Conference submission (2/1/02).

Modeling Auction Price Uncertainty Using Boosting-based Conditional Density Estimation

Robert E. Schapire SCHAPIRE@RESEARCH.ATT.COM
Peter Stone PSTONE@RESEARCH.ATT.COM
David McAllester DMAC@RESEARCH.ATT.COM
Michael L. Littman MLITTMAN@RESEARCH.ATT.COM
János A. Csirik JANOS@RESEARCH.ATT.COM
AT&T Labs – Research, Shannon Laboratory, 180 Park Avenue, Florham Park, NJ 07932

Abstract

In complicated, interacting auctions, a fundamental problem is the prediction of prices of goods in the auctions, and more broadly, the modeling of uncertainty regarding these prices. In this paper, we present a machine-learning approach to this problem. The technique is based on a new and general boosting-based algorithm for conditional density estimation problems of this kind. This algorithm, which we present in detail, is at the heart of ATTac-2001, a top-scoring agent in the recent Trading Agent Competition (TAC-01). We describe how ATTac-2001 works, the results of the competition, and controlled experiments evaluating the effectiveness of price prediction in auctions.

1. Introduction

Auctions are becoming an increasingly popular method for transacting business, especially over the internet. In an auction for a single good, it is fairly straightforward to create automated bidding strategies; for instance, an agent can keep bidding until reaching a target reserve price, or it can monitor the auction and place a winning bid just before the closing time.

When bidding for multiple interacting goods in simultaneous auctions, on the other hand, agents must be able to reason about uncertainty and make complex value assessments. For example, an agent bidding for a camera and flash may end up buying the flash and then not being able to find an affordable camera. Alternatively, if bidding for the same good in several auctions, it may purchase two flashes when only one was needed.

When bidding in any auction, it is important to be able to evaluate how much each item is worth to the agent. In interacting auctions, this also requires being able to predict the price of other items in the auction. For instance, in the example above, to determine the value of the camera, we need to guess the price of the flash; the amount that we are willing to spend on the camera then is the difference between the value of the camera-flash combination and the estimated price of the flash alone. Thus, a fundamental challenge when bidding for multiple goods is predicting the prices of all of the relevant goods before they are known.

To attack the price prediction problem, we propose a machine-learning approach: gather examples of previous auctions and the prices paid in them, then use machine-learning methods to predict these prices. This learning problem can be viewed as a conditional-density-estimation problem: given current conditions, estimate the conditional distribution of prices. To solve the problem, we devise a new boosting-based algorithm that we present in detail in this paper.

We also describe how we successfully applied the algorithm to the problem of price prediction in auctions. It was implemented as a part of ATTac-2001, a top-scoring agent[1] in the second Trading Agent Competition (TAC-01), which was held in Tampa Bay, FL on October 14, 2001. In this competition, each agent must bid simultaneously for multiple interacting goods. As observed above, the key challenge in such auctions is the modeling of uncertainty in the eventual prices of goods: with complete knowledge of eventual prices, there are direct methods for determining the optimal bids to place. Our guiding principle in the design of ATTac-2001 is to have the agent model uncertainty and, to the greatest extent possible, analytically calculate optimal bids. ATTac-2001 uses a predictive, data-driven approach to bidding based on expected marginal values of all available goods. The price-predictor presented in this paper is at the heart of the algorithm.

In addition to the results of the trading agent competition itself, we also present controlled experiments exploring the degree to which accuracy in price prediction affects overall performance in the auction.

[1]Top-scoring by one metric, and second place by another.

2. TAC

We first describe the TAC domain in order to motivate the need for our new algorithm.

We instantiated our approach as an entry in the second Trading Agent Competition (TAC), as described in this section. Building on the success of TAC-00 held in July 2000 (Wellman et al., 2001; Stone & Greenwald, 2002), TAC-01 included 19 agents from 9 countries. A key feature of TAC is that it required *autonomous bidding agents* to buy and sell *multiple interacting goods* in auctions of different types. It is designed as a benchmark problem in the complex and rapidly advancing domain of e-marketplaces, motivating researchers to apply unique approaches to a common task.

A TAC game instance pits eight autonomous bidding agents against one another. Each TAC agent is a simulated travel agent with eight clients, each of whom would like to travel from TACtown to Tampa and back again during a 5-day period. Each client is characterized by a random set of preferences for the possible arrival and departure dates; hotel rooms; and entertainment tickets. To satisfy a client, an agent must construct a travel package for that client by purchasing airline tickets to and from TACtown and securing hotel reservations; it is possible to obtain additional bonuses by providing entertainment tickets as well. A TAC agent's score in a game instance is the difference between the sum of its clients' utilities for the packages they receive and the agent's total expenditure. We provide selected details about the game next; for full details on the design and mechanisms of the TAC server and TAC game, see `http://tac.eecs.umich.edu`.

