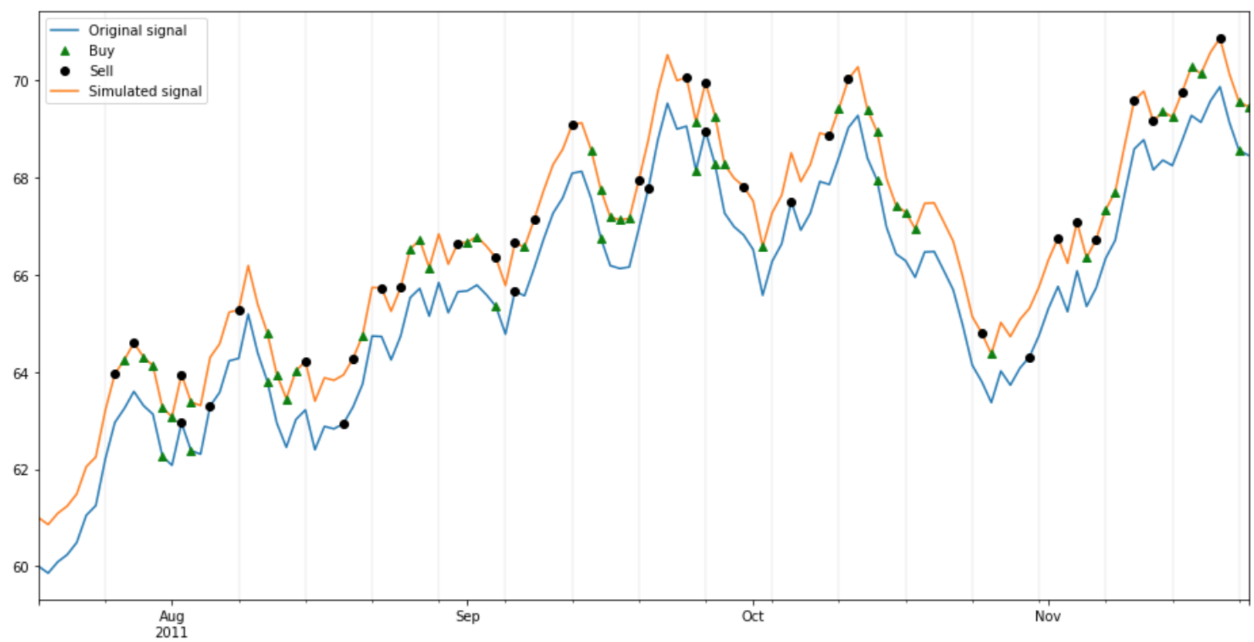

Figure 6: Signal generated by CMA-ES model compared to a signal produced by a simple Moving Average Crossover strategy.

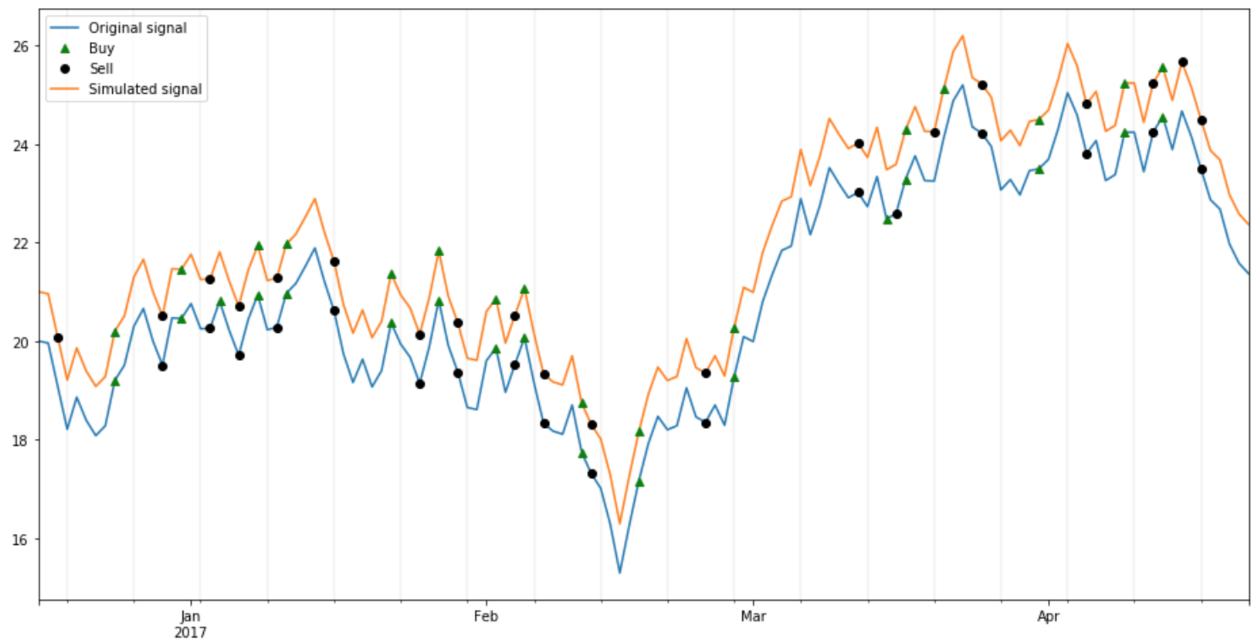

Figure 7: Signal generated by LSTM model compared to a signal produced by simple Moving Average Crossover strategy.

