Real-time Automatic Price Prediction for eBay Online Trading

Ilya Raykhel and **Dan Ventura**
Computer Science Department
Brigham Young University
iraykhel@gmail.com, dan@cs.byu.edu

Abstract

We develop a system for attribute-based prediction of final (online) auction pricing, focusing on the eBay laptop category. The system implements a feature-weighted k-NN algorithm, using evolutionary computation to determine feature weights, with prior trades used as training data. The resulting average prediction error is 16%. Mostly automatic trading using the system greatly reduces the time a reseller needs to spend on trading activities, since the bulk of market research is now done automatically with the help of the learned model. The result is a 562% increase in trading efficiency (measured as profit/hour).

Introduction

We consider the problem of product valuation for online auctions and present a system that automates the market research step that determines a "good" price for an item on eBay. For the high-volume reseller, automating the (intelligent) evaluation of a product offered for sale and predicting its final auction price significantly reduces human involvement, ultimately resulting in higher trading throughput and reduced labor costs for a professional eBay merchant. For the casual buyer, such a system will be helpful in determining whether to buy or not to buy an item, decreasing the time he or she will have to spend manually researching the market (or ameliorating the monetary cost of the absence of such research). We focus our attention on laptops because a) they can be evaluated by using well-structured data that lends itself well to machine learning techniques, b) they are representable as a set of easily quantified parameters, c) they represent a large market (about 4000/day), and d) their prices are predictable and relatively stable for short-term horizons.

There exist a large number of eBay automation tools, as well as a substantial body of research in price estimation, but the two fields appear to be largely separate. It seems that the eBay trading community does not realize the benefits that might be available to it, while computer scientists and economists are more interested in theoretical aspects of price prediction. We would like to think that our system may be the first that bridges the gap between the two.

Common eBay trading tools include *snipers* and simple *monitors* which are capable of alerting the buyer of all items that match some static search criteria; these tools do not attempt to decide whether an item is a good buy or not.

There are a number of studies analyzing factors that affect the final price of an online auction, including the effect of particular attributes on final price (Lucking-Reiley et al. 2007), the artificial domain Trading Agent Competition (Wellman et al. 2004), explicit price prediction for online auctions using machine learning on data extracted from item titles for very narrow domains (Ghani and Simmons 2004; Ghani 2005), and agent-based median price prediction (Gregg and Walczak 2004).

The work most related to ours improves upon these earlier approaches by mining product descriptions and using boosted machine learning to predict the final price (Heijst, Potharst, and Wezel 2008). It achieves much better generality than earlier approaches and is empirically applied to entire product categories (Canon digital cameras and Nike men shoes). While their approach is more general than ours (they do not have to explicitly specify category-specific features to be extracted), our application performs better on the category of our choice. Their Mean Relative Error (MRE) is 34% on Nike shoes and 58% on Canon cameras. For comparison, we use the same metric to measure the accuracy of our results and achieve an MRE of 16% for laptops.

Methods

The application consists of two primary components: a web-based module written in PHP, and an offline module written in C. The web component is responsible for communicating with both eBay and end users of the program, as well as for testing instances against a model in real time. The local component is responsible for learning the pricing model. A database stored on the web server is the single point of contact between the two components of the software.

System overview

Our application periodically gathers data for past laptop auctions and stores that data in a database. It periodically runs a training algorithm on that data and builds a price prediction model and stores this model in the database as well. For the casual buyer, a public web form is maintained that allows

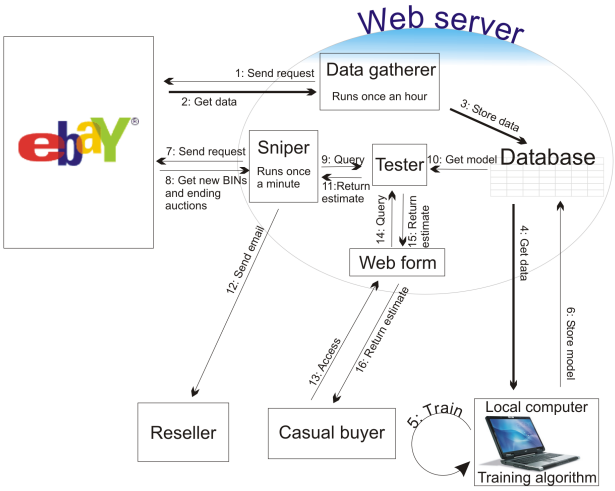

Figure 1: *Graphic overview of the entire system*. Numbers in the figure represent a rough temporal ordering of information exchange. The system consists of two primary components: a local one, responsible for learning, and a web-based one, that performs the communication with eBay.