TAC agents buy flights, hotel rooms and entertainment tickets through auctions, run from the TAC server at the University of Michigan. Each game instance lasts 12 minutes and includes a total of 28 auctions of 3 different types:

Flights (8 auctions): There is a separate auction for each type of airline ticket: to Tampa (*inflights*) on days 1–4 and from Tampa (*outflights*) on days 2–5. There is an unlimited supply of airline tickets, and their ask price changes randomly every 30 seconds or so, with an increasing bias upwards. When the server receives a bid at or above the ask price, the transaction is cleared immediately at the ask price and no resale is allowed.

Hotel Rooms (8): There are two different types of hotel rooms—the Tampa Towers (TT) and the Shoreline Shanties (SS)—each of which has 16 rooms available on days 1–4. The rooms are sold in a 16th-price *ascending* (English) auction, meaning that for each of the 8 types of hotel rooms, the 16 highest bidders get the rooms at the 16th highest price. The ask price is the current 16th-highest bid and transactions clear only when the auction closes. No bid withdrawal or resale is allowed, though the price of bids may be lowered provided the agent does not reduce the number of rooms it would win were the auction to close. One *randomly chosen* hotel auction closes at minutes 4–11 of the 12-minute game.

Entertainment Tickets (12): Alligator wrestling, amusement park, and museum tickets are each sold for days 1–4 in continuous double auctions. Here, agents can *buy and sell* tickets, with transactions clearing immediately when one agent places a buy bid at a price at least as high as another agent's sell price. Unlike the other auction types in which the goods are sold from a centralized stock, each agent starts with a (skewed) random endowment of entertainment tickets.

Each TAC agent has eight clients with randomly assigned travel preferences. Clients have parameters for ideal arrival day, *IAD* (1–4); ideal departure day, *IDD* (2–5); hotel premium, *HP* ($50–$150); and entertainment values, *EV* ($0–$200) for each type of entertainment ticket.

The utility obtained by a client is determined by the travel package that it is given in combination with its preferences. To obtain a non-zero utility, the client must be assigned a *feasible* travel package consisting of an inflight on some arrival day *AD*, an outflight on a departure day *DD*, and hotel rooms of the *same type* (TT or SS) for the days in between. At most one entertainment ticket of each type can be assigned, and no more than one on each day. Given a feasible package, the client's utility is defined as

$$1000 - \mathit{travelPenalty} + \mathit{hotelBonus} + \mathit{funBonus}$$

where

- *travelPenalty* = $100(|AD - IAD| + |DD - IDD|)$
- *hotelBonus* = *HP* if the client is in the TT, 0 otherwise.
- *funBonus* = sum of *EV*s for assigned entertainment tickets.

A TAC agent's *score* is the sum of its clients' utilities in the optimal allocation of its goods (computed by the TAC server) minus its expenditures.

TAC-01 was organized as a series of four competition phases, culminating with the semifinals and finals on October 14, 2001 at the EC-01 conference in Tampa, Florida. First, the qualifying round, consisting of about 270 games per agent, served to select the 16 agents that would participate in the semifinals. Second, the seeding round, consisting of about 315 games per agent, was used to divide these agents into two groups of eight. After the semifinals on the morning of the 14th, four teams from each group were selected to compete in the finals during that same afternoon.

3. Hotel Price Prediction

As discussed earlier, a central part of our strategy depends on the ability to predict prices, particularly hotel prices, at various points in the game. To do this as accurately as possible, we used machine-learning techniques that would examine the hotel prices actually paid in previous games to predict prices in future games. This section discusses this part of our strategy in detail, including a new boosting-based algorithm for conditional density estimation.

There is bound to be considerable uncertainty regarding hotel prices since these depend on many unknown factors, such as the time at which the hotel room will close, who the other agents are, what kind of clients have been assigned to each agent, etc. Thus, *exactly* predicting the price of a hotel room is hopeless. Instead, we regard the closing price as a random variable that we need to estimate conditional on our current state of knowledge (i.e., number of minutes remaining in the game, ask price of each hotel, flight prices, etc.). We might then attempt to predict this variable's conditional expected value. However, our strategy requires that we not only predict expected value, but that we also be able to estimate the *entire* conditional distribution of this random variable so that we can *sample* hotel prices.

