DEVELOPMENT OF STOCK PLAYERS THROUGH LEARNING CLASSIFIER SYSTEM IN "REAL" MARKET

Gianmarco Sabbatini
Lorenzo Costantini
Matteo Mazza

May 3, 2019

Prof: Pietro Terna

1 Abstract

The goal of the work is to carry out an *agent-based* simulation of the stock market trend, using agents that operate differently: the most of them will make random decisions, while the others will be trained with a *learning classifier system* (LCS). After that we will study if the strategy adopted by the agents trained with LCS is better than the others.

2 Introduction

Most social systems involve complex interactions among many individuals. The sum of the simple interaction among the agents cannot explain the behaviour of the global system. For this reason, the traditional economy have only partially succeeded in extracting general features from simplified human behaviour.
Finance is one of the field where traditional economy approach fail. In fact, many of the foundations are in a state of confusion and new theories raise.
A way that is used to study these systems is the *agent-based* simulation. Using sophisticated mathematics and computational tools is possible to find out some *macro features* emerging from the interaction of individual strategies.
In this work we will resume what was thought by Arthur and Holland, at the end of the 80s at the Santa Fe Institute (SFI). Their idea was to populate the financial market of agents with different strategies. The winning ones are kept and replicated while the weakest ones are discarded. In this way, new space for new methods is created. The article to which we refer, however, is successive and written by LeBaron([2]), one of the co-authors of Holland and Arthur.

What we propose to do in this work is to reproduce the SFI stock market with the addition of the book. The book is a prospectus that collects the purchase and sale proposals related to a specific instrument. The order book is constantly updated on the basis of new orders placed and is divided into two parts: on the left side ther are the purchase proposals, while on the right side there are the sales orders. The proposals are arranged in ascending order, the transaction takes place when the major element of the left side of the book has a value greater than or equal to that of the lower element of the right side. In the event that the previous requirement is satisfied, the price of the stock that is exchanged is the once fixed by the agent who is selling.
As in the real exchange, where orders are canceled at the end of the day, the book is also reset in the simulation after a certain number of cycles. Each book is related to a single title. In the work the agents will act on a single title.
The training phase of the LCS algorithm is performed on data taken from a real stock market. The training is carried out offline: the training phase is carried out before the simulation starts.
Agents will have the opportunity to buy and sell the stock at any time, assuming they have both money to buy and stocks to sell. Each day each agent acts one time. Every transaction involves one stock. To change the volume exchanged by a certain type of agent is possible to modify the number of agents of that type.
In this work it has been studied the behaviour of a single stock.

3 Learning Classifier System

Learning Classifier Systems (LCS) are a model of *ruled-based machine learning* that contains a genetic algorithm (discovery component) and a learning component (supervised, unsupervised or reinforcing). These systems aim to identify a set of rules that depends on the specific situation; the derived rules act collectively in order to make predictions.
There are multiple implementations of the algorithm. In this work was chosen Michigan-style with offline supervised learning. The name derives from the fact that this type of algorithm was conceived by John Holland while working at Ann Arbor University in Michigan.
The steps of the algorithm are shown below:

Enviroment: The environment is the source of data upon which an LCS learns. Each training instance is assumed to include some number of features (market data with which to make the forecast) and a single endpoint (action). For Michigan-style systems, one instance from the environment is trained on each learning cycle.

Rule/Classifier: A rule is a context dependent relationship between state values and some prediction. Rules typically take the form of an IF {condition} THEN {action}. The rule is only applicable when its condition is satisfied.

Given binary data LCS traditionally applies a ternary rule representation: rules can include either a 0, 1, or # where the "don't care" symbol (i.e. "#") stands for "wild card", a way of generalizing the rule.
The classifier is the rule with the addition of associated parameters, some of the most common are: quantity (number of rules equal to each other), match count (number of times in which the rule was applied), correct count (number of correct application of the rule), accuracy (ratio between correct count and match count), fitness (parameter that influences the reproduction of the rule).

Matching: from the environment a rule is taken and compared with all the rules in the population, where the comparison is made only on the conditions and not on the action. The part of rules that overcome the match ends up in the MatchSet. From this two subsets are generated, CorrectSet and IncorrectSet. In the first one end up the rules that have the same end point of the rule coming from the environment, the remaining end up in the second set.

