2008-11-30

Real-Time Automatic Price Prediction for eBay Online Trading

Ilya Igorevitch Raykhel
Brigham Young University - Provo

Follow this and additional works at: https://scholarsarchive.byu.edu/etd

Part of the Computer Sciences Commons

BYU ScholarsArchive Citation
https://scholarsarchive.byu.edu/etd/1631

This Thesis is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in All Theses and Dissertations by an authorized administrator of BYU ScholarsArchive. For more information, please contact scholarsarchive@byu.edu, ellen_amatangelo@byu.edu.
REAL-TIME AUTOMATIC PRICE PREDICTION FOR EBAY ONLINE TRADING

by

Ilya Raykhel

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science
Brigham Young University
December 2008
BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

Ilya Raykhel

This thesis has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

___________________  ________________________________
Date  Dan Ventura, Chair

___________________  ________________________________
Date  Eric Ringger

___________________  ________________________________
Date  Scott Woodfield
As chair of the candidate’s graduate committee, I have read the thesis of Ilya Raykhel in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

Date
Dan Ventura
Chair, Graduate Committee

Accepted for the Department

Date
Kent Seamons
Graduate Coordinator

Accepted for the College

Date
Thomas W. Sederberg
Associate Dean, College of Physical and Mathematical Sciences
ABSTRACT

REAL-TIME AUTOMATIC PRICE PREDICTION FOR EBAY ONLINE TRADING

Ilya Raykhel
Department of Computer Science
Master of Science

While Machine Learning is one of the most popular research areas in Computer Science, there are still only a few deployed applications intended for use by the general public. We have developed an exemplary application that can be directly applied to eBay trading. Our system predicts how much an item would sell for on eBay based on that item's attributes. We ran our experiments on the eBay laptop category, with prior trades used as training data. The system implements a feature-weighted k-Nearest Neighbor algorithm, using genetic algorithms to determine feature weights. Our results demonstrate an average prediction error of 16%; we have also shown that this application greatly reduces the time a reseller would need to spend on trading activities, since the bulk of market research is now done automatically with the help of the learned model.
Table of Contents

1 Introduction .................................................................................................................. 1
   1.1 Use case: casual buyers ......................................................................................... 3
   1.2 Use case: resellers .............................................................................................. 3
   1.3 Proposed solution .................................................................................................. 4
   1.4 Thesis outline ........................................................................................................ 5

2 Related work .................................................................................................................. 6

3 Project description ......................................................................................................... 9
   3.1 Global overview ..................................................................................................... 9
   3.2 Data acquisition and selection ............................................................................ 12
      3.2.1 Data acquisition ........................................................................................... 16
      3.2.2 Data selection ............................................................................................... 17
      3.2.3 Mining the title ............................................................................................ 19
   3.3 Training algorithm .................................................................................................. 22

4 Results .......................................................................................................................... 27
   4.1 Preliminary measurements .................................................................................... 28
      4.1.1 Determining $k$ for $k$-Nearest Neighbor .................................................... 31
      4.1.2 Determining optimal $L_n$-norm .................................................................... 32
      4.1.3 Determining allowed missing values .............................................................. 33
      4.1.4 Determining rare values threshold ................................................................. 34
      4.1.5 Determining cache size ................................................................................... 35
      4.1.6 Determining outlier elimination factor ........................................................... 36
      4.1.7 Determining population size .......................................................................... 37
      4.1.8 Determining survival rate .............................................................................. 38
      4.1.9 Determining variability ................................................................................... 39
      4.1.10 Determining generations ............................................................................... 40
   4.2 Final MRE measurement ......................................................................................... 40
   4.3 Usability validation ................................................................................................. 42
      4.3.1 Manual benchmarking .................................................................................... 43
      4.3.2 Automatic usability validation ......................................................................... 44

5 Conclusion ..................................................................................................................... 46

Bibliography ....................................................................................................................... 48
List of Figures