To set this up as a learning problem, we gathered a set of training examples from previously played games. We defined a set of features for describing each example that together are meant to comprise a snap-shot of all the relevant information available at the time each prediction is made. All of the features we used are real-valued; a couple of the features can have a special value $\perp$ indicating "value unknown." We used the following basic features:

- The number of minutes remaining in the game.
- The price of each hotel room, i.e., the current ask price for rooms that have not closed or the actual selling price for rooms that have closed.
- The closing time of each hotel room. Note that this feature is defined even for rooms that have not yet closed, as explained below.
- The prices of each of the flights.

To this basic list, we added a number of redundant variations, which we thought might help the learning algorithm:

- The closing price of hotel rooms that have closed (or $\perp$ if the room has not yet closed).
- The current ask price of hotel rooms that have not closed (or $\perp$ if the room has already closed).
- The closing time of each hotel room minus the closing time of the room whose price we are trying to predict.
- The number of minutes from the current time until each hotel room closes.

During the seeding rounds, it was impossible to know during play who our opponents were, although this information was available at the end of each game, and therefore during training. During the semifinals and finals, we did know the identities of all our competitors. Therefore, in preparation for the semifinals and finals, we added the following features:

- The number of players playing (ordinarily eight, but sometimes fewer, for instance if one or more players crashed).
- A bit for each player indicating whether or not that player participated in this game.

We trained specialized predictors for predicting the price of each type of hotel room. In other words, one predictor was specialized for predicting only the price of TT on day 1, another for predicting SS on day 2, etc. This would seem to require eight separate predictors. However, the tournament game is naturally symmetric about its middle in the sense that we can create an equivalent game by exchanging the hotel rooms on days 1 and 2 with those on days 4 and 3 (respectively), and by exchanging the inbound flights on days 1, 2, 3 and 4 with the outbound flights on days 5, 4, 3 and 2 (respectively). Thus, with appropriate transformations, the outer days (1 and 4) can be treated equivalently, and likewise for the inner days (2 and 3), reducing the number of specialized predictors by half.

We also created specialized predictors for predicting in the first minute after flight prices had been quoted but prior to receiving any hotel price information. Thus, a total of eight specialized predictors were built (for each combination of TT versus SS, inner versus outer day, and first minute versus not first minute).

We trained our predictors to predict not the actual closing price of each room per se, but rather how much the price would increase, i.e., the difference between the closing price and the current price. We thought that this might be an easier quantity to predict, and, because our predictor never outputs a negative number when trained on nonnegative data, this approach also ensures that we never predict a closing price below the current bid.

From each of the previously played games, we were able to extract many examples. Specifically, for each minute of the game and for each room that had not yet closed, we extracted the values of all of the features described above at that moment in the game, plus the actual closing price of the room (which we are trying to predict).

Note that during training, there is no problem extracting the closing times of all of the rooms. During the actual play of a game, we do not know the closing times of rooms that have not yet closed. However, we do know the exact distribution of closing times of all of the rooms that have not yet closed. Therefore, to sample a vector of hotel prices, we can first sample according to this distribution over closing times, and then use our predictor to sample hotel prices using these sampled closing times.

3.1. The Learning Algorithm

Having described how we set up the learning problem, we are now ready to describe the learning algorithm that we used. Briefly, we solved this learning problem by first reducing to a multiclass, multi-label classification problem (or alternatively a multiple logistic regression problem), and then applying boosting techniques developed by Schapire and Singer (1999; 2000) combined with a modification of boosting algorithms for logistic regression proposed by Collins, Schapire and Singer (2002). The result is a new machine-learning algorithm for solving conditional density estimation problems, described in detail in the remainder of this section.

Abstractly, we are given pairs $(x_1, y_1), \ldots, (x_m, y_m)$ where each x_i belongs to a space X and each y_i is in $\mathbb{R}$. In our case, the x_i's are the auction-specific feature vectors described above; for some n, $X \subseteq (\mathbb{R} \cup \{\perp\})^n$. Each target quantity y_i is the difference between closing price and current price. Given a new x, our goal is to estimate the conditional distribution of y given x.

We proceed with the working assumption that all training and test examples (x, y) are i.i.d. (i.e, drawn independently from identical distributions). Although this assumption is false in our case (for instance, because the agents, including ours, are changing over time), it seems like a reasonable approximation that greatly reduces the difficulty of the learning task.

Our first step is to reduce the estimation problem to a classification problem by breaking the range of the y_i's into bins:

$$[b_0, b_1), [b_1, b_2), \ldots, [b_k, b_{k+1}]$$

for some breakpoints $b_0 < b_1 < \cdots < b_k \leq b_{k+1}$ where for our problem, we chose $k = 50$. The endpoints b_0 and b_{k+1} are chosen to be the smallest and largest y_i values observed during training. We choose the remaining breakpoints $b_1, \ldots, b_k$ so that roughly an equal number of training labels y_i fall into each bin. (More technically, breakpoints are chosen so that the entropy of the distribution of bin frequencies is maximized).

