Learning to Predict Prices in a Supply Chain Management Game

Shuo Chen, Amy Greenwald, and Aysun Bascetincelik

August 30, 2007

Abstract

Economic decisions can benefit greatly from accurate predictions of market prices, but making such predictions is a difficult problem and an area of active research. In this paper, we present and compare several techniques for predicting market prices that we have employed in the Trading Agent Competition Supply Chain Management (TAC SCM) Prediction Challenge. These strategies include simple heuristics and various machine learning approaches, such as simple perceptrons and support vector regression. We show that the heuristic methods are very good, especially for predicting current prices, but that the machine learning techniques may be more appropriate for future price predictions.

1 Introduction

A manufacturer that procures raw materials, converts them into final products, and sells the products to customers is said to manage a supply chain. Successful supply chain management in a competitive market requires accurately predicting the prices of raw materials and the prices at which final products will be sold. The Trading Agent Competition Supply Chain Management (TAC SCM) game simulates such a market, and the TAC SCM Prediction Challenge, a new competition in 2007, was designed to provide a controlled environment for researching how such predictions can be made.

In the TAC SCM game, software agents act as computer manufacturers that compete to win orders from customers, purchase components from suppliers, and produce and deliver finished products to customers. The customers send *requests for quotes* (RFQs) for computers to all the participating agents, and the agents bid prices on those RFQs in a sealed-bid reverse auction. In order to maximize its profit relative to other agents, an agent should be able to determine the price at which each RFQ will be ordered. Furthermore, the agent sends RFQs to suppliers to purchase components, and predictions of the prices at which those RFQs will be offered are essential to deciding not only which components to procure from which suppliers and when, but also which customer RFQs to bid on and for how much.

The importance of these predictions has led to the creation of the TAC SCM Prediction Challenge, which offers a sandbox for experimenting with different prediction techniques as well as a framework for evaluating those techniques, free of the intricate complexities of the rest of the SCM game. Here we present the methods we have implemented and compare their effectiveness. The paper is organized as follows. Section 2 describes TAC SCM and the Prediction Challenge in greater detail, and reviews related prior work. Section 3 and Section 4 discuss our work on computer and component price predictions, respectively. Each section presents first the sources of information available to us, then the different prediction techniques we have attempted, and finally the results we have obtained. Section 5 shows the results of the final round in the Prediction Challenge. Finally, we interpret our results and conclude the paper in Section 6.

2 Background

2.1 The TAC SCM Game

In the TAC SCM game,[1] six computer manufacturers compete to produce and sell 16 different types of computers or *stock keeping units* (SKUs), each requiring a unique combination of 10 different kinds of CPU, motherboard, memory, and hard disk components. Each kind of component may be supplied by up to two different suppliers, who may be thought of as simpler versions of the agents that must also make offers in response to agent RFQs, produce components, and deliver ordered components to agents. The suppliers attempt to maximize their revenue and offer prices based on the ratio of demand to supply.

Each game lasts for 220 simulated days. Every day, the agent receives a set of RFQs from a single conglomeration of customers. Each RFQ consists of a computer type, the quantity desired, the due date, a reserve price indicating the highest price that the customers are willing to pay, and a penalty that will be exacted each day the delivery is late. Each of these properties is chosen uniformly at random from an interval.

The RFQ that an agent sends to a supplier is similar to a customer RFQ. It consists of a component type, the quantity desired, the due date, and a reserve price, which may be zero if the agent does not wish to constrain the price. On the next day, the supplier to whom the RFQ was sent sends back at least one offer. Each offer may match the RFQ exactly, in which case it would be the only one, or may be a *partial offer* with a reduced quantity or an *earliest complete offer* with a later due date. The SCM agent then decides which offers to take and sends orders to the suppliers on the same day.

The agents themselves issue offers to customers in response to RFQs received on the same day. They must offer prices without knowing the prices that other agents have offered. On the following day, the customers awards for each RFQ the agent that offered the lowest price with an order, and the winning agent can then deliver that order and

[1] The full specification for this game is available at `http://www.sics.se/tac/page.php?id=14`

receive payment.