anyone to enter desired laptop parameters which are evaluated against the stored model, resulting in a price estimate that is returned to the user. For resellers, the system periodically queries eBay for laptop auctions that are ending in the next minute, and for buy-it-now laptops that were listed in the last minute; evaluates each of these items against the stored model; for ending auctions, checks if current auction price is δ below the price estimate and if so, emails the reseller; for new buy-it-now items, checks if buy-it-now price is δ below the price estimate and if so, emails the reseller (δ is a profit margin set by the reseller). An overview of the system is presented in Figure 1.

Some of the web-based components run periodically using cron job scheduling. The local training module is run manually every week to update the model in accordance with market changes (this is likely more frequently than necessary). This periodic retraining assimilates large-scale market trends, such as a change in the market's valuation of a particular technology or a gradual increase in netbook sales.

Since laptop prices go down rapidly, for training we only use the latest $10,000$ collected instances (roughly corresponding to 50 days of data gathering). Our statistics show that laptop prices drop by 6% in this timeframe, an interesting correspondence with Moores Law since 6% in 50 days translates to a 50% price drop in 1.5 years.

Data acquisition and selection

Our data acquisition process is largely predetermined by eBay procedures and its APIs. Specifically, eBay allows searching for items through its APIs and gives an XML-based set of search results, but these results are not complete. To acquire all the required information, we need to send an individual get-item-by-id request for every laptop in which we are interested. Additionally, eBay limits the number of daily requests to 5000 per developer, unless the application is manually approved by eBay staff. Furthermore, eBay allows searching only for items that are being sold at the moment of the request, not for auctions that have already ended. However, if one requests an ended auction by item id, eBay will return the information, including the final selling price.

Data acquisition Data is accumulated using tools provided by eBay: search, marketplace research and Terapeak. Where applicable, these statistics assume a $\$200$ minimum price for all laptops (introduced to remove laptop accessories listed in the wrong eBay category). About 4000 new laptops are listed for sale every 24 hours. At any given moment the number of active laptop auctions is around $13,000$ in addition to approximately $14,000$ laptops listed at fixed prices. Of these items, we are only interested in laptops that sold on auctions, because fixed prices are preset by the sellers and do not necessarily reflect the real market price. Of 4000 laptops listed daily, 2500 are listed on auctions, and 78% of these sell. These approximately 2000 laptops are our potential training data.

From these, we cull a set of "good" examples represented by the following list of attributes: *brand*, *family*, *series*, *CPU type*, *multi-core configuration*, *CPU speed*, *RAM size*, *disk size*, *LCD size*, *operating system*, *optical drive type*, *condition*, *seller feedback count*, *seller feedback percentage*, *seller powerseller level* and *auction duration*. There is a mix of nominal and numerical attributes, and as part of our data selection process we impose restrictions on the ranges of some of the features. Most of these attributes are gathered from an eBay-defined set of parameters that is specific to the laptop category and is available through the API. In addition to these, we extract some information from the title and also record some auction- and seller-related data.

Data selection Our data selection process consists of three layers. The first layer is applied when we are searching for ending auctions on eBay. It primarily consists of the keywords we specify in the search query submitted to eBay: "** -(broken, parts, cracked, dead, damaged, as is, bad, no, not, lot, only, repair, repairs, fix, for)*". This query is applied only to the auction title and it allows us to remove most of the broken laptops from our dataset. Some additional restrictions are also specified in this step: the item must have at least one bid, the item price can not exceed $\$2500$, the item must not be listed as a lot, and the seller must accept PayPal as a payment method. Of the 2000 laptops that are our potential (daily) training set, this step leaves about 1500.

