To Buy or Not to Buy: Mining Airline Fare Data to Minimize Ticket Purchase Price

Oren Etzioni
Dept. Computer Science
University of Washington
Seattle, Washington 98195
etzioni@cs.washington.edu

Craig Knoblock
Information Sciences Institute
University of Southern California
Marina del Rey, CA 90292
knoblock@isi.edu

Rattapoom Tuchinda
Dept. of Computer Science
University of Southern California
Los Angeles, CA 90089
pipet@isi.edu

Alexander Yates
Dept. Computer Science
University of Washington
Seattle, Washington 98195
ayates@cs.washington.edu

ABSTRACT

As product prices become increasingly available on the World Wide Web, consumers attempt to understand how corporations vary these prices over time. However, corporations change prices based on proprietary algorithms and hidden variables (e.g., the number of unsold seats on a flight). Is it possible to develop data mining techniques that will enable consumers to predict price changes under these conditions?

This paper reports on a pilot study in the domain of airline ticket prices where we recorded over 12,000 price observations over a 41 day period. When trained on this data, Hamlet — our multi-strategy data mining algorithm — generated a predictive model that saved 607 simulated passengers $283,904 by advising them when to buy and when to postpone ticket purchases. Remarkably, a clairvoyant algorithm with complete knowledge of future prices could save at most $320,572 in our simulation, thus HAMLET's savings were 88.6% of optimal. The algorithm's savings of $283,904 represents an average savings of 27.1% per simulated passenger for whom savings are possible. Our pilot study suggests that mining of price data available over the web has the potential to save consumers substantial sums of money per annum, at least until corporations begin to fight back.

1. INTRODUCTION AND MOTIVATION

Corporations often use complex policies to determine product pricing. Prices vary over time and due to promotions, coupons, and the use of multiple sales channels. The airline industry is one of the most sophisticated in its use of dynamic pricing strategies in an attempt to maximize its revenue. Airlines have many fare classes for seats on the same flight, use different sales channels (e.g., travel agents, priceline.com, consolidators), and frequently vary the price per seat over time based on a slew of factors including seasonality, availability of seats, competitive moves by other airlines, and more. The airlines are said to use proprietary software to compute ticket prices on any given day, but the algorithms used are jealously guarded trade secrets [19]. Hotels, rental car agencies, and other vendors with a "standing" inventory are increasingly using similar techniques.

As product prices become increasingly available on the World Wide Web, consumers have the opportunity to become more sophisticated shoppers. They are able to comparison shop efficiently and to track prices over time; they can attempt to identify pricing patterns and rush or delay purchases based on anticipated price changes (e.g., "I'll wait to buy, because they always have a big sale in the spring..."). In this paper we describe the use of data mining methods to help consumers with this task. We report on a pilot study in the domain of airline prices where an automatically learned model, based on price information available on the Web, was able to save consumers a substantial sum of money in simulation.

The paper addresses the following central questions:

- **What is the behavior of airline ticket price over time?** Do prices change frequently? Do they move in small increments or in large jumps? Do they tend to go up or down over time? Our pilot study enables us to begin to characterize the complex behavior of airline ticket prices.

- **What data mining methods are necessary to detect patterns in price data?** In this paper we consider reinforcement learning, rule learning, time series methods, and combinations of the above.

- **Can Web price tracking coupled with data mining save consumers money in practice?** Vendors vary prices based on numerous variables whose values are not available on the Web. For example, an airline may discount seats on a flight if the number of unsold seats, on a particular date, is high relative to the airline's model. However, consumers do not have access to the airline's model or to the number of available seats on the flights. Thus, *a priori*, price changes could appear random or unpredictable to a consumer tracking prices over the Web. In fact, we have found price changes to be surprisingly predictable.

The remainder of this paper is organized as follows. Section 2 describes our data collection mechanism and analyzes the basic characteristics of airline pricing in our data. Section 3 considers related work in the areas of Computational Finance and Time Series Analysis. Section 4 introduces our data mining methods and describes how each method was tailored to our domain. We investigated Rule learning [8], Q-learning [25], moving averages models [13], and the combination of these methods via stacked generalization [28]. Next, Section 5 describes our simulation and the performance of each of the methods on our test data. The section also reports on a sensitivity analysis to assess the robustness of our results to changes in the simulation. We conclude with a discussion of future work and a summary of the paper's contributions.

2. DATA COLLECTION

We collected airline fare data directly from the Orbitz web site. We chose the Orbitz site because the airlines that participate in the site must agree to make their lowest prices available through the web site (including Web fares). In order to extract the large amount of data required for our machine learning algorithms, we built a flight data collection agent that runs at a scheduled interval, automatically navigates through the Orbitz site, extracts the pricing data, and stores the result in a database.

We built our flight data collection agent using AgentBuilder [2] for wrapping web sites and Theseus for executing the agent [3]. AgentBuilder exploits machine learning technology [15] that allows a user to provide examples of the data to be extracted from a page. The system then learns the extraction rules to reliably convert the HTML data into XML data. The navigation through a web site is specified in AgentBuilder by showing the sequence of pages that should be followed to reach the required data. Once the system has learned the extraction rules and the navigation path, AgentBuilder compiles this into a Theseus plan. Theseus is a streaming dataflow execution system that supports highly optimized execution of a plan in a network environment. The system maximizes the parallelism across different operators and streams data between operations to support the efficient execution of plans with complex navigation paths and extraction from multiple pages.

For the purpose of our pilot study, we restricted ourselves to collecting data on non-stop flights for two routes: Los Angeles (LAX) to Boston (BOS) and Seattle (SEA) to Washington, DC (IAD). Our departure dates spanned January 2003. For each departure date, we began collecting pricing data 21 days in advance at three-hour intervals; data for each departure date was collected 8 times a day. Overall, we collected over 12,000 fare observations over a 41 day period for six different airlines including American, United, etc. We used three-hour intervals to limit the number of `http` requests to the Orbitz web site. For each flight, we recorded the *lowest* fare available for purchase through Orbitz.[1]

2.1 Pricing Behavior in Our Data

We found that the price of tickets on a particular flight can change as often as seven times in a single day. We categorize price change into two types: dependent price changes and independent price changes. Dependent changes occur when prices of similar flights (i.e. having the same origin and destination) from the same airline change at the same time. This type of change can happen as often as once or twice a day when airlines adjust their prices to maximize their overall revenue or "yield". Independent changes occur when the price of a particular flight changes independently of similar flights from the same airline. We speculate that this type of change results from the change in the seat availability of the particular flight. Table 1 shows the average number of change per flight aggregated over all airlines for each route. Overall, 762 price changes occurred across all the flights in our data. 63% of the changes can be classified as dependent changes based on the behavior of other flights by the same airline.