For each of the breakpoints b_j ($j = 1, \ldots, k$), our learning algorithm attempts to estimate the probability that a new y (given x) will be at least b_j. Given such estimates p_j for each b_j, we can then estimate the probability that y is in the bin $[b_j, b_{j+1})$ by $p_{j+1} - p_j$ (and we can then use a constant density within each bin). We thus have reduced the problem to one of estimating multiple conditional Bernoulli variables corresponding to the event $y \geq b_j$, and for this, we use a logistic regression algorithm based on boosting techniques as described by Collins, Schapire and Singer (2002).

Our learning algorithm constructs a real-valued function $f : X \times \{1, \ldots, k\} \to \mathbb{R}$ with the interpretation that

$$\frac{1}{1 + \exp(-f(x, j))} \tag{1}$$

is our estimate of the probability that $y \geq b_j$, given x. The negative log likelihood of the conditional Bernoulli variable corresponding to y_i being above or below b_j is then

$$\ln\left(1 + e^{-s_j(y_i) f(x_i, j)}\right)$$

where

$$s_j(y) = \begin{cases} +1 & \text{if } y \geq b_j \\ -1 & \text{if } y < b_j. \end{cases} \tag{2}$$

We attempt to minimize this quantity for all training examples (x_i, y_i) and all breakpoints b_j. Specifically, we try to find a function f minimizing

$$\sum_{i=1}^{m} \sum_{j=1}^{k} \ln\left(1 + e^{-s_j(y_i) f(x_i, j)}\right).$$

We use a boosting-like algorithm described by Collins, Schapire and Singer (2002) for minimizing objective functions of exactly this form. Specifically, we build the function f in rounds. On each round t, we add a new base function $h_t : X \times \{1, \ldots, k\} \to \mathbb{R}$. Let

$$f_t = \sum_{t'=1}^{t-1} h_{t'}$$

be the accumulating sum. Following Collins, Schapire and Singer, to construct each h_t, we first let

$$W_t(i, j) = \frac{1}{1 + e^{s_j(y_i) f_t(x_i, j)}}$$

be a set of weights on example-breakpoint pairs. We then choose h_t to minimize

$$\sum_{i=1}^{m} \sum_{j=1}^{k} W_t(i, j) e^{-s_j(y_i) h_t(x_i, j)} \tag{3}$$

over some space of "simple" base functions h_t. For this work, we considered all "decision stumps" h of the form

$$h(x, j) = \begin{cases} a_j & \text{if } \phi(x) \geq \theta \\ b_j & \text{if } \phi(x) < \theta \\ c_j & \text{if } \phi(x) = \perp \end{cases}$$

where $\phi(\cdot)$ is one of the features described above, and θ, a_j, b_j and c_j are all real numbers. In other words, such an h simply compares one feature ϕ to a threshold θ and returns a vector of numbers $h(x, \cdot)$ that depends only on whether $\phi(x)$ is unknown ($\perp$), or above or below θ. Schapire and Singer (2000) show how to efficiently search for the best such h over all possible choices of ϕ, θ, a_j, b_j and c_j. (We also employed their technique for "smoothing" a_j, b_j and c_j.)

When computed by this sort of iterative procedure, Collins, Schapire and Singer (2002) prove the asymptotic convergence of f_t to the minimum of the objective function in Eq. (3) over all linear combinations of the base functions. For this problem, we fixed the number of rounds to $T = 300$. Let $f = f_{T+1}$ be the final predictor.

As noted above, given a new feature vector x, we compute p_j as in Eq. (1) to be our estimate for the probability that $y \geq b_j$, and we let $p_0 = 1$ and $p_{k+1} = 0$. For this to make sense, we need $p_1 \geq p_2 \geq \cdots \geq p_k$, or equivalently, $f(x, 1) \geq f(x, 2) \geq \cdots \geq f(x, k)$, a condition that may not hold for the learned function f. To force this condition, we replace f by a reasonable (albeit heuristic) approximation f' that is nonincreasing in j, namely, $f' = (\overline{f} + \underline{f})/2$ where $\overline{f}$ (respectively, $\underline{f}$) is the pointwise minimum (respectively, maximum) of all nonincreasing functions g that everywhere upper bound f (respectively, lower bound f).