The second selectivity layer is applied when we retrieve individual data for every sold laptop using a get-item-by-id type of request. First, it imposes some of the range restrictions mentioned above: laptops must have CPU speed, RAM size, Hard disk size and LCD size parameters specified; shipping costs must be specified and not exceed $\$100$; seller must have a feedback count of at least 20, and a feedback percentage of at least 93%; buyer must have a feedback count of at least 10, and a feedback percentage of at least 80%. Second, the free-form laptop description is

searched for the word "warranty"; if it is not present (or present in the no warranty form), the description is then searched for words "bad", "not working", "broken", "damaged", "parts","repair", "p&r", "cracked", "dead", "no ac", "no power", "as is", "as-is", "mystery", "freezes", "no battery", "no video" and "missing". If any of these are present, the laptop is discarded. This final procedure removes all the broken laptops not filtered out in the first selectivity layer, while attempting to preserve good data by assuming that warranty information guarantees good laptop condition. Of the 1500 laptops remaining after the previous step, only about 450 are at this point actually written to the database; most of the laptops are removed because of missing attributes.

The final selectivity layer is applied right before the training algorithm is started (and, to some degree, during its execution). First, it imposes additional range conditions (for example, all laptops with more than M missing nominal values are discarded). Second, for the various nominal features it counts the number of unique values encountered. If a particular nominal value is encountered fewer than θ times in the dataset, the laptops that have this value are removed. This is done to ensure that rare, misspelled and incorrect values do not skew our results. Third, the training algorithm builds an initial prediction model for laptop prices and tests all of the dataset instances against this model. The instances which the model places far away from any other instances are removed from the dataset as outliers (any point that is a factor of η further from its nearest neighbor than the average distance between nearest neighbors). These instances are usually the result of scamming activity or severe discrepancies between laptop presentation in its attribute set and free-form description. This final selectivity layer leaves about 200 instances out of 450, and these represent our daily training data. The entire process is summed up in Figure 2.

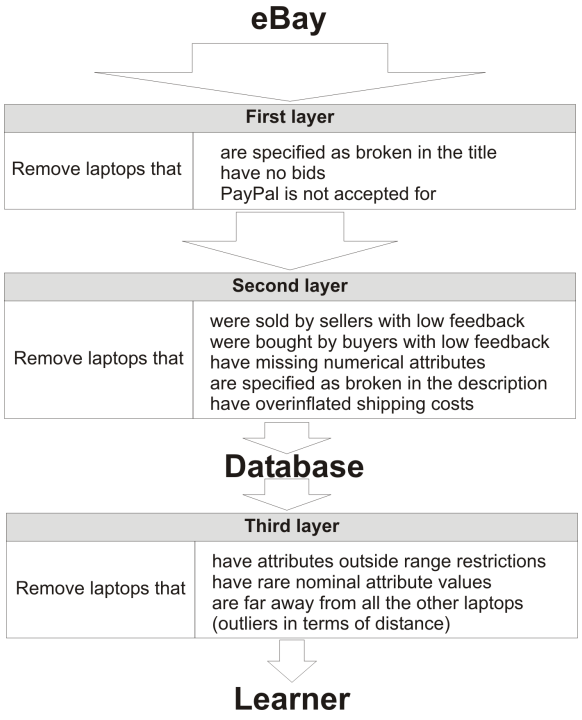

Figure 2: *Overview of data selection process*. Only about 10% of all sold laptops are eventually used as training data.

Mining the title While most of the features are supplied to us by eBay in a form ready to be stored, two features need to be extracted from the free-form item title: laptop *family* and laptop *series*. Laptop titles are free-form; however the length is limited to 55 characters. Typically sellers specify laptop brand, family, model and the best features of their laptop, possibly including common words like "laptop" or "notebook". Our goal is to extract the family, which is typically a sensible-sounding made-up word, and the model, which is a jumble of letters, digits, dashes etc. that follows the manufacturers own cryptic naming convention.