Figure 1. Graphic overview of the entire system .................................................. 10
Figure 2. Training data for eBay item #290254696414 .............................................. 14
Figure 3. Data acquisition process ......................................................................... 16
Figure 4. Overview of data selection process ............................................................ 19
Figure 5. Laptop model extraction routine ................................................................. 22
Figure 6. Genetic algorithm pseudocode .................................................................. 26
Figure 7. Determining $k$ for $k$-Nearest Neighbor .................................................. 31
Figure 8. Determining optimal distance norm ............................................................ 32
Figure 9. Determining allowed missing values ......................................................... 33
Figure 10. Determining rare values threshold ............................................................ 34
Figure 11. Determining cache size .......................................................................... 35
Figure 12. Determining outlier elimination factor ..................................................... 36
Figure 13. Determining population size .................................................................... 37
Figure 14. Determining survival rate ........................................................................ 38
Figure 15. Determining variability ........................................................................... 39
Figure 16. Determining generations ......................................................................... 40
Figure 17. Final MRE measurement ......................................................................... 41
Figure 18. A screenshot of the public gateway to our application ............................... 43

List of Tables

Table 1. List of all used features ............................................................................... 15
Table 2. Usability validation baseline ........................................................................ 44
Table 3. Automatic usability validation results .......................................................... 45
Chapter 1

Introduction

EBay is the largest online marketplace in the world, where anyone can buy and sell anything within legal limits. The daily number of transactions on eBay measures in millions [1]. While most of these trades are "auctions" – the highest bidder gets the item – there is also a large number of fixed price trades, called "Buy It Now"s, or "BINs". There are also mixes of both: you can either win an item in auction, or buy it right away for a prescribed price.

Naturally, when looking to buy an item on eBay, one does some market research. Depending on the item sought and the buyer's persistence, this research may take many forms that differ in the amount of time spent and the research methodology. Imagine buying a good laptop on eBay. Especially if the buyer does not know exactly which model of laptop he or she wants. In that case the bulk of this research will be determining what is a "good" price for a laptop (if the buyer is not interested in minimizing price, he or she goes to a retail store or manufacturer’s website instead of eBay). This is usually done by comparing laptop offers from different sources. Internet-savvy eBay buyers may use the “Completed Listings” [2] feature to find past auctions for similar laptops that would clearly tell them the market price. For most people this process involves at least some guessing, not to mention a lot of time spent comparing various models, making bids, asking questions of sellers, etc. One of the goals of this thesis is to greatly speed up and simplify this step in market research that determines a "good" price for an item on eBay.

The other goal of this research addresses high-volume eBay resellers. Resellers seek under-priced items on eBay, buy them, and then sell them back either on eBay, or through other sources. The biggest difference between resellers and "casual" buyers is that resellers usually don't particularly care what kind of item they are buying as long as it is under-priced, while casual buyers take great care in selecting what they are buying. Additionally, resellers would have already done
most of the market research they need, and can usually determine quickly whether an item is worth buying or not. However, despite their experience, resellers still have to manually inspect eBay listings, monitor them, and be physically present at a computer every time they need to buy something. With high trading volumes and low profit margins, this manual involvement is very time-consuming, and can easily translate into a full-time job. Our research addresses this problem and provides automation in product selection. Resellers still make the final decision, but only on a very small subset of all products – those deemed potentially worthy of buying by our software.

If we can train a machine to make an intelligent evaluation of a product offered for sale and predict its final auction price, we can greatly reduce human involvement, ultimately resulting in higher trading throughput and reduced labor costs for a professional eBay merchant. For an average person 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 the monetary cost of the absence of such research).

We are focusing our research on laptops. The eBay laptop category [3] was used for this research for a number of reasons:

- Laptops can be evaluated by using well-structured data that lends itself well to machine learning techniques. They have such parameters as CPU Speed, RAM size etc. that are easily represented by objective numerical values. Furthermore, eBay structures most of the data by itself, greatly reducing the need to mine data from the item descriptions.
- The Laptop market is large: about 4000 auctions a day. This allows us to gather a significant amount of training data, or, as the case may be, a sufficient amount of high-quality data.
- Laptop prices are predictable and do not shift randomly.
- The author has extensive experience in the market.