Route	*Avg. number of price changes*
LAX-BOS	6.8
SEA-IAD	5.4

Table 1: Average number of price changes per route.

We found that the round-trip ticket price for flights can vary significantly depending on airlines and destination. Table 2 shows the minimum price, maximum price, and the maximum difference in prices that can occur for flights on each route.

Route	*Min Price*	*Max Price*	*Max Price Change*
LAX-BOS	275	2524	2219
SEA-IAD	281	1668	1387

Table 2: Minimum price, Maximum price, and Maximum change in ticket price per route. All prices in this paper refer to the lowest fare available through Orbitz.

For many flights there are easily discernible price tiers where ticket prices fall into a relatively small price range. The number of tiers typically varies from two to four, depending on the airline and the particular flight. Even flights from the same airline with the same schedule (but with different departure dates) can have different numbers of tiers. For example, there are two price tiers for the flight in Figure 1, four price tiers in Figure 4 and three price tiers in Figure 2 and Figure 3.

Price matching does play an important role in airline pricing structure. Airlines use sophisticated software to track their competitors' pricing history and propose adjustments that optimize their overall revenue. To change the price, airlines need to submit the change to the Airline Tariff Publishing Company(ATPCO)[2] , the organization formed by leading airlines around the world that collects and distributes

[1]The data is freely available from the authors by request.

[2]see `http://www.atpco.net`.

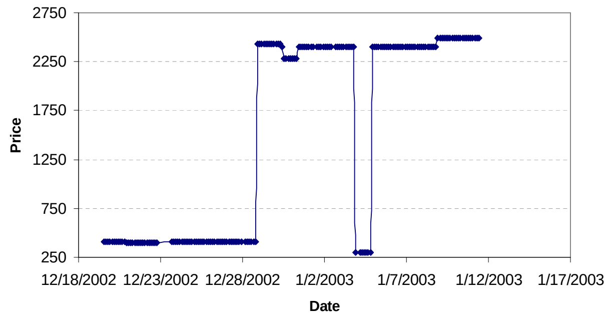

Figure 1: Price change over time for United Airlines roundtrip flight#168:169 LAX-BOS departing on Jan 12. This figure is an example of two price tiers and how consumers might benefit from the price drop.

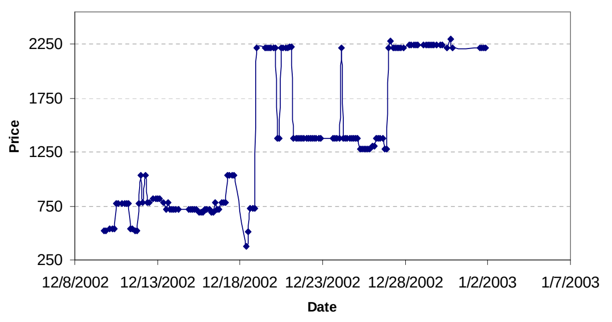

Figure 2: Price change over time for American Airlines roundtrip flight#192:223, LAX-BOS departing on Jan 2. This figure shows an example of rapid price fluctuation around the New Year.

airline pricing data. The whole process of detecting competitors' fare changes, deciding whether or not to match competitors' prices, and submitting the price update at ATPCO can take up to one day [19].

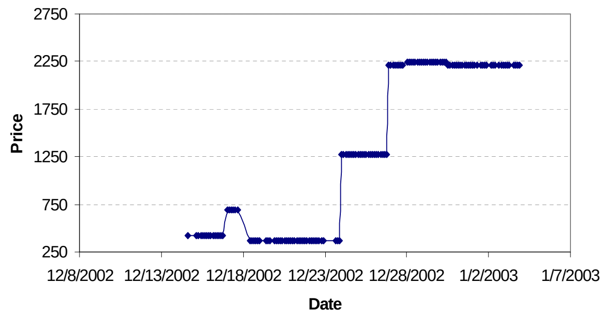

Figure 3: Price change over time for American Airlines roundtrip flight#192:223, LAX-BOS departing on Jan 7. This figure shows an example of three price tiers and low price fluctuation

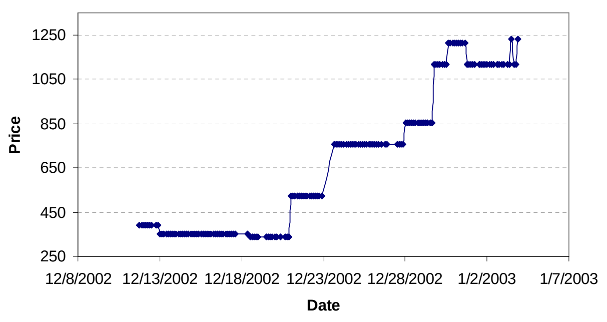

Figure 4: Price change over time for Alaska Airlines roundtrip flight#6:3, SEA-IAD departing on Jan 4. This figure shows an example of four price tiers.

There is no guarantee that if an airline drops their prices that competing airlines will match the price. However, we can use this information to help us predict if the price of a particular flight will drop if the price of a similar flight from a competing airline drops.

While price changes appear to be fairly orderly, flights that depart around holidays tend to fluctuate more. Figure 2 and Figure 3 show how pricing strategies differ between two flights from American Airlines that have the same schedule but fly on different dates. Figure 2 shows the flight that departs around the new year, while Figure 3 shows the flight that departs one week after the first flight. Both flights have the tier structure that we described earlier in this section, but the pricing in the first flight fluctuates more often.