Covering: if after the match phase, there are no elements in the CorrectSet, the covering mechanism intervenes. This takes the rule from the environment, randomly chooses the features to be copied into the new rule and replaces the others with the #, reporting the unaltered action.

Parameter updates: the rule parameters of any rule are updated to reflect the new experience gained from the current training instance. The rules of CorrectSet will increase the correct count and so fitness and accuracy, vice versa for IncorrectSet.

Subsuming: if we have two rules that have the same accuracy but a different level of generality and moreover one is a specification of the other, the less general is eliminated and the quantity of the more general one is increased.

Genetic algorithm: at this point to introduce new rules in the population, we use a strongly elitist genetic algorithm that makes "reproduce" only the rules with greater fitness.

Deletion: the population of the algorithm rules is limited. For this limit, the rules with the lowest fitness are canceled.

Condensation: after the previous phases the redundant rules and those generated for a short time are canceled. This because they have not therefore been subjected to a sufficient selection process. In this way a group of rules is obtained with a smaller number that increases both the interpretability and the performance during the decision phase.

Prediction: the status of the problem is presented to the algorithm as a vector of features. In a first matching phase we compare the vector of the features with those of the rules. A MatchSet is created with the rules applicable to the situation. To decide what action to take, look at the components of the MatchSet, especially their end point. The agent will perform the action supported by the most rules.

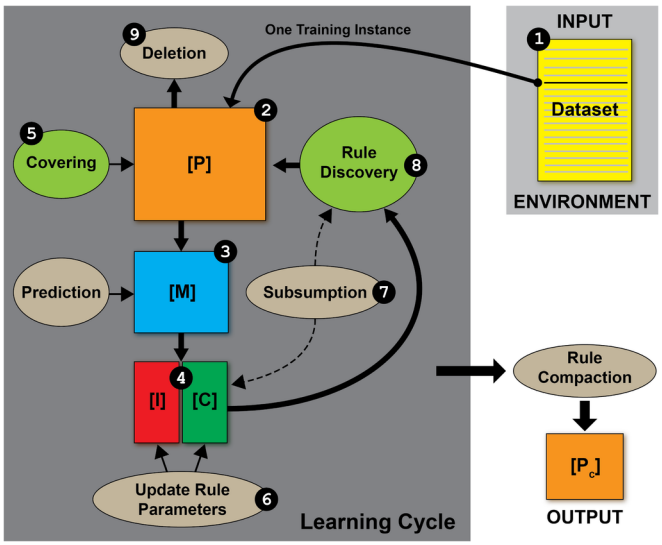

Figure 1: Schematic representation of the functioning of an LCS algorithm

3.1 Choice of rules

Our departure data can be taken from this different ways:

- take data from a market composed of agents acting randomly, making sure there are rising and falling bubbles;
- take data from a real market;
- create the time series by ourselves with some particular features (this series must not be periodic and it has to have bubbles).

In this work we have used the second option.
To create the environment, the following rules have been applied to the source data:
In the Table 1, p_i is the price of the stock at the tick i. T-period MA is the moving average calculated with a period T using this equation:

$$T - period\ MA = \frac{1}{T} \sum_{i=-T}^{0} p_i \tag{1}$$

vol_i is the volume at the tick i. The volume is an indicator of the liquidity of a given financial asset. A considerable increase/decrease in the volume is generally followed by a strong change in the price of the stock. So the volume is an index of the increased/decreased interest of investors for the stock. volmedio is the average of all the volumes of the market considered while devstd_vol is the standard deviation of the volumes. The eleventh bit is the action: if the

Bit	Condition
1	$p_i >$ 5-period MA
2	$p_i >$ 10-period MA
3	p_i / $p_{i-2} > 1.02$
4	p_i / $p_{i-2} < 1/1.02$
5	10-period MA_i - 10-period $MA_{i-5} > 0$
6	10-period MA_i - 10-period $MA_{i-5} < 0$
7	p_i - $p_{i-10} > 0$
8	p_i - $p_{i-5} > 0$
9	vol_i - volmedio > devstd_vol
10	volmedio - vol_i > devstd_vol

Table 1: presentation of the rules

price at the present time is greater than 1.5% of the price in five days then the agent sells a stock, vice versa buys. The agent passes if neither of these two conditions is satisfied.

