Machine learning optimal parameters for a channel breakout system for trading commodities

Martin Neal
Becky Engley

May 26, 2009

1 Introduction

The commodities market is an exchange where raw materials and agricultural products are traded. Commodities are usually traded via futures contracts. A farmer can sell a futures contract on his crop, months or even years in advance of the harvest, guaranteeing the price he will recieve at harvest time. A grocery chain can buy the contract, with the confidence that the price will not go up when the farmer delivers. Futures contracts protect farmers and buyers from unexpected price changes.

Investors can also buy and sell the futures contracts, attempting to capitalize on increases and decreases in price. If an investor expects an increase in the value of a commodity, he will buy a contract while the price is low, and then sell his contract at a later time, when the price is higher. In addition to buying before selling, it is also possible to sell a contract before buying it back, which an investor may do if he expects a commoditiy's value to decrease. Buying before selling is referred to as *trading long*; selling before buying is known as *trading short*.

It is often difficult for investors to determine when to buy and when to sell. For example, it is impossible to recognize a minimum in price, until after the price has increased above the minimum. Investors are left guessing as to whether the price will continue to increase, or immediately begin decreasing again. Guessing often proves to be unprofitable.

Momsen describes an automated trading system that attempts to predict trends in commodity prices, providing the investor with a guide to buying and selling. Momsen defines a pair of channel lines above and below the commodity's price. The lines are defined using the price history. When the commodity's value undergoes a change in trend, it crosses one of the channel lines, alerting the investor to the change. At this point, the investor can decide to buy or sell, appropriately.

Momsen's channel breakout system relies on a set of six parameters to define the channel lines and date ranges between which an investor may buy and sell. This paper aims to data mine commodity market prices, in order to learn better parameters for Momsen's system. We use two machine learning algorithms to search the parameter space; simulated annealing, and a genetic algorithm.

2 Related Work

Momsen's book, Superstar Seasonals, describes his channel breakout system as a model for predicting uptrends and downtrends. According to his system, a commodity may be traded between a range of dates specified by the static parameters *entry_window_open* and *entry_window_close*. A trade must be completed no later than the *exit_trade* date, a third parameter of the system.

Momsen defines an upper channel line and a lower channel line. The upper channel line is determined by the maximum high value over the previous m days. The lower channel line is determined by the minimum low value over the previous n days. Here, m and n are also parameters of the system and are different for each commodity. If trading long, the upper channel line is the entry line, and m is referred to as the *entry_threshold*. The lower channel line is the trail-stop line, and n is referred to as the *trail_stop_threshold*. If trading short, the channel lines are reversed.

The sixth and final system parameter, the *stop_loss_threshold*, serves as a safety net to prevent large losses. When trading short, the stop-loss is defined by the maximum high value over the previous q days, where q is strictly less than m. When trading long, the stop-loss is defined by the minimum low value over the previous q days, where q is strictly less than n. The trail-stop line protects profits, while the stop-loss line minimizes losses.

According to the system, if the current value of the commodity crosses the entry line, it indicates the beginning of a new trend, and the trade is begun. When the current value of the commodity crosses the trail-stop line, the trade is completed. More often than not, the cross of a channel line incorrectly predicts the beginning of a new trend. In these cases, small losses are incurred. However, when the system correctly predicts a long trend, the profits generated far outweigh small losses.

Over the past thirty years, Momsen has traded this system, with outstanding results. We intend to extend his work by machine learning optimal values for the six trading parameters.

3 Methods

Local search algorithms are used to maximize an objective function. Our objective function is our algorithm for the channel breakout system, which takes the six trading parameters as input, and returns the profit earned. Optimal paramers maximize the profit earned as a result of trading the system. We first discuss the algorithm for our objective function. Then, we present the search algorithms that we will use to explore the parameter space.

3.1 Trading the System

The channel breakout system processes financial market data for an individual commodity, over a single contract year. We describe our algorithm for the channel breakout system below, along with our data set.

3.1.1 Data Set

We have financial market data for fourteen commodities over a period of thirty years. Because the learning system is evaluated per contract year, each year comprises a single data point. We have approximately thirty data points for each commodity. The commodities include live cattle, pork bellies, corn, wheat, lean hogs, crude oil, unleaded gasoline, heating oil, and orange juice. Some commodities are traded during multiple seasons. The data includes open, high, low, and closing values for each commodity for every trading day.

The data set will be partitioned into two sets. For each commodity, we will reserve one third of the data points for testing; the remaining two thirds will be used to train the system. We will use the earlier years of data for training, and the more recent years for testing. We will not reserve any data for validation, as we do not intend to validate our predetermined model.