To facilitate this, we have compiled a table of every laptop model released by all major manufacturers. This database table has the following fields:

1. Brand: One of the eBay-predefined features, extracted from the attribute set of the laptop listing.
2. Family: Manufacturer-defined words such as "Latitude", or "Satellite Pro".
3. Series: A common name for all similar laptops, for example "D610".
4. Model: A regular expression representing laptops falling under the same series, for example "D610(-\ w{0,5})?"
5. Status: 1 for laptops added to this table manually; 0 for words added automatically
6. Count: For words added automatically, count of how often the word was encountered during data acquisition.

When we process the title, we look for regular expression matches against the "Model" field in the table, and if a match is found, the extracted model is the value of the "Series" field. However, if a matching regular expression is not found in the table, we perform the second step of the algorithm. First, we remove common words from the title, such as "notebook", "wireless", "centrino", etc. Second, we remove all common symbol combinations used to represent numerical features. Each word or symbol combination that remains in the title after this step is added to the laptop table, and its *status* field is assigned to be 0. If this word is already in the table, we increment its *count* field. We periodically manually inspect the instances that have the highest *count* and, if needed, either add these words as new models, or add these words to the blacklist of common words that are removed from the title. This approach allows us to detect the releases of new laptops by the manufacturers without monitoring the manufacturers themselves. This algorithm manages to extract the model from 83% of all the titles for laptops that manage to bypass the first two selectivity layers. We do not know how many titles actually have the model specified. The family is extracted for 91% of laptops not made by Gateway (which does not have families).

Model selection

We use feature-weighted k-Nearest Neighbor for the price prediction model, with the following design decisions: for continuous attributes, we use the (weighted) L_n-norm to compute distance; the distance between nominal attributes is set to 1 if they do not match, or to 0 if they do match; if a nominal attribute is missing, it does not match anything, and we do not allow missing numerical attributes.

For setting the weights, we use evolutionary computation to find the feature weights that minimize the price prediction error (see Algorithm 1). Since this means running k-NN against the dataset for each fitness evaluation, training time can be significant because the dataset is growing every hour, and because of k-NN's $O(n^2)$ complexity. This is ameliorated by the fact that weight optimization need be done only infrequently. To further ameliorate the time complexity of the search, we note that the set of nearest neighbors for every instance in the dataset changes infrequently. This allows us to cache the set of potential nearest neighbors for every instance, and during the algorithm execution look for nearest neighbors only in this cached set, rather than in the whole dataset. The size Γ of this cache is determined empirically. This approach decreases computation time, while having almost no effect on accuracy.

We sort our population by its predictive error and use the following formula to determine the probability s of a member of the population surviving to the next generation:

$$s = \left(1 - \frac{rank}{P}\right)^{\frac{1}{2r}}$$

where $rank$ is the index of this member of the population in an array sorted by MRE (increasing), P is population size, and r is a survival rate. The formula has a number of properties: the best member always survives, for $r = 0.5$ the survival chance of the middle member of the population is 0.5, and smaller r results in fewer survivors. The remainder of the population is filled by children, whose parents are chosen at random from the survivors with equal probability.

The mutation multiplier λ is introduced to increase the variability of the population. This number can take random values around 1, with the possible range of the values determined by a variability parameter ν. In our experiments we found that decreasing λ over time yields the best results—changing emphasis from exploration to exploitation, similar to the temperature parameter in Simulated Annealing algorithms. In general, λ is chosen randomly from the range $(1 - \nu^g, 1 + \nu^g)$, where ν is the variability parameter and g is the current generation.

Algorithm 1 Evolutionary computation for feature weighting, driven by search for weights that result in the lowest MRE on the training data. The type of crossover is selected randomly between three options.