With this modified function f', we can compute modified probabilities p_j. To sample a single point according to the estimated distribution on $\mathbb{R}$ associated with f', we choose bin $[b_j, b_{j+1})$ with probability $p_j - p_{j+1}$, and then select a

```
**Input**: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in X$, $y_i \in \mathbb{R}$
    positive integers $k$ and $T$

**Compute breakpoints**: $b_0 < b_1 < \cdots < b_{k+1}$ where
  • $b_0 = \min_i y_i$
  • $b_{k+1} = \max_i y_i$
  • $b_1, \ldots, b_k$ chosen to minimize $\sum_{j=0}^{k} q_j \ln q_j$
    where $q_0, \ldots, q_k$ are fraction of $y_i$'s in
    $[b_0, b_1), [b_1, b_2), \ldots, [b_k, b_{k+1}]$ (using dynamic programing)

**Boosting**:
  • for $t = 1, \ldots T$:
      • compute weights $W_t(i,j) = \frac{1}{1 + e^{s_j(y_i) f_t(x_i,j)}}$
        where $s_j(y)$ is as in Eq. (2)
      • use $W_t$ to obtain base function $h_t : X \times \{1, \ldots, k\} \to$
        $\mathbb{R}$ minimizing $\sum_{i=1}^{m} \sum_{j=1}^{k} W_t(i,j) e^{-s_j(y_i) h_t(x_i,j)}$ over
        all decision stumps

**Output sampling rule**:
  • let $f = \sum_{t=1}^{T} h_t$
  • let $f' = (\overline{f} + \underline{f})/2$ where
        $\overline{f}(x,j) = \max\{f(x,j') : j \le j' \le k\}$
        $\underline{f}(x,j) = \min\{f(x,j') : 1 \le j' \le j\}$
  • to sample, given $x \in X$
      • let $p_j = \frac{1}{1 + e^{-f'(x,j)}}$
      • let $p_0 = 1, p_{k+1} = 0$
      • choose $j \in \{0, \ldots, k\}$ randomly with probability $p_j - p_{j+1}$
      • choose $y$ uniformly at random from $[b_j, b_{j+1}]$
      • output $y$
```

Table 1. The boosting-based algorithm for conditional density estimation.

point from this bin uniformly at random. Expected value according to this distribution is easily computed as

$$\sum_{j=0}^{k} (p_j - p_{j+1}) \left(\frac{b_{j+1} - b_j}{2} \right).$$

Table 1 shows pseudo-code for the entire algorithm.

4. ATTac-2001

Having described hotel price prediction in detail, we now overview some of the other key components of ATTac-2001 and how they use the price predictor.

Table 2 shows a high-level overview of ATTac-2001. The italicized portions use the learned price predictor in some way.

```
**When the first flight quotes are posted:**
  • Compute $G^*$ with current holdings and *expected prices*
  • Buy the flights in $G^*$ for which expected cost of postponing
    commitment exceeds the *expected benefit of postponing commitment*
**Starting 1 minute before each hotel close:**
      • Compute $G^*$ with current holdings and *expected prices*
      • Buy the flights in $G^*$ for which expected cost of post-
        poning commitment exceeds *expected benefit of post-
        poning commitment* (30 seconds)
      • Bid *hotel room expected marginal values* given hold-
        ings, new flights, and *expected hotel purchases* (30
        seconds)
**Last minute:** Buy remaining flights as needed by $G^*$
**In parallel (continuously):** Buy/sell entertainment tickets based
    on their *expected values*
```

Table 2. ATTac-2001's high-level algorithm. The italicized portions are described in the remainder of this section.

4.1. Goods Allocation

A core subproblem for TAC agents is the allocation problem: finding the most profitable allocation of goods to clients, G^*, given a set of owned goods and prices for all other goods. We denote the value of G^* (i.e. the score one would attain with G^*) as $v(G^*)$. G^* and $v(G^*)$ can be found (usually within 0.01 seconds on a 650 MHz Pentium II using the "LPsolve" package) via integer linear programming (Stone et al., 2001). An approximation to $v(G^*)$ can be obtained reliably quickly via LP relaxation.

4.2. Hotel Expected Marginal Values

Using the hotel price prediction module as described above, ATTac-2001 is equipped to determine its bids for hotel rooms.

Every minute, for each hotel room that is still open, ATTac-2001 assumes that that hotel will close next and computes the marginal value of that hotel room given the predicted closing prices of the other hotel rooms. If the hotel does not close next, then it assumes that it will have a chance to revise its bids. Since these predicted prices are represented as distributions of possible futures, ATTac-2001 samples from these distributions and averages the marginal values to obtain an expected marginal value. Using the full minute for computation between closing times (or 30 seconds if there are still flights to consider), ATTac-2001 divides the available time among the different open hotel rooms and generates as many price samples as possible for each hotel room. In the end, ATTac-2001 bids the expected marginal values for each of the hotels.