In terms of pricing strategy, we can divide the airlines into two categories. The first category covers airlines that are big players in the industry, such as United Airlines, and American Airlines. The second category covers smaller airlines that concentrate on selling low-price tickets, such as Air Trans and Southwest. We have found that pricing policies tend to be similar for airlines that belong to the same category. Pricing for airlines in the first category is expensive and fluctuates often, while pricing for airlines in the second category is moderate and appears relatively stable. However, there are some policies that every airline seems to use. For example, airlines usually increase ticket prices two weeks before departure dates and ticket prices are at a maximum on departure dates.

3. RELATED WORK

To the best of our knowledge, previous work in the AI community on the problem of predicting product prices over time has been limited to the Trading Agent Competition (TAC) [27]. In 2002, TAC focused on the travel domain. TAC relies on a simulator of airline, hotel, and ticket prices and the competitors build agents to bid on these. The problem is different from ours since the competition works as an auction (similar to Priceline.com) where the prices of the goods are affected by the outcomes of other auctions. Whereas we gathered actual flight price data from Orbitz, TAC simulates flight prices using a stochastic process that follows a random walk with an increasingly upward bias. Also, the TAC auction of airline tickets assumes that the supply of airline tickets is unlimited. Several TAC competitors have explored a range of methods for price prediction including historical averaging, neural nets, and boosting. It is difficult to know how these methods would perform if reconfigured for our price mining task. In the future we plan to apply our approach to the competition to facilitate a direct comparison.

There has been some recent interest in temporal data mining (see [23] for a survey). However, the problems studied under this heading are often quite different from our own (e.g., [1]). Researchers have investigated subsequence matching, computing the similarity between two temporal sequences, discrete temporal sequences, and more. There has also been algorithmic work on time series methods within the data mining community (e.g., [4]). We discuss time series methods below.

Problems that are closely related to price prediction over time have been studied in Statistics under the heading of "time series analysis" [7, 13, 9] and in Computational Finance [20, 22, 21] under the heading of "optimal stopping problems". However, these techniques have not been used to predict price changes for consumer goods based on data available over the web. Moreover, we combine these techniques with rule learning techniques to improve their performance.

Computational Finance is concerned with predicting prices and making buying decisions in markets for stock, options, and commodities. Prices in such markets are not determined by a hidden algorithm, as in the product pricing case, but rather by supply and demand as determined by the actions of a large number of buyers and sellers. Thus, for example, stock prices tend to move in small incremental steps rather than in the large, tiered jumps observed in the airline data.

Nevertheless, there are well known problems in options trading that are related to ours. First, there is the early exercise of American Calls on stocks that pay dividends. The second problem is the exercise of American Puts on stocks that don't pay dividends. These problems are described in sections 11.12 and 7.6 respectively of [14]. In both cases, there may be a time before the expiration of an option at which its exercise is optimal. Reinforcement learning methods have been applied to both problems, and that is one reason we consider reinforcement learning for our problem.

Time series analysis is a large body of statistical techniques that apply to a sequence of values of a variable that varies over time due to some underlying process or structure [7, 13, 9]. The observations of product prices over time are naturally viewed as time series data. Standard data mining techniques are "trained" on a set of data to produce a predictive model based on that data, which is then tested on a separate set of test data. In contrast, time series techniques would attempt to predict the value of a variable based on its own history. For example, our moving average model attempts to predict the future changes in the price of a ticket on a flight from the flight's own price history.

There is also significant interest in bidding and pricing strategies for online auctions. For example, in [24] Harshit et al. use cluster analysis techniques to categorize the bidding strategies being used by the bidders. And in [17], Lucking-Reiley et al. explore the various factors that determine the final price paid in an online auction, such as the length of the auction, whether there is a reserve price, and the reputation of the seller. However, these techniques are not readily applicable to our price mining problem.

Comparison shopping "bots" gather price data available on the web for a wide range of products.[3] These are descendants of the Shopbot [11] which automatically learned to extract product and price information from online merchants' web sites. None of these services attempts to analyze and predict the behavior of product prices over time. Thus, the data mining methods in this paper complement the body of work on shopbots.

4. DATA MINING METHODS

In this section we describe the various data mining methods we investigated: Ripper [8], Q-learning [25], and time series [13, 9]. We then explain how our data mining algorithm, HAMLET, combines the results of these methods using a variant of stacked generalization [26, 28].

We divided our data into a training set and a test set. The training set consisted of fare observations for a set of flights over a 36 day period, and the test set was the fare observations for the set of chronologically later flights departing over a 6 day period. All of our experiments enforce the following essential invariant: all the observations in the training set were chronologically earlier than the observations in the test set. In this way, we ensured that we were using the past to predict the future, but not vice versa.

4.1 Rule Learning

Our first step was to run the popular Ripper rule learning system [8] on our training data. Ripper is an efficient separate and conquer rule learner. We represented each price observation to Ripper as a vector of the following features:

- Flight number.
- Number of hours until departure.
- Current price.
- Airline.
- Route (LAX-BOS or SEA-IAD).

The class labels on each training instance were "buy" or "wait."

We considered a host of additional features derived from the data, but they did not improve Ripper's performance. We did not represent key variables like the number of unsold seats on a flight, whether an airline is running a promotion, or seasonal variables because HAMLET did not have access to this information. However, see Section 6 for a discussion of how HAMLET might be able to obtain this information in the future.

Some sample rules generated by Ripper are shown in Figure 5.

Ripper has numerous parameters, and we explored the appropriate settings for them extensively. The two parameters whose settings resulted in substantial improvement to Ripper's results are the parameter that causes Ripper to simplify its model and the parameter which sets the ratio of the cost of a false negative to the cost of a false positive. We set that ratio of misclassified "buy" training instances to misclassified "wait" training instances to 0.2 with good

[3]See, for example, `froogle.google.com` and `mysimon.com`.

IF *observation count* $>=$ 84 AND *price* $>=$ 2223
AND *route* $= LAX - BOS$ THEN *wait*

IF *airline* $=$ *United* AND *price* $>=$ 360
AND *observation count* $>=$ 146 THEN *wait*

Figure 5: Sample Ripper rules. Observation count refers to the number of price observations remaining before flight departure.

results. This choice is based on the intuition that misclassifying too many "wait" examples could lead to the flight selling out, which we wanted to avoid. In future work we plan to implement MetaCost [10] and assess to what extent it improves Ripper's performance in this domain.