2.2 The Prediction Challenge

In the TAC SCM Prediction Challenge,[2] the agent's task is to predict computer and component prices on the behalf of an SCM agent. The SCM agent, called PAgent, is an agent designed by the creators of the challenge with simple but reasonable behavior and that wins a moderate number of customer orders. The predictions are not used by PAgent in any way. In fact, during the competition, SCM game logs with PAgent participating are read, and the prediction agent receives only PAgent's incoming and outgoing messages, day by day. Predictions must be made for the current day before the next day's messages are received.

Four kinds of predictions must be made each day. They are:

1. The price at which each customer RFQ received today will be ordered tomorrow,
2. The median order price for each SKU 20 days from today,
3. The price that will be offered tomorrow for each supplier RFQ that PAgent sent today, and
4. The price that will be offered for each supplier RFQ that PAgent will send 20 days from today.

Once the predictions are made, they are compared to the actual prices and the root mean square (RMS) error in each prediction category is calculated for the current day. In the error calculation, each price is divided by the base price for that component or SKU. Customer RFQs that do not result in an order and supplier RFQs that result in partial, earliest-complete, or no offers are excluded. RMS errors are also calculated for the entire game and for all the games. The prediction agent makes predictions for a total of 48 games, divided into three sets, in each of which PAgent plays 16 games against the same competitors.

2.3 Prior Work

Although the Prediction Challenge was new in 2007, researchers have been interested in price predictions for the SCM game itself, and a variety of approaches have been studied. However, most work has focused on determining the best price at which to bid for computers, or the probability of winning a bid at a given price. Simple online and historical learning techniques, as well as fixed strategies, were examined and compared in [2], which shows that dynamic pricing strategies perform better than fixed bids. A naive Bayes approach that classifies each bid as winning or losing and then derives the

[2]More information about the challenge can be found at `http://www.cs.utexas.edu/~TacTex/PredictionChallenge/`

price-probability function was developed in [5] and is similar to the machine learning techniques presented here. Kiekintveld et. al. [3] used a nearest neighbors approach that combined online and historical learning in the 2005 version of Deep Maize, an agent in TAC SCM. Pardoe and Stone [6] explored several machine learning techniques for predicting the probability of winning a bid, including support vector machines, naive Bayes, nearest neighbors, regression trees, and boosted decision stumps. They rejected the first three based on initial testing and showed that the latter two performed best. However, in a later version of TacTex, a successful agent in TAC SCM 2006, they adapted instead a particle filter as the price-probability model for the customer side [7]. Overall, the prediction of component prices, which is essential to both procurement and bidding decisions in TAC SCM, has not been explored in depth, and this work contributes to research in this area.

3 Computer Price Predictions

3.1 Inputs

In predicting current and future computer prices, several sources of information are available. Every day, the prediction agent receives the current day's customer RFQs, each with a set of properties, and a price report, which consists of the previous day's highest and lowest order price for each SKU. The agent also knows all of PAgent's past offers to customers, and which ones were awarded with orders. Every 20 days, the prediction agent receives a market report that includes the quantity of each SKU requested and ordered and the average order price for each SKU during the 20-day period.

3.2 Methods

3.2.1 A Simple Heuristic

The first approach we attempted was a simple heuristic adopted in Botticelli [1], Brown University's TAC SCM agent, as the computer prices model. This model predicts the probability of winning an offer given the bid price for that offer by using the price report and the agent's past offers. It plots the following $d + 2$ points for each SKU every day and calculates a least-squares regression line for them:

- The highest and lowest order prices for that SKU yesterday, plotted respectively at probability 0 and probability 1, and
- Botticelli's daily average offer price for that SKU for each of the past d days, each plotted at the ratio of offers won to offers issued on that day.

The justification for the first two points is that bids with lower prices are more likely to be won than bids with higher prices. The regression line represents the complementary

cumulative distribution function for the winning price, since the probability of winning an offer at a given price is the probability that the true winning price is greater than or equal to that price. However, we only need a single number for the Prediction Challenge, the actual winning price, which may be predicted as the expected value of the winning price. We obtained this number by taking the average of the price at probability 1 and the price at probability 0 on the regression line.