The algorithm is described precisely and with explanation in Table 3.

4.3. Other Price Predictor Uses

ATTac-2001 makes flight bidding decisions based on a cost-benefit analysis: in particular, ATTac-2001 computes the

```
• For each hotel (in order of increasing expected price):
• Repeat until time bound
    1. Generate a random hotel closing order (only other open hotels)
    2. *Sample* closing prices from predicted hotel price distributions
    3. Given these closing prices, compute $V_0, V_1, \ldots V_n$
      – $V_i \equiv v(G^*)$ if owning $i$ of the hotel
      – Estimate $v(G^*)$ with LP relaxation
      – Assume that no additional hotel rooms of this type can be bought
      – For other hotels, assume outstanding bids above sampled price are already owned (i.e. they cannot be withdrawn).
      – Note that $V_0 \leq V_1 \leq \ldots \leq V_n$: the values are monotonically increasing since having more goods cannot be worse in terms of possible allocations.
• The value of the $i$th copy of the room is the mean of $V_i - V_{i-1}$ over all the samples.
• Note further that $V_1 - V_0 \geq V_2 - V_1 \ldots \geq V_n - V_{n-1}$: the values differences are monotonically decreasing since each additional room will be assigned to the client who can derive the most value from it.
• Bid for one room at the value of the $i$th copy of the room for all $i$ such that the value is at least as much as the current price. Due to the monotonicity noted in the step above, no matter what the closing price, the desired number of rooms at that price will be purchased.
```

Table 3. The algorithm for generating bids for hotel rooms.

incremental cost of postponing bidding for a particular flight versus the value of delaying commitment. ATTac-2001 takes the cost of postponing commitment to be the average predicted cost of the flight over the next several whole minutes. It computes this cost based on a knowledge of the general form of the price adjustment function and the observed price points thus far in the current game.

Fundamentally, the benefit of postponing commitments to flights is that additional information about the eventual hotel prices becomes known. Thus, the benefit of postponing commitment is computed by sampling possible future price vectors and determining, on average, how much better the agent could do if it bought a different flight instead of the one in question. If it is optimal to buy the flight in all future scenarios (price vectors), then there is no value to delaying the commitment and the flight is purchased immediately. However, if there are many scenarios in which the flight is not the best one to get, the purchase is more likely to be delayed.

The algorithm for determining the benefit of postponing commitment is similar to that for determining the marginal value of hotel rooms.

4.4. Entertainment Expected Values

The core of ATTac-2001's entertainment-ticket-bidding strategy is again a calculation of the expected marginal values of each ticket. For each ticket, ATTac-2001 computes the expected value of having one more and one fewer of the ticket, again by sampling hotel prices and computing the relative values of $v(G^*)$ with varying ticket supplies. These calculations give bounds on the bid and ask prices it is willing to post. The actual bid and ask prices are a linear function of time remaining in the game: ATTac-2001 settles for a smaller and smaller profit from ticket transactions as the game goes on. Details of the functions mapping estimated ticket values and game time to bid and ask prices remained unchanged from the previous year's agent (Stone et al., 2001).

5. Results

5.1. TAC-01 Competition

Of the 19 teams that entered the qualifying round, ATTac-2001 was one of eight agents to make it to the finals on the afternoon of October 14th. The finals consisted of 24 games among the same eight agents. In raw score, ATTac-2001 ended up finishing a very close second to livingagents (Fritschi & Dorer, 2002), scoring an average of two fewer points per game.

After the competition, the TAC team at the University of Michigan conducted a regression analysis of the effects of the client profiles on agent scores. Using data from the seeding rounds, it was determined that agents did better when their clients had:

1. fewer total preferred travel days;
2. higher total entertainment values; and
3. a higher ratio of outer days (1 and 4) to inner (2 and 3) in preferred trip intervals.

Based on these significant measures, the games in the finals could be handicapped according to each agents' aggregate client profiles. Doing so indicated that livingagents' clients were much easier to satisfy than those of ATTac-2001, giving ATTac-2001 the highest handicapped score.

5.2. Controlled Experiments

ATTac-2001's success in the competition demonstrates its effectiveness as a complete system. However, since the different agents differ along several dimensions, the competition results cannot isolate the successful approaches. In this section we report on controlled experiments designed to test the efficacy of ATTac-2001's machine-learning approach to price prediction.

We attempted to determine experimentally how the quality of ATTac-2001's hotel price predictions affects its performance. To this end, we devised seven price prediction schemes, varying considerably in sophistication and inspired by approaches taken by other TAC competitors, and incorporated these schemes into our agent. We then played these seven agents against one another repeatedly, with regular retraining as described below.