Inputs:
Number of generations G, population size P, variability ν, survival rate r
Initialize:
Pop. Cur of weight vectors $w_{ij} = \texttt{rand}(0.001, 1.0)$, $1 < i < P$ and $1 < j < Numattr$
$mrate \leftarrow \nu$
for g = 1 to G **do**
  calculate fitness ϵ_i and rank $rank_i$ for each w_i
  for each w_i in Cur **do**
    $s_i \leftarrow \texttt{survive}(w_i, rank_i, P, r)$
    add w_i to new population New with probability s_i
  while $|New| < P$ **do**
    randomly select parent u from Cur
    randomly select from {C1, C2, C3}
    if C1 **then** {single parent mutation}
      for each child weight v_j **do**
        $\lambda \leftarrow 1 + \texttt{rand}(-mrate, mrate)$
        $v_j \leftarrow \lambda u_j$
    if C2 **then** {2-parent multi-point crossover}
      randomly select parent z from Cur
      for each child weight v_j **do**
        $\lambda \leftarrow 1 + \texttt{rand}(-mrate, mrate)$
        randomly choose $v_j \leftarrow \lambda u_j$ or $v_j \leftarrow \lambda z_j$
    if C3 **then** {2-parent average}
      randomly select parent z from Cur
      for each child weight v_j **do**
        $\lambda \leftarrow 1 + \texttt{rand}(-mrate, mrate)$
        $v_j \leftarrow \lambda(u_j + z_j)/2$
    include v in New
  $mrate \leftarrow \nu * mrate$
return member of New with highest fitness, w_{best}

Empirical Results

Our validation mechanism is two-fold: standard statistical measures to evaluate prediction accuracy, and an estimation of the effect on trading efficiency. For measuring prediction accuracy, we use the Mean Relative Error (MRE) metric:

$$\epsilon = \frac{1}{N} \sum_{i=1}^{N} \frac{|p_i - o_i|}{o_i}$$

where N is a number of instances in the test set, p_i is the ith predicted price and o_i is the ith observed price. This is a fairly standard metric, and in particular, Van Heijst *et al.* report MRE scores from 0.34 to 0.58 for their approach, depending on the product category (2008). Results we report are averages acquired using 10-fold cross validation on several months of (pre-processed) eBay trading data.

To evaluate trading efficiency, we measure trading time and profit both for unassisted and assisted (by our system) trading. For the unassisted case, we act as an eBay trader willing to dedicate themselves to manually checking eBay for new buy-it-now laptops for two hours every day. We actually perform the actions of a trader, buying the laptops and then reselling them, tracking time spent and profit. For the assisted case, we run the application in its "resellers" mode for a period of time and again actually perform the actions of an eBay trader, now accepting buying suggestions from our application and evaluating the efficacy of those buying decisions with a human trader. We measure the total time spent making those decisions. We then resell all the laptops we

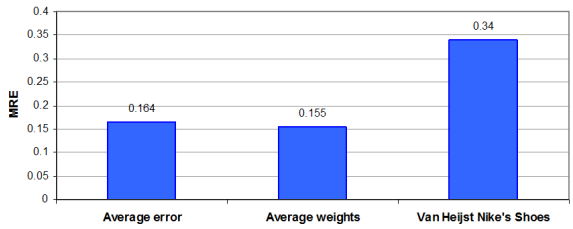

Figure 3: *Final MRE measurement*. The first column shows MRE using 20-fold cross-validation. The second column shows MRE over the full dataset for averaged normalized weights (used during trading efficiency validation). The third column is Van Heijst *et al*'s. (2008) reported result.

bought and calculate the total profit. We compare the profit made with and without using our system, as well as the total time spent on eBay. (A virtual comparison commonly adopted for stock trading evaluation is infeasible here because it requires all laptops virtually bought to be virtually re-sold; since most of the laptops bought on eBay are not re-sold, it is impossible to measure the "real" auction [re-]selling price, unless we are actually performing the sale.)

Parameter estimation

There are a large number of parameters that affect the performance of our system (empirical range in parenthesis): number of neighbors k $(1-19)$, the norm used for calculating nearest neighbors n $(0.1-3.0)$, max allowed missing values M $(1-6)$, rare values threshold θ $(0-6)$, neighbors cache size Γ $(5-200)$, outlier elimination factor η $(2.0-6.0)$, population size P $(0-700)$, survival rate r $(0.0-1.0)$, variability of weight mutation ν $(0.8-1.0)$, and number of generations G $(0-150)$. Our parameter tuning process consists of fixing all parameters but one, then sweeping over different values of this single parameter to determine its optimal value, using 10-fold cross validation. It can be argued that this process is flawed if various parameters are not independent, but empirical results suggest that it produces reasonably good parameter settings. The final values used in our evaluations are $k = 5$, $n = 1$, $M = 2$, $\theta = 0$, $\Gamma = 30$, $\eta = 4.8$, $P = 50$, $r = 0.3$, $\nu = 0.955$, $G = 80$.