3.1.2 The Channel Breakout System

The algorithm for the channel breakout system takes in the six trading parameters, and returns the total profit earned. Our $TradeSystem()$ function is our objective function, the return value of which we are trying to maximize. It is used as the $value()$ function for the Simulated Annealing algorithm, and the $fitness()$ function for the Genetic algorithm, both defined in a later section.

```
TradeSystem(entry_window_open, entry_window_close, exit_date,
        entry_threshold, trail_stop_threshold, stop_loss_threshold)

  compute entry, trail_stop and stop_loss channels, using thresholds

  while(entry_window_open < current_date < entry_window_close OR in_trade)
    if(not in_trade)
      if(entry_channel is crossed)
        in_trade ← TRUE
        entry_price ← current_price
        stop_loss_value ← stop_loss_channel[current_date]
    else
      if(trail_stop_channel is crossed OR stop_loss_value is crossed OR
        current_date = exit_date)

        in_trade ← FALSE
        exit_price ← current_price
    current_date ← current_date + 1

  compute profit using entry_price and exit_price
  return profit
```

We may repeatedly enter and then exit a trade many times over the course of one trading year. When in a trade, we check every day, to determine if we should exit. When not currently in a trade, we check to determine if we should enter, until the entry window closes. When we enter, we compute the entry price based on the value at which we crossed the entry channel line. The stop-loss value is also based on this cross-point. When we exit, we compute the exit price based on the value at which we crossed the closer of the trail-stop channel and the stop-loss value. The profit for this trade is the difference between the entry and exit prices.

3.2 Searching the Parameter Space

Optimal parameters maximize the profit earned as a result of trading the system. We will approach this optimization problem using two different machine learning techniques: simulated annealing, and genetic algorithms. Below we present these two algorithms, along with an algorithm for a random learner, which we will use as a baseline comparison.

3.2.1 Simulated Annealing

Simulated Annealing combines the best of hill climbing and random walk heuristic algorithms. Hill climbing algorithms learn quickly, but they generally only find local maxima, because they never move downhill. Random walks are guaranteed to find the global maximum but take far too long to do so. Simulated Annealing combines these approaches, yielding both efficiency and completeness. We present the psuedo-code for the Simulated Annealing algorithm below.

```
SimulatedAnnealing(number_iterations)

    current ← 6 random parameter values
    max ← current

    for t ← 1 to number_iterations

        next ← successor(current, n, dist_type)
        ΔE ← value(next) - value(current)

        if(ΔE > 0)
            current ← next
        else
            current ← next, only with probability P(ΔE) · f(t)

        if(value(current) > value(max))
            max ← current

    return max
```

The *value*() function trades the system with the six parameters and returns the profit made (or lost). Current and next are both nodes. In this context, a node is a set of values for the six parameters. $f(t)$ is a linearly decreasing function of time; it decreases the probability of downward steps as time increases. $P(\Delta E)$ gives the probability of ΔE occuring. We compute $P(\Delta E)$ using a normal distribution based on the previous values for ΔE.

The choice of *successor*() function greatly affects the learning speed of the algorithm. Our *successor*() function adjusts n of the six parameters in current by a random amount, δ, which may be sampled from one of three distributions: a uniformly random distribution, a normal distribution, or a constant distribution of size one. The parameter *dist_type* specifies the distribution to use.

3.2.2 Genetic Algorithm

Genetic Algorithms model evolutionary processes. In our proposed implementation, a set of six system parameters represents an individual. Many individuals form a population which is repeatedly bred and then culled. Breeding swaps random parameters from two or more parents to create children. Culling evaluates each individual using a $fitness()$ function, and eliminates unfit individuals from the population. The $fitness()$ function returns the profit earned by trading the channel breakout system using the individual's six parameters. We present the psuedo-code for the Genetic Algorithm below.

```
Genetic(number_iterations)

    population ← createPopulation(size)

    for t ← 1 to number_iterations
        for i ← 1 to size(population)

            parents ← random-subset(population, size)
            children ← reproduce(parents)
            children ← mutate(children) with small random probability
            population ← add(population, children)

        population ← cull(population, threshold)

    return best-individual(population)
```

This generic algorithm leaves a lot of room for experimentation. The initial population size may vary. The probability of mutation may be changed. The mutation of an attribute may be uniformly or normally distributed. The *reproduce*() function may take between two and six parents. These parents randomly swap attributes to produce one or more children, which are added to the population. The population is then culled, which preserves a threshold number of individuals, removing all others as unfit. We intend to experiment with many of these variations.