4.2 Q-learning

As our next step we considered Q-learning, a species of reinforcement learning [25]. Reinforcement learning seems like a natural fit because in each state we have to decide whether to buy or to wait. Yet the reward (or penalty) associated with the decision is only determined later, when HAMLET determines whether it saved or lost money through its buying policy. Reinforcement learning is also a popular technique in Computational Finance [20, 22, 21].

The standard Q-learning formula is:

$$Q(a,s) = R(s,a) + \gamma max_{a'}(Q(a',s'))$$

Here, $R(s,a)$ is the immediate reward, γ is the discount factor for future rewards, and s' is the state resulting from taking action a in state s. In our domain there are exactly two possible actions in each state: b for 'buy' and w for 'wait'.

Of course, the particular reward function used is critical to the success (or failure) of Q-learning. In our study, the reward associated with b is the negative of the ticket price at that state, and the state resulting from b is a terminal state so there is no future reward. The immediate reward associated with w is zero as long as the flight does not sell out in the next time step. We set $\gamma = 1$, so we do not discount future rewards.

To discourage the algorithm from learning a model that waits until flights sell out, we introduce a "penalty" for such flights in the reward function. Specifically, in the case where the flight does sell out on the next turn, we make the immediate reward for waiting a negative constant whose absolute value is substantially greater than the price for any flight. In our data set, the maximum price of any ticket was \$2524, so we set the reward for reaching a sold-out state to be -300,000. This setting can best be explained below, after we introduce a notion of equivalence classes among states.

In short, we define the Q function by

$$\begin{aligned} Q(b,s) &= -price(s) \\ Q(w,s) &= \begin{cases} -300000 & \text{if flight sells out after } s. \\ max(Q(b,s'),Q(w,s')) & \text{otherwise.} \end{cases} \end{aligned}$$

To generalize from the training data we used a variant of the averaging step described in [18]. More specifically, we defined an equivalence class over states, which enabled the algorithm to train on a limited set of observations of the class and then use the learned model to generate predictions for other states in the class.

To define our equivalence class we need to introduce some notation. Airlines typically use the same flight number (e.g., UA 168) to refer to multiple flights with the same route that depart at the same time on different dates. Thus, United flight 168 departs once daily from LAX to Boston at 10:15pm. We refer to a particular flight by a combination of its flight number and date. For example, UA168-Jan7 refers to flight 168 which departs on January 7th, 2003. Since we observe the price of each flight eight times in every 24 hour period, there are many price observations for each flight. We distinguish among them by recording the number of remaining observations until the flight departs. Thus, UA168-Jan7-40 is the price observation for flight UA168, departing on January 7, which was recorded on January 2nd. Our equivalence class is the set of states with the same flight number, and the same number of remaining observations, but different departure dates. Thus, the states denoted UA168-Jan7-40 and UA168-Jan10-40 are in the same equivalence class, but the state UA168-Jan7-39 is not. We denote that s and s^* are in the same equivalence class by $s \sim s^*$. Thus, each state is represented as a triple of the form UA168-Jan10-40.

Thus, our revised Q-learning formula is:

$$Q(a,s) = Avg_{s^* \sim s}(R(s^*,a) + max_{a'}(Q(a',s')))$$

The reason for choosing -300,000 is now more apparent: the large penalty can tilt the average toward a low value, even when many Q values are being averaged together. Suppose, for example, that there are ten training examples in the same equivalence class, and each has a current price of \$2,500. Suppose now that nine of the ten examples drops in price to \$2,000 at some point in the future, but the tenth example sells out on the next turn. The Q value for waiting in any state in this equivalence class will be $(9 * -2000 + -300000)/10 = -31800$, or still much less then the Q value for any equivalence class where no flight sells out on the next turn. Thus the choice of reward for a flight that sells out will determine how willing the Q-Learning algorithm will be to risk waiting when there's a chance a flight may sell out. Using a hill climbing search in the space of penalties, we found -300,000 to be locally optimal.

Q-learning can be very slow, but we were able to exploit the structure of the problem and the close relationship between dynamic programming and reinforcement learning (see [25]) to complete the learning in three passes over the training set, which bodes well for scaling the algorithm to much larger data sets.

The reinforcement learning problem we face has a particularly nice structure, in which the value of $Q(b,s)$ depends only on the price in state s, and the value of $Q(w,s)$ depends only on the Q values of exactly one other state: the state containing the same flight number and departure date but with three hours less time left until departure. Applying dynamic programming is thus straightforward. We order the training data from least amount of time left until most, and for each state visited, we record the negative of the price for the value of $Q(b,s)$ and look up the values for $Q(b,s')$ and $Q(w,s')$ to determine the value of $Q(w,s)$. Because of the ordering of the training data, $Q(b,s')$ and $Q(w,s')$ will always be defined by the time we visit state s.

From the above description one can see that the initial training step can be accomplished in a single pass over the data. In order to convert the learned Q values to their generalized form – that is, in order to compute averages over states in the same equivalence class – we perform a postprocessing step that takes two additional passes over the table of Q values.

The output of Q-learning is the learned policy, which determines whether to buy or wait in unseen states by mapping them to the appropriate equivalence class.

4.3 Time Series

Time series analysis is a large and diverse subfield of Statistics whose goal is to detect and predict trends. In this paper, we investigated a first order moving average model method with surprisingly good results. The basic idea is to predict the difference between the present price observation at time t, p_t, and the immediately subsequent one p_{t+1} based on the differences between prior price observations. Thus, whereas Q-learning and Ripper attempt to generalize from the behavior of a set of past flights in the training data to the behavior of future flights, the moving average model attempts to predict the price behavior of a flight in the test data based on its own history.

We derive our model following closely the derivation in [13]. A stationary time series ϵ_t is said to be *white noise* if

$$\rho(\epsilon_t, \epsilon_s) = 0, \forall t \neq s$$

where ρ is the correlation between the two random variables ϵ_t and ϵ_s. The subscripts s and t represent two distinct time points. This derivation assumes that the mean of ϵ_t is zero.