4 Preparing the data

To make our predictions we use the difference between the price of the title at the current time, and the one five days later. The market trend depends on the day of the week, because of the costs of the commissions, so we have to make a comparison between the same days (for example the difference between two monday). Our timeseries present the daily closing price of the stock. The market, however is closed in some special days like Christmas or New Year's day and to make an automatic system of creating the environment is impossible. To overcome this problem we have to adjust the data. With a short script we have found the days left and added them artificially. The closing price of the these days is the price of the previous day, beacause the best predictor of tomorrow is today. Regarding the volume we have associate the avarage volume. In fact, a large volume means lots of transactions that implies usually a sensible difference in the price.

5 Model tests

At the beginning, before the implementation of the artificial market, the model has been tested on the real time series to understand his behaviour and to understand how the parameters work. Two test were made.
In the first the algorithm was trained on the whole dataset and then tested on the same data. In the second the algorithm was trained only on the 70% of the data and tested on the other 30%. The most important issue to understand was to verify that the algorithm could recognize the stock's price big variations. To

do this, we have create two counter. The once called "correct" was incremented when the decision of the algorithm was 'Buy' or 'Sell' and matched with the correct action. The other called "incorrect" was incremented when the decision of the algorithm was 'Buy' and the right action was 'Sell' and vice versa. We were not interested in the case in which it decided to pass even if the correct action was to play in some way. This means that the important things in this moment is not win a lot but not lose.
We can't give precise results because of the stochastic part of the LCS' training, either in the covering and either in genetic algorithm. Generally in the the first way to test the LCS the results were around 60% of correct actions, while in the second way we obtained between 54% and 57%. Another things that stands out from these tests and don't surprise is that the algorithm works much better on time series which have a unique or a few trends in the long period. Titles which presents lot of variations are more difficult to be understood from the LCS, because the component of noise is higher than the signal.

6 The Artificial Stock Market

6.1 The actors

All the agents that will act in our market present the same form. They're object with four principal attributes:

- Cash: represent the amount of money of our agent. It can be either positive or negative number (if he hasn't enough money to buy someone else lend him money and he goes negative);
- Stock: represent the number of stocks owned buy the agent. It can be either positive or negative number (there is someone other that gives him stock to trade if he hasn't);
- Gain: this quantity is calculated at the end of the program as the algebraic sum of the cash and the number of stocks multiplied their value at the end of the running. It is a resume of the behaviour of the agent.
- Id: is an identity number that identify the single agents and is important because it connects the the agent to their offer in the book.

In addition there are also two important common methods:

- Action: with this method the agent decide what to do among the option 'Buy', 'Sell' or 'Pass', with a logic that depends from its type;
- Price: with this method the agent decide the price at which buy or sell;

6.1.1 Random Agent

The Random agents are the ones which represent the larger part of the agents of our market. They're simple agents that decide to act randomly.
In their Action method they extract a random number between 0 and 1, and if this number is minor than 0.45 they buy, if it is included in the range between 0.45 and 0.9 they sell otherwise they pass.
Their Price method take in input the closing price of the previous day. To this one we add a number extracted from a gaussian distribution with average 0 and as variance the 1% of the input price.
A stock market composed only from this type of agents take the form of a real market even if is done by only noise and no signal. In the following picture we can see the evolution of a stock price in the time driven by random agents.