Here are the seven hotel prediction schemes that we used, in decreasing order of sophistication:

- ATTac-2001$_s$: This is the "full-strength" agent based on boosting that was used during the tournament.
- ConditionalMean$_s$: This agent samples prices from the empirical distribution of prices from previously played games, conditioned on the closing time of the hotel room. In other words, it collects all historical hotel prices and breaks them down by the time at which the hotel closed (as well as room type, as usual). The price predictor then simply samples from the collection of prices corresponding to the given closing time.
- SimpleMean$_s$: This agent samples prices from the empirical distribution of prices from previously played games, without regard to the closing time of the hotel room (but still broken down by room type).
- ATTac-2001$_{ns}$ ConditionalMean$_{ns}$, SimpleMean$_{ns}$: These agents predict in the same way as their corresponding predictors above, but instead of returning a random sample from the estimated distribution of hotel prices, they deterministically return the expected value of the distribution.
- CurrentBid: This agent uses a very simple predictor that always predicts that the hotel room will close at its current price.

In every case, whenever the price predictor returns a price that is below the current price, we replace it with the current price (since prices cannot go down).

In our experiments, we added as an eighth agent EarlyBidder, inspired by the livingagents agent. EarlyBidder uses SimpleMean$_{ns}$, determines an optimal set of purchases, and then places bids for these goods at sufficiently high prices to ensure that they will be purchased ($1001 for all hotel rooms, just as livingagents did in TAC-01) right after the first flight quotes. It then never revises these bids.

Each of these agents require training, i.e., data from previously played games. However, we are faced with a sort of "chicken and egg" problem: to run the agents, we need to first train the agents using data from games involving the agent, but to get this kind of data, we need to first run the agents. To get around this problem, we ran the agents in phases. In Phase I, which consisted of 126 games, we used training data from the seeding, semifinals and finals rounds of TAC-01. In Phase II, lasting 157 games, we retrained the agents once every six hours using all of the data from the seeding, semifinals and finals rounds as well as all of the games played so far during the course of the experiment. Finally, in Phase III, lasting 622 games, we continued to retrain the agents once every six hours, but now using only data from games played during the course of the experiment, and not including data from the seeding, semifinals and finals rounds.

Table 4 shows how the agents performed in each of these phases. Much of what we observe in this table is consistent with our expectations. As expected, the more sophisticated boosting-based agents (ATTac-2001$_s$ and ATTac-2001$_{ns}$) clearly dominated the agents based on simpler prediction schemes. Moreover, with continued training, these agents improved markedly relative to EarlyBidder. We also see the performance of the simplest agent, CurrentBid, which does not employ any kind of training, significantly decline relative to the other data-driven agents.

Agent	*Relative Score*
Phase I	
EarlyBidder	140.3 ± 38.6
ATTac-2001$_{ns}$	105.2 ± 49.5
ATTac-2001$_s$	27.8 ± 42.1
ConditionalMean$_{ns}$	8.6 ± 41.2
SimpleMean$_{ns}$	-28.8 ± 45.1
CurrentBid	-33.7 ± 52.4
SimpleMean$_s$	-72.0 ± 47.5
ConditionalMean$_s$	-147.5 ± 35.6
Phase II	
EarlyBidder	152.8 ± 43.4
ATTac-2001$_{ns}$	131.6 ± 47.7
ATTac-2001$_s$	86.1 ± 44.7
ConditionalMean$_{ns}$	3.5 ± 37.5
SimpleMean$_{ns}$	-53.9 ± 40.1
SimpleMean$_s$	-71.6 ± 42.8
ConditionalMean$_s$	-91.4 ± 41.9
CurrentBid	-157.1 ± 54.8
Phase III	
ATTac-2001$_{ns}$	166.2 ± 20.8
ATTac-2001$_s$	122.3 ± 19.4
EarlyBidder	117.0 ± 18.0
SimpleMean$_{ns}$	-11.5 ± 21.7
SimpleMean$_s$	-44.1 ± 18.2
ConditionalMean$_{ns}$	-60.1 ± 19.7
ConditionalMean$_s$	-91.1 ± 17.6
CurrentBid	-198.8 ± 26.0

Table 4. The average relative scores for eight agents in the three phases of a controlled experiment in which the hotel prediction algorithm was varied. The relative score of an agent is its score minus the average score of all agents in that game.