A *simple moving average* is a series x_t generated from a white noise series ϵ_t by the rule:

$$x_t = \epsilon_t + \beta\epsilon_{t-1}$$

Now, suppose that we want to predict x_{t+1} from the observations $x_1...x_t$ for the simple moving average $x_t = \epsilon_t + \beta\epsilon_{t-1}$. Since $x_{t+1} = \epsilon_{t+1} + \beta\epsilon_t$ and since the best prediction of ϵ_{t+1} is zero, the optimal prediction of x_{t+1} is $\beta\epsilon_t$. To obtain a useful prediction, however, we must express ϵ_t in terms of $x_1, \ldots x_t$. Since the optimal prediction for x_t is $\beta\epsilon_{t-1}$. , and since $x_t = \epsilon_t + \beta\epsilon_{t-1}$, we have:

$$\epsilon_t = x_t - \beta\epsilon_{t-1},$$

which is the difference between x_t and its optimal prediction. From this equation, we can derive a simple recursive procedure to calculate each ϵ_i and then make the prediction for x_{t+1} by calculating $\beta\epsilon_t$. This derivation assumes that β is fixed and known. We used gradient descent to fit β to the training data, and found that setting it to -1 yielded the best performance.

Thus, we have a uni-variate time series of price differences, and we predict the next price difference to be $\beta\epsilon_{t-1}$. We use the following simple decision rule based on the time series: if we predict that the price is about to go up at the next time step then buy, otherwise wait. More formally:
IF $\beta\epsilon_{t-1} \geq 0$ THEN *buy* ELSE *wait.*
Thus, the time series model makes its decisions based on a one-step prediction of the ticket price change.

4.4 Stacked Generalization

In the last decade or so ensemble-based learning techniques such as bagging [5], boosting [12], and stacking [26, 28], which combine the results of multiple generalizers, have been shown to improve generalizer accuracy on many data sets. In our study, we investigated multiple data mining methods with very different characteristics (Ripper, Q-learning, and time series) so it makes sense to combine their outputs.

Let TS be the output of the Time Series algorithm,
and let QL be the output of Q-Learning.

IF *observation count* >= 160 AND *airline* = *United*
AND *price* >= 360 AND TS = *buy* AND QL = *wait*
THEN *wait*

Figure 6: A sample rule generated by Hamlet.

We preferred stacking to voting algorithms such as weighted majority [16] or bagging [5] because we believed that there were identifiable *conditions* under which one method's model would be more successful than another. See, for example, the sample rule in Figure 6.

Standard stacking methods separate the original vector representation of training examples (*level-0* data in Wolpert's terminology), and use the class labels from each level-0 generalizer, along with the example's true classification as input to a meta-level (or *level-1*) generalizer. To avoid over-fitting, "care is taken to ensure that the models are formed from a batch of training data that does not include the instance in question" [26].

In our implementation of stacking, we collapsed level-0 and level-1 features. Specifically, we used the feature representation described in Section 4.1 but added three additional features corresponding to the class labels (buy or wait) computed for each training example by our level-0 generalizers. To add our three level-1 features to the data, we applied the model produced by each base-level generalizer (Ripper, Q-learning, and time series) to each instance in the training data and labeled it with "buy" or "wait". Thus, we added features of the form TS = buy (time series says to buy) and QL = wait (Q-learning says to wait).

We then used Ripper as our level-1 generalizer, running it over this augmented training data. We omitted leave-one-out cross validation because of the temporal nature of our data. Although a form of cross validation is possible on temporal data, it was not necessary because each of our base learners did not appear to overfit the training data.

Our stacked generalizer was our most successful data mining method as shown in Table 3 and we refer to it as HAMLET.

4.5 Hand-Generated Rule

After we studied the data in depth and consulted with travel agents, we were able to come up with a fairly simple policy "by hand". We describe it below, and include it in our results as a baseline for comparison with the more complex models produced by our data mining algorithms.

The intuition underlying the hand-generated model is as follows. First, to avoid sell outs we do not want to wait too long. By inspection of the data, we decided to *buy* if the price has not dropped within 7 days of the departure date. We can compute an expectation for the lowest price of the flight in the future based on similar flights in the training data. If the current price is higher than the expected minimum then it is best to wait. Otherwise, we buy.

More formally, let $MinPrice(s, t)$ of a flight in the training set denote the minimum price of that flight over the interval starting from s days before departure up until time t (or until the flight sells out). Let $ExpPrice(s, t)$

for a particular flight number denote the average over all $MinPrice(s, t)$ for flights in the training set with that flight number. Suppose a passenger asks at time t_0 to buy a ticket that leaves in s_0 days, and whose current price is $CurPrice$. The hand-generated rule is shown in Figure 7.

IF $ExpPrice(s_0, t_0) <= CurPrice$
AND $s_0 > 7$ days THEN *wait*
ELSE *buy*

Figure 7: Hand-crafted rule for deciding whether to wait or buy.

5. EXPERIMENTAL RESULTS

In this section we describe the simulation we used to assess the savings due to each of the data mining methods described earlier. We then compare the methods in Tables 3 and 4, perform a sensitivity analysis of the comparison along several dimensions, and consider the implications of our pilot study.

5.1 Ticket Purchasing Simulation

The most natural way to assess the quality of the predictive models generated by the data mining methods described in Section 4 is to quantify the savings that each model would generate for a population of passengers. For us, a passenger is a person wanting to buy a ticket on a particular flight at a particular date and time. It is easy to imagine that an online travel booking site such as Expedia or Travelocity would offer each passenger to take charge of the *timing* of his or her ticket purchase in order to save the passenger money.

Since we are not yet ready to expose HAMLET to real passengers, we simulated passengers by generating a uniform distribution of passengers wanting to purchase tickets on various flights as a function of time. The simulation generated one passenger for each fare observation in our set of test data. The total number of passengers was 4488. Thus, each simulated passenger has a particular flight for which they need to buy a ticket and an earliest time point at which they could purchase that ticket. The passenger decides, based on the predictive model used, whether to immediately buy or to wait. We recorded for each simulated passenger, and for each predictive model considered, the price of the ticket purchased, and the optimal price for that passenger given their earliest time point and the subsequent price behavior for that flight. Thus, we are able to determine the dollar benefit (or cost) for each passenger.