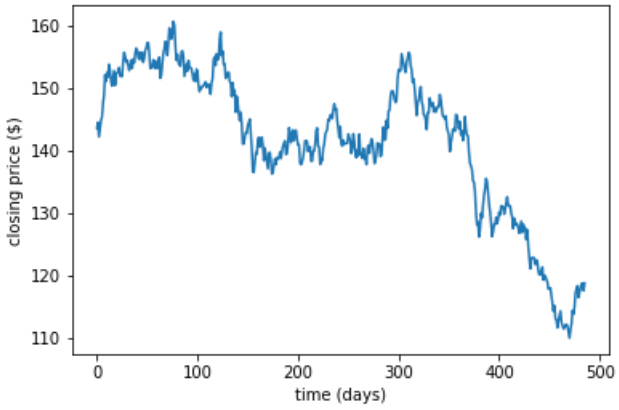

Figure 2: Market in which act only 200 Random Agents

6.1.2 Arbitrajeur

Now we have a market that qualitatively goes like a real market but there's no signal below only the noise made by Random Agents. For this reason if the LCS agent try to play in this market all its training will be useless because even if it has understood something about the signal, this would be lost.
To avoid this problem a new agent was created: the arbitrajeur. This agent has the task to "transport" the signal from the real stock market to the articial one. To do that it has to know how the real market evolves.
In its Action method it takes in input the real price and the price of the artificial market. It will buy if the first is higher than the second, and will sell in the opposite case.
Its Price method works in a simple way. If the action is buy, it returns a price that is the 0.1% bigger of the smallest price in the selling queue of the

book. Same reasoning when it has to sell. This because his role in the game is fundamental and its action must be realized.

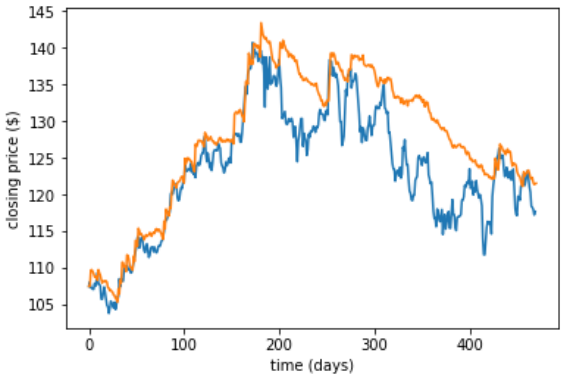

Figure 3: The orange curve is the simulated market and the blue one is the real market. Here only five arbitrajeur acts among 200 Random Agents.

We can see that a few arbitrajeurs on the total number of agents guarantee a trend of the artificial market very similar to the real one.

6.1.3 LCS Agent

This is the "clever agent", the one which we aspect will act in a better way then the others.
First of all we have to split the times series in two parts. The first one will contain the 70% of the data (train set) and the remaining data will compose the second one (test set). Unlike the agents described above this one has a one more method called Train. In this function the LCS agents is trained on the first part of the data. The training is made using a library called XCS. This method takes in input the train set and returns the model that will be used from these agent to takes his decision. The model contains a list of rules with this form: the first ten bits represents the status of the market and the last position there is the action to do. Associated to the rules, the model has also some parameter: fitness and numerosity. The training is performed immediately after the creation of an agent of this type. So at the moment in which the simulation of the real market begins, every LCS agents have their own model.
Like the other actors the decisions are taken in the function Action. This receives in input the current status of the market coded in the same way as the condition of the rules. At this point, only the rules that matches with the market status are considered. The process of matching consists in comparing the bit of the status with the corresponding one in the rule. After the matching process the survived rules vote. Each remaining rule has a different action and to decide we

count how many rules there are for each action weighted with their numerosity and fitness. The returned action will be the most voted.
The method Price of this agent works as the one in the arbitrejeur method.

6.1.4 The proportion of the agents

The proportion of the actors is very important in this kind of market. If the number of arbitrajeur agents is too high then we are looking at the real market and no at an artificial one. In fact as shown in figure 4 an excessive number of arbitrajeur follow exactly the behaviour of the real market.

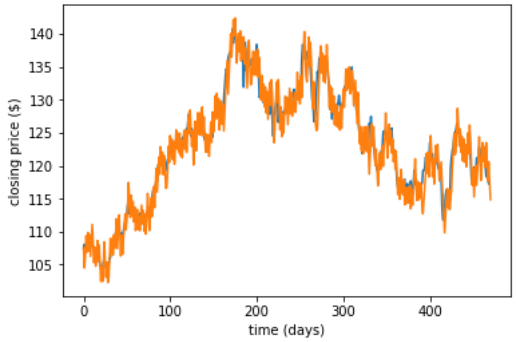

Figure 4: Market in which act only 200 Random Agents and 70 arbitrajeur

Considering that the training of LCS agent is done in a market in which it is not present, also in the simulated market its action must be irrelevant. If the number of the intelligent agents is relevant then their rules are no more "right" because the market has been modified too much and is no more similar at the real one. This concept is shown in figure 5 where the simulated market (orange line) is far from the real market (blue line).