3.2.2 A Perceptron Approach

Next, we attempted a simple perceptron approach. The idea is to learn to predict the winning price from PAgent's past offers to customers and which ones became orders. Each offer corresponds to a customer RFQ, which has a set of properties that we used as the input values or features for the perceptron. We also know the price of each offer and whether it was won or lost. This information gives upper and lower bounds on the true winning price, since it should be lower than the price of any offer that was lost and greater than or equal to the price of any offer that was won. The goal is to learn the weights that would result in an output that is on the right side of the offer price for each inequality in the training data. That is, we aim to learn weights $(w_1, w_2, \ldots, w_m)$ and bias b such that:

$$\begin{pmatrix} f_{11} & f_{12} & \ldots & f_{1m} & 1 \\ f_{21} & f_{22} & \ldots & f_{2m} & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ f_{n1} & f_{n2} & \ldots & f_{nm} & 1 \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \\ b \end{pmatrix} \begin{matrix} < \\ \geq \\ \vdots \\ < \end{matrix} \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_n \end{pmatrix} \begin{matrix} \text{offer lost} \\ \text{offer won} \\ \vdots \\ \text{offer lost} \end{matrix} \tag{1}$$

where n is the number of training instances (past customer offers), m is the number of features (five properties of the RFQ), f_{ij} is the jth feature in the ith training instance, and p_i is the offer price in the ith training instance. The final column indicates whether each offer was won or lost.

After some experimentation, we settled on the following update rule, where y_i is the perceptron's output for the ith training instance and α is the learning rate:

$$\Delta w_j, \Delta b = \begin{cases} +\alpha f_{ij} & \text{if } y_i < p_i \text{ and the } i\text{th offer was won} \\ -\alpha f_{ij} & \text{if } y_i \geq p_i \text{ and the } i\text{th offer was lost} \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

The learning rate α decays as the right half of a Gaussian with $\mu = 0$ and $\sigma = \frac{1}{\sqrt{2\pi}}$, so that it is 1 at time 0. The decaying learning rate is necessary to force the weights to converge.

Since the order in which the perceptron goes through the training data affects the final weights, we start every day with 10 copies of the weights learned so far, and update each copy using the current day's customer offers presented in a different random order.

After all 10 copies have been updated, we take their average to be the final weights for that day. The order price of a customer RFQ received on the current day is then predicted as $\vec{f} \cdot \vec{w} + b$. The predictions made by the perceptron turned out to be inexplicably bad for any reasonable number of iterations through the training data, so this approach is not included in the Results section.

3.2.3 Support Vector Regression

Attempting the perceptron approach was not completely fruitless, however, as it yielded another idea. The fact that the magnitudes of the perceptron's weights seemed too large motivated us to solve for $\vec{w}$ and b in (1) using mathematical programming techniques and minimizing the magnitude of the weights vector. Because the quadratic program is not feasible on every day, we introduce slack variables ς_i into the problem, so that the quadratic program on each day becomes:

$$\begin{aligned} \text{minimize} \quad & \|\vec{w}\|^2 + \sum_{i=1}^{n} \varsigma_i \\ \text{subject to} \quad & \begin{cases} \vec{f_i} \cdot \vec{w} + b + \varsigma_i \geq p_i & \text{for each offer won} \\ \vec{f_j} \cdot \vec{w} + b - \varsigma_j < p_j & \text{for each offer lost} \\ \varsigma_i \geq 0 \end{cases} \end{aligned} \tag{3}$$

This approach is a variation of support vector regression, which is described in greater detail in [8]. We use only the current day's customer orders, because using past days' data not only made the predictions worse but also increased the size of the problem very quickly, since there are about 200 customer RFQs every day.