In some cases, delaying a purchase will result in the desired flight selling out before HAMLET purchases the ticket for the passenger. In such cases, we can determine the flight closest in departure time to the one the passenger requested and have HAMLET buy a ticket for that flight. This is clearly an undesirable outcome, but if it's infrequent enough, it seems reasonable to believe that *some* passengers would be willing to take that risk in order to save money. In our simulation, HAMLET waited until flights sold out only 0.71 % of the time. Moreover, in virtually all of these cases HAMLET was able to re-book the passenger on a different non-stop flight within two hours of the original departure time.[4] In all cases, the passenger was re-booked on another non-stop flight on the same day.

[4]Of course, there's no guarantee of this, so passengers may decline to use HAMLET around major holidays or other situations where large blocks of flights can sell out.

The price of the re-booked flights was close to, and in many cases smaller than, the price a passenger would pay if she bought immediately on the original flight. However, to eliminate any effect of sold out flights on the savings generated by different data mining methods, we omitted the price of tickets on these flights from the savings calculation. As a result, a sold out flight cannot result in any savings. In summary, each model below is evaluated along two separate dimensions: the amount of savings generated on requested flights and the percentage of sold out flights.

Method	Savings	Losses	Net Savings
Optimal	\$320,572	\$0	\$320,572
By hand	\$228,318	\$35,329	\$192,989
Ripper	\$261,562	\$4,826	\$256,736
Time Series	\$283,515	\$0	\$283,515
Q-learning	\$228,663	\$46,873	\$181,790
HAMLET	\$288,115	\$4211	\$283,904

Table 3: Net Savings by Algorithm.

5.2 Experimental Results

Table 3 shows the savings achieved in our simulation by the following predictive models:

- **Optimal:** This model represents the maximal possible savings, which are computed by a "clairvoyant" algorithm with perfect information about future prices, and which obtained the best possible purchase price for each passenger.
- **By hand:** This model was hand-generated by one of the authors after consulting with travel agents and throughly analyzing our training data (see Figure 7).
- **Time series:** This model was generated by the moving average method described earlier.
- **Ripper:** This model was generated by Ripper.
- **Q-learning:** This model was generated by our Q-learning method.
- **Hamlet:** This model was generated by our stacking generalizer which combined the results of Ripper, Q-learning, and Time series.

Table 3 shows a comparison of the different methods. Note that the savings we are reporting are *net* savings. With the exception of **Optimal**, each predictive model lost money for some passengers in cases where delayed purchases led to higher fares. When we compute each method's savings we subtract the total money lost from the total money saved to arrive at the net savings figure as shown in the rightmost column in Table 3. We see that HAMLET outperformed each of the learning methods as well as the hand-crafted model to achieve a net savings of \$283,904. The net savings of the time series method are quite close, but that figure is somewhat misleading as explained below.

The net savings do not tell the whole story for several reasons. First, each algorithm's predictive model resulted in some flights selling out when the passenger waited too

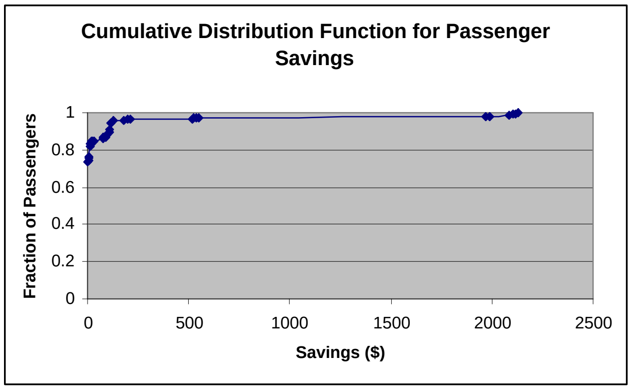

Figure 8: Fraction of passengers who can save less than or equal to the dollar value on the X axis. 75% of the passengers cannot save anything.

long to buy a ticket. For example, the time series method achieved impressive net savings, but resulted in sold-out flights 38.9% of the time. In fact, the net loss of \$0 is due to the fact that instead of buying at a more expensive price, the time series method simply waited until flights sold out. Second, we would like to know how the savings compare to the optimal savings that a clairvoyant algorithm could achieve. Third, we would like to understand what percentage of the ticket price is saved. For close to 75% of the passengers in our test set, savings was not possible because prices never dropped from the point that the passenger requested a ticket until the flight departed (see Figure 8). Thus, we report the percent savings in ticket prices over both the set of flights where savings was possible ("feasible flights") and over all flights. See Table 4 for a compilation of these percentages.

Table 4 shows that HAMLET achieved a remarkable 88.6% of the optimal savings possible. HAMLET's waiting policy led to flights selling out 0.71% of the time. In virtually all of these cases, the passenger was re-booked on another non-stop flight within two hours. In all cases, the passenger was re-booked on another non-stop flight on the same day. The price of the re-booked flight was lower than the original price in all cases in our data, though there is no guarantee that this would be the case in general. So as to avoid increase our savings figure due to sold out flights, the savings on these flights were omitted from the savings calculation.[5]

While additional experiments on larger data sets are clearly necessary, our pilot results are promising as they show a 6.3% savings on large sums of money spent by travelers annually. Furthermore, on "feasible flights", where savings is possible, that percentage shoots up to 27.1%. Because we only gathered data for 21 days before each flight in our test set, passengers arrived at most 21 days before a flight in our simulation. We conjecture that if we gather a more extensive data set, we would find more possibilities to save money for passengers who currently buy their tickets 30, 45, or more days before the flight date.

5.3 Sensitivity Analysis

To test the robustness of our results to changes in our simulation, we varied two key parameters. First, we changed the distribution of passengers requesting flight tickets. Second, we changed the model of a passenger from one where a passenger wants to purchase a ticket on a particular flight to one where a passenger wants to fly at any time during a three hour interval. The interval model is similar to the interface offered at many travel web sites where a potential buyer specifies if they want to fly in the morning, afternoon, or evening.