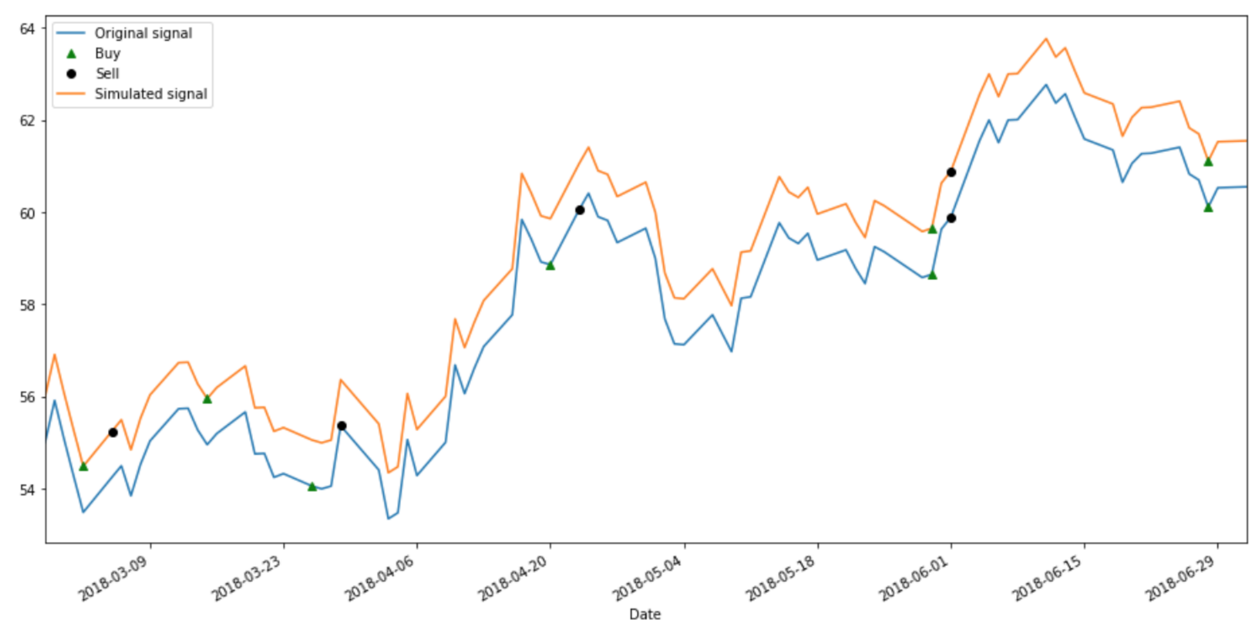

Figure 8: Signal generated by LSTM model compared to a signal produced by Gap up strategy.

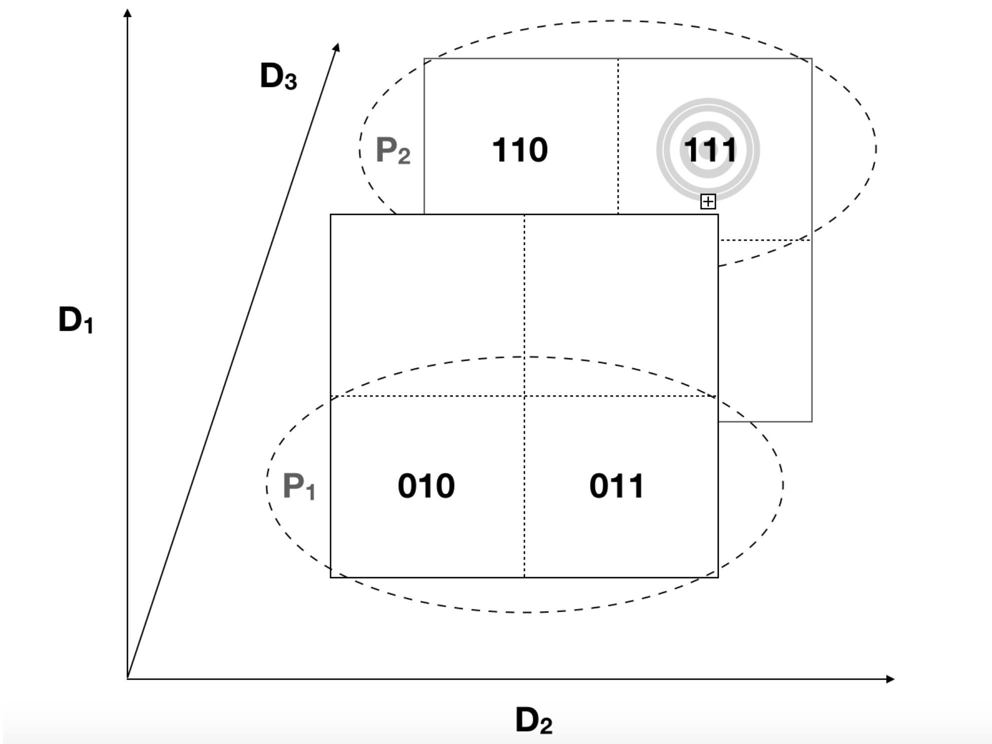

Figure 9: An example of a search space with only 3 dimensions. Each cell is an individual encoding a solution. P_1 and P_2 represent different populations, produced by a genetic algorithm.