We also added more features to the quadratic program, such as the past, current, and predicted future customer demand, for which we used Deep Maize's Bayesian model.[3] The SKUs are divided into three market segments (low, mid, and high), and the number of RFQs in each market segment is determined by a Poisson distribution whose parameter varies according to a random walk. The Bayesian model predicts the number of RFQs in each market segment on any day in the game given past observations [4]. Finally, we added as features the sum of the costs of the components required for the SKU as predicted by the supplier-side model described in Section 4.2.1, and the average of the highest and lowest order prices for the SKU for a number of past days.

3.3 Results

Table 1 summarizes our results. The overall RMS errors for all 48 games in the Prediction Challenge qualifying round (on which we performed most of our tests) is shown for each strategy. The errors under "Current Prices" are for predictions of the prices on the

[3]Deep Maize has released their source code for this approach at `http://www.sics.se/tac/showagents.php?id=19`

Method	Current Prices	Future Prices
Sample	0.04882	0.10654
Heuristic	0.04665	0.10170
SVR	0.05463	0.13132

Table 1: RMS errors for various computer prices prediction strategies.

next day, and those under "Future Prices" are for predictions of the prices 20 days from the current day. The "sample" strategy is the default given to us by the designers of the challenge, which simply predicts the price as the average of yesterday's highest and lowest order prices for the RFQ's SKU. For support vector regression (SVR), we used only the properties of the RFQ and the last 5 days' price reports as features, as the other features seemed to make predictions worse.

4 Component Price Predictions

4.1 Inputs

The information available for predicting current and future component prices include information that is known daily and information from the market report received every 20 days. Every day, the prediction agent receives supplier offers in response to PAgent's supplier RFQs on the previous day, and also knows all the past supplier offers received. Every 20 days, the prediction agent finds out the quantity of each component type ordered and delivered, the mean offer price for each component per SKU, and the mean production capacity for each supplier product line during the 20-day period. A *supplier product line* is a supplier-component pair, and each supplier product line maintains a separate production capacity.

4.2 Methods

4.2.1 Botticelli's Model

Again, we started with Botticelli's supplier-side model. In this model, each day's supplier offers are used to predict the prices of components. Each known offer price is stored and indexed by the component, the supplier, and the due date for that offer. Only the prices from the current day's offers are stored, so that we only use the most recent information. Then, to predict the offer price for a supplier RFQ sent on the current day, we perform the following steps:

1. Estimate the price as the known price for that RFQ's component, supplier, and the nearest due date for which data is available,

2. Estimate the supplier's available capacity for the RFQ by using the supplier pricing formula and the estimated price given by Step 1,

3. Adjust the estimated capacity by adding to it the difference between the quantity offered and the quantity ordered on the current day of that component from the current day to the RFQ's due date, and finally

4. Re-estimate the offer price by using the supplier pricing formula and the adjusted available capacity estimate obtained in Step 3.

The supplier pricing formula, known as Equation (10) in the TAC SCM specifications, determines the offer price on a given day d for a component c due on day $d + i + 1$, as follows:

$$P_{d,i} = P_c^{base} \left(1 - \delta \left(\frac{C_{d,i}^{avl}}{iC_d^{ac}}\right)\right) \tag{4}$$

where

- $P_{d,i}$ is the offer price,
- P_c^{base} is the base price for component c, given by the TAC SCM specifications,
- δ is the supplier price discount factor, chosen to be 0.5,
- C_d^{ac} is the actual capacity for the supplier product line on day d, determined each day by a mean-reverting random walk, and
- $C_{d,i}^{avl}$ is the capacity available to produce the component, which is the inventory plus the free capacity (the actual capacity less the capacity that the supplier has offered or committed on that day) on every day between day d and day $d + i$, minus any capacity needed between day d and day $d + i$ to satisfy requests due after $d + i + 1$.

When calculating $C_{d,i}^{avl}$, suppliers assume that all offers sent on day d will become committed. Therefore, Step 3 (adjusting the estimated $C_{d,i}^{avl}$) makes sense because in determining prices on the current day, the suppliers believed that all offers would result in orders, whereas we know which offers we decided to take and thus can better estimate the available capacity the suppliers will use to calculate prices for the following day. We always take C_d^{ac} to be the supplier's nominal capacity, which is the mean that the random walk determining the actual capacity reverts to.