We used the following distributions to model the times at which passengers "arrive" and request a ticket purchase:

- **Uniform:** a uniform distribution of simulated passengers over our test data;
- **Linear Decrease:** a distribution in which the number of passengers arriving at the system decreased linearly as the amount of time left before departure decreased;
- **Quadratic Decrease:** a distribution like Linear Decrease, but with a quadratic relationship;
- **Square Root Decrease:** a distribution like Linear Decrease, but with a square root relationship;
- **Linear Increase:** a distribution like Linear Decrease, except that the number of passengers increase as the amount of time left before departure decreased;
- **Quadratic Increase:** a distribution like Linear Increase, but with a quadratic relationship;
- **Square Root Increase:** a distribution like Linear Increase, but with a square root relationship.

Table 5 reports the savings, as a percentages of optimal savings, under the different distributions. As the bottom line in the table shows, HAMLET's savings varied by at most 5.2%. HAMLET's savings were above 83% of optimal, and bigger than those of the other methods, under all distributions. Moreover, HAMLET's savings were substantially less sensitive to the passenger distribution than the savings of the other methods.

Changing the model of a passenger's travel request from asking to purchase a ticket on a particular flight to requesting to purchase a ticket at any time during a particular three hour interval is shown in Table 6. This different model yields somewhat larger changes in HAMLET's performance, but does not change our results qualitatively. HAMLET still achieves a substantial percentage of the optimal savings (74.5%) and its sold out flights drop to only 0.1%. Furthermore, HAMLET still substantially outperforms the other data mining methods.

Method	Net Savings	% of Optimal	% Sold Out
Optimal	\$323,802	100%	0%
By hand	\$163,523	55.5%	0%
Ripper	\$197,541	61.0%	0%
Time Series	\$198,541	61.3%	6.3%
Q-Learning	\$164,014	50.7%	0.2%
HAMLET	\$241,262	74.5%	0.1%

Table 6: Performance of algorithms on multiple flights over three hour interval.

Overall, our analysis confirms that HAMLET's performance on the test data is robust to the parameters we varied.

[5]The number of sold out flights for all methods (except Time Series) is sufficiently small that this conservative decision only has a minuscule effect on the results reported.

Method	Percent Savings (feasible flights)	Percent Savings (all flights)	Percent of Optimal Savings	Percent Sold Out
Optimal	30.6%	7.1%	100%	0%
By hand	21.8%	4.3%	60.2%	0.36%
Ripper	24.6%	5.7%	80.1%	0.45%
Time Series	27.1%	6.3%	88.4%	38.9%
Q-learning	21.7%	4.0%	56.7%	0.46%
HAMLET	27.1%	6.3%	88.6%	0.71%

Table 4: Comparison of Savings on a Percentage basis.

Distribution	By hand	Q-Learn	Time Series	Ripper	HAMLET
Quadratic Decrease	50.2%	48.9%	42.8%	69.4%	83.4%
Linear Decrease	56.1%	54.3%	55.0%	75.8%	86.6%
Sqrt Decrease	58.9%	56.6%	59.1%	78.2%	87.7%
Uniform	60.2%	57.4%	60.4%	80.1%	88.6%
Sqrt Increase	63.8%	60.5%	62.8%	83.5%	90.2%
Linear Increase	65.5%	61.4%	64.6%	85.7%	91.1%
Quadratic Increase	67.2%	61.2%	64.3%	87.4%	91.5%
Maximal deviation from Uniform	10.0%	8.5%	17.6%	10.7%	5.2%

Table 5: Sensitivity of Algorithms to Distribution of Passengers. The bottom line records the deviation of the algorithm's results from the uniform distribution.

6. FUTURE WORK

There are several promising directions for future work on price mining. In the next 3 months we plan to perform a more comprehensive study on airline pricing with data collected over a longer period of time and over more routes. We plan to include multi-leg flights in this new data set. The pricing behavior of multi-leg flights is different than that of non-stop flights because each leg in the flight can cause a change in the price, and because pricing through airline hubs appears to behave differently as well.

We also plan to exploit other sources of information to further improve HAMLET's predictions. We do not currently have access to a key variable — the number of unsold seats on a flight. However, there are seating charts available on some sites (e.g., `ual.com`), which provide information about how full a flight is even before purchasing a ticket. We can then combine this availability information with our analysis of independent and dependent price changes to more accurately predict independent price changes and to all but eliminate the chances of HAMLET waiting until a flight is sold out.

To use the methods in this paper on the full set of flights available through a site such as Orbitz would require collecting vast amounts of data. One possible way to address this problem is to build agents on demand that collect the required data to make price predictions for on a particular future flight on a particular day. The agents would still need to collect data for multiple flights, but the amount of data would be much smaller. This type of agent would fit well within the Electric Elves system [6], which deploys a set of personalized agents to monitor various aspects of a trip. For example, Elves can notify you if your flight is delayed or canceled or let you know if there is an earlier connecting flight to your destination.

Beyond airline pricing, we believe that the techniques described in this paper will apply to other product categories. In the travel industry, hotels and car rental agencies employ many of the same pricing strategies as the airlines and it would be interesting to see how much HAMLET can save in these product categories. Similarly, online shopping sites such as Amazon and Wal-mart are beginning to explore more sophisticated pricing strategies and HAMLET will allow consumers to make more informed decisions. Finally, reverse auction sites, such as half.com, also provide an opportunity for HAMLET to learn about pricing over time and make recommendations about purchasing an item right away or waiting to buy it. In general, price mining over time provides a new dimension for comparison shopping engines to exploit.

We recognize that if a progeny of HAMLET would achieve wide spread use it could start to impact the airlines' (already slim) profit margins. Could the airlines introduce noise into their pricing patterns in an attempt to fool a price miner? While we have not studied this question in depth, the obvious problem is that changing airfares in order to fool a price miner would also affect consumers directly accessing the site. Thus, the airlines have to show the prices that they actually want to charge for tickets. Of course, more prosaic methods of trying to block a price miner such as placing prices inside GIF files or blocking the IP address of the price miner are possible and would result in an "arms race" of sorts between price miners and online merchants.