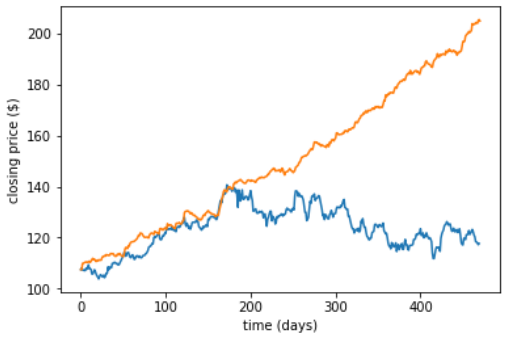

Figure 5: Market in which act only 200 Random Agents and 2 arbitrajeur and 20 LCS agents.

In conclusion, to obtain a fine simulated market we must have:

- big number of Random Agents;
- one arbitrajeur each 20 Random Agents;
- a few LCS Agents.

In our market the LCS agents, when they are more than one, is distributed among the Random agents.

6.2 Calculate the gain

The gain is an attribute that is own by each type of agents. It represents the performance of the agents in term of money. Since the mean gain of a random agent is zero and the arbitrajeur's gain is not relevant (because it is only a link between the simulated market and the real one), the only gain we are interested in is the one of the LCS agent. In this work this quantity has been calculated in two different ways.
In the first way each time the agent buy or sell a stock his attribute cash is decremented or incremented by the price of the action in that moment. In the same time also the the attribute stock is modified: it is incremented by 1 every time the agent buys a stock and decremented in case the agent sells. The final gain is calculated using the following equation:

$$gain = cash + stock * p_f \tag{2}$$

where p_f is the final price of the stock at the end of the simulation. In this case, the agent could accumulate stocks and settle accounts at the end of the simulation.
In the second option the agent has to close the transaction after five days. This means that if at a certain time t it decides to buy after five days it will sell the same stock. The same is valid in the case in which the action at time t is sell, after five days it has to close the transaction buying.
So the gain in case of at time t the action is 'Sell' the gain will be: $gain = p_t - p_{t+5}$ else if the action is 'Buy' the gain will be: $gain = p_{t+5} - p_t$.

7 Results

Consider a simulated stock market with 500 random agents, 8 arbitrajeur and 4 LCS with separate training.
We have tried this algorithm on the time series of the following companies: Amazon, Toyota, Fiat, Volkswagen, Terna, Apple and Microsoft.
Looking at the gain calculated in the first way it's positive and in average in two years and a half it's equal to the value of the action.
Considering this quantity as a proxy of the capabilities of the algorithm we can conclude that it can learn a pattern followed by the time series.

If the LCS agent has to close the transaction after five days strictly, so using the second way to calculate the gain, the results obtained are positive but closed to zero so minor than when was calculated in the first way.

8 Especial results

In the end, the interesting thing was to understand if the algorithm could find a pattern among different stocks. To do this, the model was trained on a company's time series and tested on another company's time series.
The results are in some ways interesting. The algorithm works in a good way if trained and tested on company's of the same type, for example trained on Fiat and tested on Toyota or trained on Apple and tested on Microsoft.
On the other hand the same is not true if trained and tested on two companies of different sectors, for example trained on Toyota and tested on Amazon.
From what observed, we can conclude that similar company trend's time series have a common pattern below learned from LCS. Different company trend's time series are not regulated from the same pattern.
So we can conclude that the LCS algorithm is able to catch the pattern below the trend of a stock but these are not general pattern applicable to every time series.

References

[1] W Brian Arthur. Asset pricing under endogenous expectations in an artificial stock market. In *The economy as an evolving complex system II*, pages 31–60. CRC Press, 2018.

[2] Blake LeBaron. Building the santa fe artificial stock market. *Physica A*, 2002.