3.2.3 Random Walk

Here we describe our random walk algorithm for use as a baseline comparison. The algorithm is similar to simulated annealing, in that the next node is chosen via a *successor*() function (where again, a node is a set of six parameters). Unlike simulated annealing, our random walk algorithm has no preference for climbing uphill. We present the psuedo-code for our random walk algorithm below.

```
RandomWalk(number_iterations)

    current ← 6 random parameter values
    max ← current

    for t ← 1 to number_iterations
        current ← successor(current, n, dist_type)

        if(value(current) > value(max))
            max ← current

    return max
```

As with simulated annealing, the *successor*() function adjusts n of the six parameters in current by a random amount, δ, which may be sampled from the same three distributions.

4 Experiments and Results

We begin by experimenting with variations on each of our learning algorithms. We will investigate the effects of differing types of successor functions, and experiment with modifications to our genetic algorithm, including altering the initial population size, the probability of mutation, and details of the reproduction function. After determining the best set of variations for each learning algorithm, we will use our algorithms to search for the optimal parameters for each commodity.

4.1 Successor Function

A successor function takes in a state (i.e., a set of six parameters), and modifies it in some way to produce a new state. Our simulated annealing, genetic, and random walk algorithms all use a successor function. In the case of our genetic algorithm, the successor function is our mutate function, which takes in an individual, and mutates it to produce a slightly different individual.

Our successor functions can be defined by two components, a distribution type, and a number of parameters to modify, n. The parameters are modified by a random amount δ, which may be sampled from one of three distributions: a uniformly random distribution, a normal distribution,

or a constant distribution of size one. We are currently using the June crude oil commodity to experiment with all three distributions for $n = 1$, 3 and 6.

We will conduct twenty-seven experiments. An experiment consists of a learning algorithm, a distribution type, and a number of parameters to modify. For each experiment, we will train the learner, using the earliest two thirds of the June crude oil data. In order to study the learning speed for each experiment, we will train the learner, using an increasing number of iterations. We will range from 10 iterations to 200, in steps of 10, recording the best parameters at each step. We will evaluate the set of best parameters, using the testing data, and record the profit, for each step. To minimize noise, we will take an average of 100 trials at each step. Below we present plots of the profit earned as a function of the number of iterations, for each learning algorithm.

WE WILL INCLUDE THREE PLOTS OF PROFIT VS. NUMBER OF ITERATIONS, ONE FOR EACH LEARNING ALGORITHM. EACH PLOT WILL HAVE 9 LINES, REPRESENTING 9 DIFFERENT SUCCESSOR VARIATIONS FOR THAT LEARNING ALGORITHM.

4.2 Genetic Algorithm variations

We intend to experiment with the mutation probability, population size, and reproduction function, for our genetic algorithm. Below we present plots of the profit earned as a function of each of the above variations.

PLOTS OF: PROFIT EARNED VS. MUTATION PROBABILITY, PROFIT EARNED VS. POPULATION SIZE AND PROFIT EARNED VS. REPRODUCTION FUNCTION

4.3 Finding Optimal Parameters

After determining the successor functions and genetic algorithm variations, we will use each learning algorithm to find optimal parameters for each of the fourteen commodities. For each commodity, we will first train the learner over 300 iterations, recording the most profitable set of six parameters. We will then evaluate this set of parameters, using the testing data.

Below we will present the parameters obtained from each of our three learning algorithms, along with Momsen's published parameter values, for all fourteen commodities. For each set of parameters, we will report the profit earned after evaluating the parameters with the testing data.

NOTE: THESE TABLES WILL BE FILLED IN FOR THE FINAL DRAFT.

April	Channel Thresholds			Window Dates			
Live Cattle	Entry	Trail Stop	Stop Loss	Open	Close	Exit	Profit
Simulated Annealing	a	b	c	d	e	f	$
Genetic Algorithms	a	b	c	d	e	f	$
Random Walk	a	b	c	d	e	f	$
Momsen	a	b	c	d	e	f	$

August	Channel Thresholds			Window Dates			
Pork Bellies	Entry	Trail Stop	Stop Loss	Open	Close	Exit	Profit
Simulated Annealing	a	b	c	d	e	f	$
Genetic Algorithms	a	b	c	d	e	f	$
Random Walk	a	b	c	d	e	f	$
Momsen	a	b	c	d	e	f	$

December	Channel Thresholds			Window Dates			
Corn	Entry	Trail Stop	Stop Loss	Open	Close	Exit	Profit
Simulated Annealing	a	b	c	d	e	f	$
Genetic Algorithms	a	b	c	d	e	f	$
Random Walk	a	b	c	d	e	f	$
Momsen	a	b	c	d	e	f	$