Pricing prediction error

We performed all our parameter estimation on the same "offline" dataset of 5954 instances that passed the first two selectivity layers of our data filter. By the time the parameter estimation was completed the "online" dataset had grown to 10521 instances, which we used with 20-fold cross-validation to estimate ϵ, the prediction error. Note that all of the data that we use for training is highly sterilized, having passed rigorous selection procedures. This is true for both training and test folds (except that test subsets were not subjected to any selections that required knowledge of the model, for example outlier removal).

Figure 3 shows the estimated prediction error for laptop prices. The first column shows the average error on the test set over 20 folds of cross-validation. The second column shows the result of averaging the normalized best weights produced over 20 folds, and testing the entire dataset with these weights (these averaged weights are used for the trading efficiency evaluation). For comparison, the third column shows the results of Van Heijst et al. (2008) on the Nike shoes category (note the generality/performance trade-off).

Trading efficiency

While many measures of efficiency might be considered, here we focus on the intuitive and useful metric of profit/hour. The denominator only includes the amount of time taken by the reseller to make a buying decision. It does not include time for the following reseller tasks:

- going through the eBay/PayPal bureaucracy
- testing/repairing the laptop upon its arrival
- listing the laptop back on eBay
- packing the laptop
- shipping the laptop

These quantities are effectively constant for a single laptop and do not depend on whether the reseller uses our system or not. As for profit, eBay and PayPal fees, as well as shipping costs, are factored in. In addition to profit/hour we calculate a number of secondary metrics: total amount of profit made, and percentage return on investment. While we focus our validation on resellers, we also provide a web form that anyone can access for free: `http://ebay.xirax.net`.

For this experiment, we lift most of the restrictions on the tested instances—while we would want only perfectly working laptops for our training data, quite frequently the most profit is to be made on semi-broken laptops that can be repaired, or on laptops listed with incorrect or incomplete information. As a result, the main function of the human trader is a feasibility evaluation of recommended instances.

Before we can evaluate our system, we need to establish profit metrics for a reseller not using our system. For that benchmark we combine two sets of data:

1. For a period of two weeks we spent two hours per day on eBay looking for under-priced, buy-it-now laptops. Those found were resold. As such, we know both the profit we made, and the time we spent making it.
2. For a period of a couple of months, we bought and resold laptops without explicitly tracking our time spent online evaluating potential laptops. This data contains a much larger number of laptops bought and sold (increasing the reliability of the data) and if we assume that the time/purchase value is roughly equivalent between the two data sets, we can estimate the time for the second data set from the first one.

For comparison, assisted trading efficiency validation was performed in a fashion similar to the manual benchmarking. We activated the sniper module in our system to query eBay every minute for buy-it-now laptops listed in the last minute. The sniper passes laptop data to the tester, which evaluates it against the model. If the tester decides that the laptop is underpriced by at least 20% and $\$50$, the sniper sends an

Number	Profit	Profit/hr	Return	Time spent
10	$443	$15.82	10.7%	28 hours
41	$2052	$17.87	10.9%	115 hours
51	$2495	$17.45	10.8%	143 hours
19	$585	$98.04	10.3%	5.97 hours

Table 1: *Trading efficiency baseline and validation results*. The first row is data from the unassisted 2-week trading experiment (2 hours/day). The second row is data from an earlier (unassisted) trading cycle with estimated time spent. The third row is the sum of rows one and two. The fourth row is data from the assisted 2-week trading experiment. Our system provides dramatic profit/hour improvement and also a slight increase in the amount of overall profit.

email alert to a specified email address. Over the course of two weeks the system sent us alerts for 236 laptops, an overwhelming majority of which were broken beyond inexpensive repair, had errors in listings, or were not laptops at all. Based on these alerts, we purchased 19 laptops that we deemed to be actually underpriced. We measured the time it took us to evaluate all 236 alerts. Most were discarded or acted on in less than a minute; however a few required more extensive research, up to five minutes or so in a few cases. The results for both unassisted and assisted trading are summarized in Table 1, with the most notable being a 562% increase in trading efficiency when using our system.