On the other hand, there are some phenomena in this table that were very surprising to us. Most surprising was the failure of sampling to help. Our strategy relies heavily not only on estimating hotel prices, but also taking samples from the distribution of hotel prices. Yet these results indicate that using expected hotel price, rather than price samples, consistently performs better. We speculate that this may be because an insufficient number of samples are being used (due to computational limitations) so that the numbers derived from these samples have too high a variance. Another possibility is that the method of using samples to estimate scores consistently overestimates the expected score because it assumes the agent can behave with perfect knowledge for each individual sample—a property of our approximation scheme.

We were also surprised that ConditionalMean$_s$ and ConditionalMean$_{ns}$ eventually performed worse than the less sophisticated SimpleMean$_s$ and SimpleMean$_{ns}$. We do not fully understand why this happened. One possibility is that the simpler model happens to give predictions that are just as good as the more complicated model, perhaps because closing time is not terribly informative, or perhaps because the adjustment to price based on current price is more significant. The simpler model has the additional advantage that its statistics are based on all of the price data, regardless of closing time, whereas the conditional model makes each prediction based on only an eighth of the data (since there are eight possible closing times, each equally likely).

As a measure of the inaccuracy of the predictions made by the three non-sampling agents, we measured the root mean squared error of the predictions made in Phase III. These were: 56.0 for ATTac-2001$_{ns}$, 66.6 for SimpleMean$_{ns}$, 69.8 for CurrentBid and 71.3 for ConditionalMean$_{ns}$. Thus, we see that the lower the prediction accuracy (according to this measure), the higher the score (correlation -0.88).

6. Conclusion

In this paper, we have introduced a boosting-based algorithm for conditional density estimation. It is designed to be applicable in any scenario in which one wishes to estimate the probability distribution of real random variables associated with objects of some sort (such as feature vectors).

ATTac-2001 used this learning algorithm to compete successfully in TAC-01, a domain featuring simultaneous auctions for multiple interacting goods. We believe that price prediction and the modeling of price uncertainty will be key challenges in the design of agents for auctions and e-commerce, and our results indicate that prediction accuracy will be critical to achieving high performance. The generality of the technique described in this paper suggests that it will be widely applicable in such domains.

One such real application is the Federal Communications Commission's auctioning off of radio spectrum (Weber, 1997; Cramton, 1997). Especially for companies that are trying to achieve national coverage, the values of the different licenses interact in complex ways. Perhaps autonomous bidding agents will be able to affect bidding strategies in such future auctions. Indeed, in related research we have started down this path by creating bidding agents in a realistic FCC Auction Simulator (Csirik et al., 2001; Reitsma et al., 2002).

In a more obvious application, an extended version of ATTac-2001 could potentially become useful to real travel agents, or to end users who wish to create their own travel packages.

Acknowledgments

This work was partially supported by the United States-Israel Binational Science Foundation (BSF), grant number 1999038.

References

Collins, M., Schapire, R. E., & Singer, Y. (2002). Logistic regression, AdaBoost and Bregman distances. *Machine Learning*, *48*.

Cramton, P. C. (1997). The FCC spectrum auctions: An early assessment. *Journal of Economics and Management Strategy*, *6*, 431–495.

Csirik, J. A., Littman, M. L., Singh, S., & Stone, P. (2001). FAucS: An FCC spectrum auction simulator for autonomous bidding agents. *Electronic Commerce: Proceedings of the Second International Workshop* (pp. 139–151). Heidelberg, Germany: Springer Verlag.

Fritschi, C., & Dorer, K. (2002). Agent-oriented software engineering for successful TAC participation. *First International Joint Conference on Autonomous Agents and Multi-Agent Systems*. Bologna.

Reitsma, P. S. A., Stone, P., Csirik, J. A., & Littman, M. L. (2002). Randomized strategic demand reduction: Getting more by asking for less. *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems*.

Schapire, R. E., & Singer, Y. (1999). Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, *37*, 297–336.

Schapire, R. E., & Singer, Y. (2000). BoosTexter: A boosting-based system for text categorization. *Machine Learning, 39*, 135–168.

Stone, P., & Greenwald, A. (2002). The first international trading agent competition: Autonomous bidding agents. *Electronic Commerce Research*. To appear.

Stone, P., Littman, M. L., Singh, S., & Kearns, M. (2001). ATTac-2000: An adaptive autonomous bidding agent. *Journal of Artificial Intelligence Research*, *15*, 189–206.

Weber, R. J. (1997). Making more from less: Strategic demand reduction in the FCC spectrum auctions. *Journal of Economics and Management Strategy*, *6*, 529–548.

Wellman, M. P., Wurman, P. R., O'Malley, K., Bangera, R., Lin, S.-d., Reeves, D., & Walsh, W. E. (2001). A trading agent competition. *IEEE Internet Computing*, *5*, 43–51.