4.2.2 A Perceptron Approach

The perceptron approach for component prices is almost identical to that for computer prices, except that there is a separate perceptron for each type of component and we

have equalities instead of inequalities:

$$\begin{pmatrix} f_{11} & f_{12} & \dots & f_{1m} & 1 \\ f_{21} & f_{22} & \dots & f_{2m} & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ f_{n1} & f_{n2} & \dots & f_{nm} & 1 \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \\ b \end{pmatrix} = \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_n \end{pmatrix} \tag{5}$$

and the perceptron update rule is:

$$\Delta w_j, \Delta b = \begin{cases} +\alpha f_{ij} & \text{if } y_i < p_i \\ -\alpha f_{ij} & \text{if } y_i > p_i \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

4.2.3 Least Squares Regression

The simplest way to solve for $\vec{w}$ and b in (5) is just to find a least-squares solution. However, while this method worked for some games, it was very bad for others, and so is not included in the Results section.

4.2.4 Support Vector Regression

The support vector regression approach is almost identical to the one used for computer prices, except that once again, we calculate the weights for each component separately and have equalities instead of inequalities. The optimization problem is:

$$\begin{aligned} \text{minimize} \quad & \|\vec{w}\|^2 + \sum_{i=1}^{n} \varsigma_i^2 \\ \text{subject to} \quad & \vec{f_i} \cdot \vec{w} + b + \varsigma_i = p_i \end{aligned} \tag{7}$$

We use the features of the RFQ, as well as features indicating market conditions, such as past, current, and future customer demand, past prices for that component, the mean past actual capacity for that supplier product line from the most recently received market report, and the estimated available capacity as described in Section 4.2.6. We also include as a feature the future supplier actual capacity predicted using a nearest neighbors approach.

4.2.5 k-Nearest Neighbors

Another approach we attempted was k-nearest neighbors. In this method, we store all supplier offers and predict the price for an RFQ as a weighted average of the prices of the k most similar supplier offers. Similarity is measured by the Euclidean distance between the features for the RFQ (including market conditions) and those for the offer. The weights for the k neighbors are inversely proportional to their distances and normalized.

Method	Current Prices	Future Prices
Sample	0.15428	0.16324
Heuristic	0.04163	0.10067
Perceptron	0.18442	0.14871
SVR	0.05855	0.09674
k-NN	0.04653	0.10338
Capacity	0.17664	0.18271

Table 2: RMS errors for various component prices prediction strategies.

4.2.6 Modeling the Available Capacity

Yet another approach, motivated by the supplier pricing formula in (4), is to model the available capacity for each supplier product line on each day. To do this, we start with the supplier's nominal capacity divided by the number of SCM agents in the game, which is approximately the amount of the supplier product line's production capacity that is available to PAgent at the beginning of the game. Then, for each supplier order sent on the current day that is for this supplier-component pair, we "schedule" the quantity ordered greedily from the next day (when the supplier can start producing for the order) forward by decrementing each day's estimated available capacity for that supplier product line by the maximum amount possible of the quantity left to satisfy. The estimated available capacity is then used in (4) to predict the price for an RFQ, with the actual capacity estimated as the supplier's nominal capacity divided by the number of SCM agents.

4.3 Results

Table 2 summarizes our results for component price predictions. Again, the RMS errors are for the games in the qualifying round. The "sample" strategy simply predicts the price for any RFQ to be 0.7 times the base price of the component. For the perceptron approach, we use only the properties of the RFQ as the features. For support vector regression (SVR) and k-nearest neighbors (k-NN), we let $k = 5$, and use the properties of the RFQ as well as the past mean supplier production capacity from the latest market report and the average component prices for the past 5 days. Also, we use only the current and previous days' supplier offers for SVR, and we use only the supplier offers from the current game for k-NN. These parameters all seemed to make performance the best.