Using the automatic system significantly decreases the time that needs to be spent on eBay looking for newly-listed laptops, and as a result the profit/hour measure increases dramatically. In addition, the system has a positive effect on the overall generated profit, since it queries eBay for the entire 24 hours every day, instead of the 2 hours/day spent for the manual benchmarking. There is, however, room for improvement. During the manual benchmarking, our two hour block of time spent on eBay covered about 20% of all the laptops offered for sale on a given day. Assuming that these 2 hours (7-9pm MST) are the busiest hours every day (which is close to reality), and assuming further that the human performance during the manual benchmarking was the best possible, the total potential profit using the automatic system is about 5 times what was made during the manual benchmarking, or approximately $2200 for the two-week period. The automatic system only achieves about one fourth of that value, due to two main factors: inconsistent availability of human trader and errors in the prediction estimate. While we did not count the number of missed alerts, we can give a rough ballpark estimate of about 25%. As for the second problem, we can combat it it by easing the automatic alert thresholds, with the tradeoff being more time spent evaluating the alerts.

Discussion

Our system facilitates significant improvements in trading efficiency on eBay by delegating the market research step to a computer; this turns what could be a full-time job into a non-intrusive activity that requires minutes, rather than hours, of human intervention. Additionally, our application directly boosts revenues, since unlike a human trader it is able to monitor eBay 24 hours/day.

For a casual one-time shopper, this application may also prove useful, since it provides a reasonably accurate price very quickly, is completely free, and is capable of storing and evaluating more data than the Terapeak tool provided by eBay Marketplace Research (currently the state-of-the-art).

The market in which our system is currently operating has a small capacity (estimated maximum net profit of $1100/week). However, there are many opportunities for expansion: auction items, additional computer categories (desktop, Macintosh), other eBay categories, and even other online auction sites.

Within the laptop category we are currently only dealing with buy-it-now items; however the functionality is in place to deal with auctions as well (though this will likely require additional human involvement as auction prices often experience severe volatility during the last minutes of an auction). Since the attribute sets are very similar, expanding to the desktop computers category and the Macintosh computers category should require only minor modifications, mostly to the model/family extraction routine. Expanding the approach to other eBay categories should also be straightforward—the feature set must be redesigned in accordance with category specifics, but everything else is already in place. Additionally, as the tool is expanded to multiple categories, it may become feasible to sell access to the tool instead of, or in addition to, directly using it.

Finally, it is interesting to consider fully automating the process, allowing the system to make purchases based solely on its evaluation without per-item sanction by a human.

References

Ghani, R., and Simmons, H. 2004. Predicting the End-Price of Online Auctions. In *Proceedings of International Workshop on Data Mining and Adaptive Modelling Methods for Economics and Management*.

Ghani, R. 2005. Price Prediction and Insurance for Online Auctions. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovering in Data Mining*, 411–418.

Gregg, D., and Walczak, S. 2004. Auction Advisor: An Agent-Based Online Auction Decision Support System. *Decision Support Systems* 41(2):449–471.

Heijst, D. V.; Potharst, R.; and Wezel, M. V. 2008. A Support System for Predicting Ebay End Prices. *Decision Support Systems* 44(4):970–982.

Lucking-Reiley, D.; Bryan, D.; Prasad, N.; and Reeves, D. 2007. Pennies from eBay: The Determinants of Price in Online Auctions. *The Journal of Industrial Economics* 55(2):223–233.

Wellman, M.; Reeves, D.; Lochner, K.; and Vorobeychik, Y. 2004. Price Prediction in a Trading Agent Competition. *Journal of Artificial Intelligence Research* 21:19–36.