Learning on Positive Experience

Nikolay Ryadchikov

December 15, 2011

Abstract

Nowadays, with broad overall computerization one of the main goals of science is to provide with efficient algorithms that can teach machines to learn and make decisions. In the paper, an approach for organizing the learning process is introduced. The main idea is to split all information obtained by a machine repeatedly undertaking some actions into two categories: positive and negative experience. The positive experience is formed by the cases when the action chosen by an algorithm resulted in a clear and observed success, while all the other cases correspond to negative experience. The former is then to be concentrated on in the decision-making process, as long as it gives an explicit rule on action choice while the information provided by negative experience decreases with the number of actions available. The approach is then applied to the selection of portfolio weights on the stock market.

Theoretical Approach

The approach is developed for the following framework. The machine is to choose an action as a response to the state of the system. After action is chosen, an observable target variable clearly shows whether the action has been successful or not. Namely, if the aim of the researcher is to maximize some kind of performance, then positive change in the corresponding variable is supposed to be a success, negative – failure, with constant change not really important where to go. Then, the actions that have led to a success form positive experience along with the states when those actions have been chosen. The rest goes to the negative experience. Now, consider the action space, the range of decisions the machine can make. Suppose the machine acts in a discrete action space. Let it choose between two actions in each state only. Assuming that optimal action is a function of state, at each period the machine searches over its memory for information about current state. Suppose, it has already have dealt with similar state before. Then, it analyzes the action that was chosen that time and the resulting outcome. If the decision resulted in a success, then the action is likely to provide us with a success in current period. Otherwise, the negative experience tells us that it is not reasonable to do the same action now, so we are to choose the other, and the latter is likely to result in a success. Hence, negative experience is as good as positive one. Let us increase the action space making it finite but greater than two. Then, past positive experience yet provides us with an explicit decision – to choose the same action that managed to provide us with a success in a similar state before. What about negative experience? Actually, the fact that in quite the same state the machine chose an action and it resulted in a failure, gives us only the advice not to use it now. However, what action among the others to choose? Yet the negative experience have provided us with some information by excluding one way of action, it does not give us an explicit decision. So, the value of negative experience falls with a wider action space. Let us increase the action space further and consider an infinite action space. As well as previously, the positive experience enables the machine to make a decision basing on accumulated data. The negative experience still tells us not to use the same action that we did before, however, now it corresponds to removing a single point from an infinite action space. Even if for the current state we have accumulated a high number of similar cases when actions undertaken resulted in a failure, it is a finite number of points to be excluded from the decision-making, yet leaving an infinite number of actions to be considered. So, in an infinite action space negative experience gives infinitely small amount of information. As a consequence, in order to construct a learning algorithm for a decision-making process, we are to concentrate on the positive experience, as long as it shows us clearly how to achieve a success.

Selecting an Issue

As a practice for described approach the following issue was selected. The machine is to simulate trading activity via taking past data on stock returns and constructing a strategy for continuous portfolio re-balancing. The data consists of monthly log returns of 10 stocks from 01/03/1994 to 12/01/2006 taken from 'Statistical models and methods for financial markets' by T.Lai and H.Xing. The state of the system is described by a vector of returns, the action space is a vector of chosen portfolio weights. So, the algorithm is expected at each step of the time series to choose a vector of weights basing on last vector of returns and accumulated positive experience. The outcome of an action is supposed to be a success if next period's portfolio return is positive, otherwise the experience is considered to be negative. Clearly, the aim of the algorithm is to maximize the cumulative return on the evolving portfolio over the range of data considered. The original data is split into two parts. First 100 of observations are assigned to train data, the rest 56 are left for evaluating the performance of the technique. So, the algorithm is to use the train data to collect some positive experience and then exploit the experience for trading over the second part of the data.