Current Computer Prices			Future Computer Prices		
Rank	Team	RMS Error	Rank	Team	RMS Error
1	TacTex	0.04554	1	TacTex	0.09156
2	DeepMaize	0.04682	2	DeepMaize	0.09586
3	Botticelli	0.04714	3	Botticelli	0.10238
4	Kshitij	0.04873	4	Kshitij	0.11094
Current Component Prices			Future Component Prices		
Rank	Team	RMS Error	Rank	Team	RMS Error
1	DeepMaize	0.03919	1	DeepMaize	0.09427
2	Botticelli	0.04168	2	Botticelli	0.09700
3	TacTex	0.04284	3	TacTex	0.10338
4	Kshitij	0.13333	4	Kshitij	0.13886

Table 3: RMS errors for all teams and categories in the final round.

5 Final Round Results

In the final round of the Prediction Challenge, we decided to use the simple heuristics for both current and future computer price predictions, and only current component price predictions. For future component prices, we used support vector regression. The results of the final round for all participating teams are summarized in Table 3. Our team was Botticelli.

6 Conclusion

For all the price prediction tasks, the simple methods adapted from Botticelli work very well. The more sophisticated machine learning techniques are surprisingly worse especially for predictions of current prices, although not much worse. This suggests that in the TAC SCM scenario, just using the recent prices, which both of the simple methods do, is better than trying to generalize the relationship between market conditions and prices. In fact, for both computer and component prices, attempting to use older data from the same game and data from past games made the predictions worse, so it seems that there is not much of a consistent relationship between market conditions and prices across games, and even within a game. Or perhaps even more sophisticated learning techniques are necessary to detect such a relationship, and implementing them may be a direction for future work. Nevertheless, for future component prices, support vector regression intriguingly yields slightly better results than the simple method, which suggests that current prices are less relevant to the future, and that predicting future market prices may be more amenable to machine learning approaches.

Acknowledgments

We would like to thank David Pardoe for organizing the TAC SCM Prediction Challenge and providing the means to experiment with and evaluate different techniques for predicting market prices. We would also like to thank Gregory Shakhnarovich for his insights on the prediction problem. This research was supported in part by the Committee on the Status of Women in Computing Research's Distributed Mentor Project.

References

[1] M. Benisch, A. Greenwald, I. Grypari, R. Lederman, V. Naroditskiy, and M. Tschantz. Botticelli: a supply chain management agent. In *Proceedings of the 3rd International Conference on Autonomous Agents and Multiagent Systems*, volume 3, pages 1174–1181, July 2004.

[2] D. A. Burke, K. N. Brown, B. Hnich, and A Tarim. Learning market prices for a real-time supply chain management trading agent. In *AAMA-06 Workshop on Trading Agent Design and Analysis*, 2006.

[3] C. Kiekintveld, J. Miller, P. R. Jordan, and M. P. Wellman. Forecasting market prices in a supply chain game. In *Sixth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1318–1325, May 2007.

[4] C. Kiekintveld, M. P. Wellman, S. Singh, J. Estelle, Y. Vorobeychik, V. Soni, and M. Rudary. Distributed feedback control for decision making on supply chains. In *Fourteenth International Conference on Automated Planning and Scheduling*, 2004.

[5] R. D. Lawrence. A machine-learning approach to optimal bid pricing. In *Proceedings of the Eighth INFORMS Computing Society Conference on Optimization and Computation in the Network Era*, 2003.

[6] D. Pardoe and P. Stone. Bidding for customer orders in TAC SCM. In *AAMAS 2004 Workshop on Agent Mediated Electronic Commerce VI: Theories for and Engineering of Distributed Mechanisms and Systems*, July 2004.

[7] D. Pardoe and P. Stone. An autonomous agent for supply chain management. In G. Adomavicius and A. Gupta, editors, *Handbooks in Information Systems Series: Business Computing*. Elsevier, 2007. To appear.

[8] A. J. Smola and B. Schölkopf. A tutorial on support vector regression. Technical Report NC2-TR-1998-030, ESPRIT Working Group in Neural and Computational Learning II, October 1998.