December	Channel Thresholds			Window Dates			
Wheat	Entry	Trail Stop	Stop Loss	Open	Close	Exit	Profit
Simulated Annealing	a	b	c	d	e	f	$
Genetic Algorithms	a	b	c	d	e	f	$
Random Walk	a	b	c	d	e	f	$
Momsen	a	b	c	d	e	f	$

December	Channel Thresholds			Window Dates			
Lean Hogs	Entry	Trail Stop	Stop Loss	Open	Close	Exit	Profit
Simulated Annealing	a	b	c	d	e	f	$
Genetic Algorithms	a	b	c	d	e	f	$
Random Walk	a	b	c	d	e	f	$
Momsen	a	b	c	d	e	f	$

February	Channel Thresholds			Window Dates			
Crude Oil	Entry	Trail Stop	Stop Loss	Open	Close	Exit	Profit
Simulated Annealing	a	b	c	d	e	f	$
Genetic Algorithms	a	b	c	d	e	f	$
Random Walk	a	b	c	d	e	f	$
Momsen	a	b	c	d	e	f	$

July	Channel Thresholds			Window Dates			
Soy Beans	Entry	Trail Stop	Stop Loss	Open	Close	Exit	Profit
Simulated Annealing	a	b	c	d	e	f	$
Genetic Algorithms	a	b	c	d	e	f	$
Random Walk	a	b	c	d	e	f	$
Momsen	a	b	c	d	e	f	$

June	Channel Thresholds			Window Dates			
Crude Oil	Entry	Trail Stop	Stop Loss	Open	Close	Exit	Profit
Simulated Annealing	a	b	c	d	e	f	$
Genetic Algorithms	a	b	c	d	e	f	$
Random Walk	a	b	c	d	e	f	$
Momsen	a	b	c	d	e	f	$

June	Channel Thresholds			Window Dates			
Unleaded Gas	Entry	Trail Stop	Stop Loss	Open	Close	Exit	Profit
Simulated Annealing	a	b	c	d	e	f	$
Genetic Algorithms	a	b	c	d	e	f	$
Random Walk	a	b	c	d	e	f	$
Momsen	a	b	c	d	e	f	$

June	Channel Thresholds			Window Dates			
Lean Hogs	Entry	Trail Stop	Stop Loss	Open	Close	Exit	Profit
Simulated Annealing	a	b	c	d	e	f	$
Genetic Algorithms	a	b	c	d	e	f	$
Random Walk	a	b	c	d	e	f	$
Momsen	a	b	c	d	e	f	$

May	Channel Thresholds			Window Dates			
Heating Oil	Entry	Trail Stop	Stop Loss	Open	Close	Exit	Profit
Simulated Annealing	a	b	c	d	e	f	$
Genetic Algorithms	a	b	c	d	e	f	$
Random Walk	a	b	c	d	e	f	$
Momsen	a	b	c	d	e	f	$

November	Channel Thresholds			Window Dates			
Crude Oil	Entry	Trail Stop	Stop Loss	Open	Close	Exit	Profit
Simulated Annealing	a	b	c	d	e	f	$
Genetic Algorithms	a	b	c	d	e	f	$
Random Walk	a	b	c	d	e	f	$
Momsen	a	b	c	d	e	f	$

October	Channel Thresholds			Window Dates			
Lean Hogs	Entry	Trail Stop	Stop Loss	Open	Close	Exit	Profit
Simulated Annealing	a	b	c	d	e	f	$
Genetic Algorithms	a	b	c	d	e	f	$
Random Walk	a	b	c	d	e	f	$
Momsen	a	b	c	d	e	f	$

September	Channel Thresholds			Window Dates			
Orange Juice	Entry	Trail Stop	Stop Loss	Open	Close	Exit	Profit
Simulated Annealing	a	b	c	d	e	f	$
Genetic Algorithms	a	b	c	d	e	f	$
Random Walk	a	b	c	d	e	f	$
Momsen	a	b	c	d	e	f	$

5 Conclusions

We intend to compare the learning rates for each of our three learning algorithms. We hope to find that simulated annealing and our genetic algorithm learn to increase the profit, more rapidly than our random walk algorithm. We will also compare the final parameters obtained for each of the learning algorithms to Momsen's published parameters. We hope to find more profitable parameter's than Momsen's.

References

[1] Tom M. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[2] John Momsen. *Superstar Seasonals: 18 Proven-Dependable Futures Trades For Profiting Year After Year*. Windsor Books, first edition, 2004.

[3] Stuart Russell and Peter Norvig. *Artificial Intelgence: A Modern Approach*. Prentice Hall, Upper Saddle River, NJ, 2003.

[4] Wikipedia. Commodity market — Wikipedia, the free encyclopedia, 2009. [Online; accessed 23-May-2009].