7. CONCLUSION

This paper reported on a pilot study in "price mining" over the web. We gathered airfare data from Orbitz and showed that it is feasible to predict price changes for flights based on historical fare data. Despite the complex algorithms used by the airlines, and the absence of information on key variables such as the number of seats available on a flight, our data mining algorithms performed surprisingly well. Most notably, our HAMLET data mining method achieved 88.6% of the possible savings by appropriately timing ticket purchases. HAMLET's waiting policy resulted in sold out flights only 0.71% of the time.

Our algorithms were drawn from Statistics (time series methods), Computational Finance (reinforcement learning) and classical machine learning (Ripper rule learning). Each algorithm was tailored to the problem at hand (e.g., we devised an appropriate reward function for reinforcement learning), and the algorithms were combined using a variant of stacking to improve their predictive accuracy.

Additional experiments on larger airfare data sets and in other pricing domains (e.g., hotels, reverse auctions) are important, but this initial pilot study provides evidence for the potential of price mining algorithms to save consumers substantial amounts of money using data that is freely available on the Internet. We believe that price mining of this sort is a fertile area for future research.

8. REFERENCES

[1] R. Agrawal and R. Srikant. Mining sequential patterns. In P. S. Yu and A. S. P. Chen, editors, *Eleventh International Conference on Data Engineering*, pages 3–14, Taipei, Taiwan, 1995. IEEE Computer Society Press.

[2] J. L. Ambite, G. Barish, C. A. Knoblock, M. Muslea, J. Oh, and S. Minton. Getting from here to there: Interactive planning and agent execution for optimizing travel. In *Proceedings of the Fourteenth Conference on Innovative Applications of Artificial Intelligence (IAAI-2002)*, pages 862–869, AAAI Press, Menlo Park, CA, 2002.

[3] G. Barish, D. DiPasquo, C. A. Knoblock, and S. Minton. Dataflow plan execution for software agents. In *Proceedings of the Fourth International Conference on Autonomous Agents (Agents-2000)*, Barcelona, Spain, 2000.

[4] D. Berndt and J. Clifford. Finding patterns in time series: a dynamic programming approach. In U. Fayyad, G. Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*. AAAI Press, 1996.

[5] L. Breiman. Bagging predictors. *Machine Learning*, 24:123–140, 1996.

[6] H. Chalupsky, Y. Gil, C. A. Knoblock, K. Lerman, J. Oh, D. V. Pynadath, T. A. Russ, and M. Tambe. Electric elves: Applying agent technology to support human organizations. In *Proceedings of the Conference on Innovative Applications of Artificial Intelligence*, 2001.

[7] C. Chatfield. *The Analysis of Time Series: An Introduction*. Chapman and Hall, London, UK, 1989.

[8] W. W. Cohen. Fast effective rule induction. In A. Prieditis and S. Russell, editors, *Proc. of the 12th International Conference on Machine Learning*, pages 115–123, Tahoe City, CA, July 9–12, 1995. Morgan Kaufmann.

[9] F. Diebold. *Elements of Forecasting*. South-Western College Publishing, 2nd edition, 2000.

[10] P. Domingos. Metacost: A general method for making classifiers cost-sensitive. In *KDD*, pages 155–164, 1999.

[11] R. Doorenbos, O. Etzioni, and D. Weld. A scalable comparison-shopping agent for the World-Wide Web. In *Proc. First Intl. Conf. Autonomous Agents*, pages 39–48, 1997.

[12] Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *Proceedings of the Thirteenth International Conference on Machine Learning*, pages 148–156, Bari, Italy, 1996. Morgan Kaufmann.

[13] C. W. J. Granger. *Forecasting in Business and Economics*. Harcourt Brace, second edition, 1989.

[14] J. C. Hull. *Options, Futures, and Other Derivatives*. Prentice Hall College Div, 5th edition, 2002.

[15] C. A. Knoblock, K. Lerman, S. Minton, and I. Muslea. Accurately and reliably extracting data from the web: A machine learning approach. In P. S. Szczepaniak, J. Segovia, J. Kacprzyk, and L. A. Zadeh, editors, *Intelligent Exploration of the Web*, pages 275–287. Springer-Verlag, Berkeley, CA, 2003.

[16] N. Littlestone and M. K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212–261, February 1994.

[17] D. Lucking-Reiley, D. Bryan, N. Prasad, and D. Reeves. Pennies from ebay: The determinants of price in online auctions. Technical report, University of Arizona, 2000.

[18] S. Mahadevan. Average reward reinforcement learning: Foundations, algorithms, and empirical results. *Machine Learning*, 22(1-3):159–195, 1996.

[19] S. McCartney. Airlines Rely on Technology To Manipuate Fare Structure. *Wall Street Journal*, November 3 1997.

[20] J. Moody and M. Saffell. Reinforcement learning for trading systems and portfolios. In *KDD*, pages 279–283, 1998.

[21] J. Moody and M. Saffell. Minimizing downside risk via stochastic dynamic programming. In Y. S. Abu-Mostafa, B. LeBaron, A. W. Lo, and A. S. Weigend, editors, *Computational Finance 1999*, Cambridge, MA, 2000. MIT Press.

[22] J. Moody and M. Saffell. Learning to trade via direct reinforcement. In *IEEE Transactions on Neural Networks, Vol. 12, No. 4*, 2001.

[23] J. F. Roddick and M. Spiliopoulou. A bibliography of temporal, spatial and spatio-temporal data mining research. *SIGKDD Explorations*, 1(1):34–38, 1999.

[24] H. S. Shah, N. R. Joshi, A. Sureka, and P. R. Wurman. Mining for bidding strategies on ebay. In *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2003.

[25] R. S. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[26] K. M. Ting and I. H. Witten. Issues in stacked generalization. *Journal of Artificial Intelligence Research*, 10:271–289, 1999.

[27] M. P. Wellman, D. M. Reeves, K. M. Lochner, and Y. Vorobeychik. Price prediction in a trading agent competition. Technical report, University of Michigan, 2002.

[28] D. Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 1992.