Detailed Application

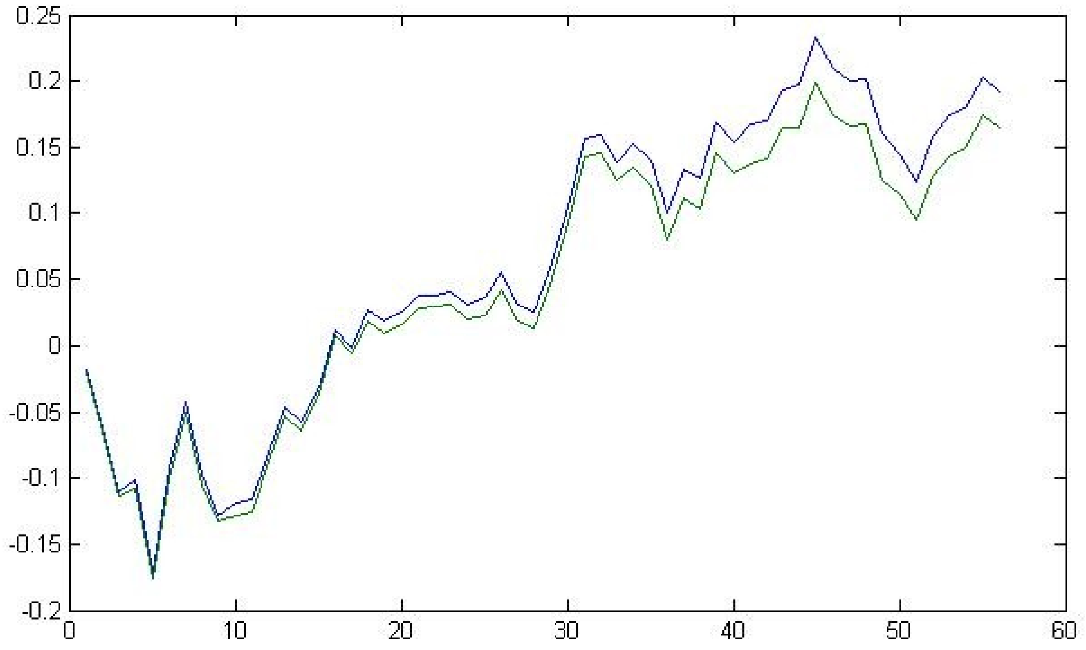
In general, the algorithm loops over the train data round and round until the cumulative portfolio return function converges within some precision on the train data range. At each step of the loop, the portfolio weights are initialized as uniform random vector with elements summing up to unity. Then, starting from the second period of the train time series, the portfolio return is evaluated as a cross-product between last period's chosen weights and current period's stocks' returns. If the return appears to be positive, the previous period's state and corresponding action - vector of weights - are added to the positive experience matrices, one for each of the two. In order to avoid overcrowding of the data sets, starting from the second loop of the outer cycle, before adding a case for positive experience data sets, the algorithm checks whether we already had a positive experience for the same state at previous loops, and if yes, it keeps the action that provided a greater return. As an outcome, there would be two matrices, one containing vectors of states for which we already had positive experience and the other containing corresponding vectors of weights that brought us the plausible result.

Then, at each step of the inner cycle, we are to choose new optimal weights according to current observed state. If the number of accumulated positive cases is fewer than some initially fixed value (call it *min_exp*), the algorithm as before generates a random uniform vector of weights summing up to one. If we already have enough positive experience, the machine undertakes the following procedure. For each state in the positive experience matrix for states it computes the distance as the sum of quadratic deviations between elements of the two states: returns on the same stock in the past positive example and current state. Once again, to let the program run faster the machine selects *min_exp* closest states among the positive, those that have *min_exp* smallest distances, though excluding states equal to the current one. The latter states can easily appear since we are continuously looping around the same data set. Finally, the optimal weight vector for current state is evaluated as a weighted average of *min_exp* weight vectors corresponding to the selected most similar states, with weights being inverse proportional to the distance. Note that we have dropped completely same states at the selection stage in order not to have infinite weights here. Consequently, at each step of the inner cycle, if the machine faces lack of positive experience, it chooses random portfolio weights, and when it already has enough information the weights vector is a weighted average of actions undertaken in the most similar cases before, remembered due to resulting positive return.

At the end of the inner loop, the program derives the cumulative portfolio return function and computes the change at this step of the outer cycle as a maximum absolute value of the difference between the current cumulative return function and the other saved after the previous loop. At the very first loop of the outer cycle the previous cumulative return function is initialized to be zero elsewhere. If the change after a loop of the outer cycle is below some selected tolerance, the machine increases the count of no-change loops, otherwise it is set back to zero. When the number of consequent no-change loops exceeds some fixed integer, the algorithm is supposed to eventually converge.

After the convergence, the two accumulated matrices for positive experience states and actions are used to make decisions over the test data range. Actually, the algorithm repeats the same procedure as over the train data, except for the fact that it does not add any new information into the positive experience data sets any more. So, it chooses the weights at each step of the time series to be the weighted average of *min_exp* positive states closest to the last observed state. Then the resulting cumulative return function is a measure of method's efficiency.

Results



The above graph shows the resulting cumulative portfolio return function over the test data range (blue line), compared to uniform portfolio performance (green line). Actually, the outcome is not certain as long as it depends on random vectors of weights generated by the algorithm in the beginning of its work while there is not enough positive experience. So, though above picture was noticed to be quite typical among consequent repeats of the algorithm, one particular implementation significantly depends on chance. Notably, the above graph was derived by repeating the main algorithm by 1000 times and taking the mean of all resulting cumulative return functions at each time period. So, the algorithm provides with a slight superiority over the uniform portfolio performance. One may argue the latter not to be a proper benchmark, however, it is one of the most parsimonious ways to judge on portfolio's performance.

Meanwhile, different methods of choosing the optimal weight vector were used and compared, among them: scaling the weights of closest states used in computing the weight vector not only inverse proportional to distance, but also proportional to return faced; running a linear regression of weight vector on state vector over the positive experience data set; choosing only the closest case in positive experience history for current portfolio weights choice; selecting one action in positive experience data set with probabilities on each positive experience case inverse proportional to distance and/or proportional to resulting return. However, none of the methods above appeared to work well or better, so it was decided to keep the model as it used to be.

Conclusion

To begin with, the theoretical approach described is supposed to be quite interesting and promising. However, the current application is, perhaps, not really suitable for the framework. Probably, the method would give better results if applied in systems when the next period's state depends not only on the previous state but on the action chosen too. Furthermore, the implied assumption that at each period of the returns' time series the optimal weight vector is a function of last returns can be considered to be invalid and irrelevant for the task. So, there is much work to do to improve the application of the main idea. As a conclusion, the paper does not pretend for originality and the author sincerely sends apologies if the approach has already been proposed by another researcher.