The following are the primary drawbacks of using the laptop category:
• Unusually high quantity of scammers.
• Significant number of non-working or semi-working laptops, whose failures cannot be easily evaluated by an AI.

There are other drawbacks, not specific to laptops, but rather encompassing the entire eBay buying and selling process. We will go over all of the intricacies of available data in the “Data acquisition and selection” Section. Suffice it to say that we will have to require very high selectivity in which trades may be used as training data.

1.1 Use case: casual buyers

Currently, there are several ways for casual buyers to determine the "true" eBay laptop price (i.e. the price this laptop would sell for on average in an auction). Most of these methods deal with manually looking for a laptop with similar parameters. A savvy shopper may search for current or past eBay auctions for that or similar laptops and infer a price from there. There are a number of ways and places where this information can be found [2,4]. However, none of these methods guarantee a good price estimate, sometimes they all fail, and they are all time-consuming.

1.2 Use case: resellers

Resellers usually have a streamlined market research process. While it’s essentially the same as for a casual buyer, it is much faster, typically taking only a few minutes. They have a different problem – under-priced items are sold very quickly. This results in a requirement to be constantly present at the computer to quickly make decisions in real time.

There are automated tools that assist resellers in their duties. These tools may make automatic bids on behalf of a reseller at the last possible second; or they may constantly scan the marketplace for newly-offered items with a buy-it-now price. While these tools may help in the first case, it is of lesser significance, and the
profit is to be made on buy-it-now items, since an auction item is likely to sell for its “real” price (unless the seller makes multiple severe errors while listing it) while an under-priced buy-it-now item only requires that the seller underestimate the item's value. But the number of buy-it-now laptops offered on eBay is very large, and the majority of them are not under-priced. So if an early-warning tool alerts the reseller about every single new buy-it-now laptop, the reseller will quickly be flooded with useless alerts.

### 1.3 Proposed solution

This is our solution for casual buyers:
The prospective shopper goes to a website and enters all of the laptop’s attributes into a web form (or perhaps it can be a browser extension). Our machine learning algorithm evaluates this data as a test instance and predicts the price, together with an estimate of how certain it is. This greatly reduces time spent doing market research, works for any kind of laptop found in eBay's laptop category, and comes together with an evaluation of the prediction's precision, allowing the shopper to decide whether further research is needed.

This is our solution for high-volume resellers:
Contrary to prior automatic tools, our tool alerts the reseller only of new buy-it-now laptops that the algorithm considers worthy of buying. Thus, the vast majority of alerts is eliminated, and only a very few relevant alerts arrive to the reseller, allowing him or her to make a final buying decision. The reseller specifies the expected profit condition – if our system considers the expected profit to be higher than specified, it sends an alert.

To learn how to predict prices, we use a feature-weighted $k$-Nearest Neighbor algorithm, for a number of reasons:

- Ability to handle both numerical and nominal attributes.
- No need for retraining upon adding or removing new instances.
- Ease of model storage.
• Ease of implementation.

We describe the training algorithm in greater detail in the “Training algorithm” Chapter.

1.4 Thesis outline

• Chapter 1 introduces the problem, provides motivation for it, and offers an overview of our solution.
• Chapter 2 discusses the related work in price estimation research and gives examples of tools similar to the one implemented for this thesis.
• Chapter 3 describes the project in detail. In particular, it gives an overview of both web-based and local components of the system, presents our data selection and gathering algorithms, and goes into both the $k$-Nearest Neighbor and Genetic components of our training algorithm.
• Chapter 4 defines validation metrics for our research and presents the results.
• Finally, Chapter 5 presents the current usage statistics of the completed application and discusses the potential for expansion and future work.
Chapter 2

Related work

Our long-term goal is to create a direct and usable application of machine learning techniques to a common consumer process. There is a large number of eBay automation tools, and there is a substantial amount of research in price estimation, but the two are almost always 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 software may be the first that bridges the gap between the two.

Among eBay trading tools a very common item is a "sniper" – a piece of software that makes bids on behalf of the buyer at the last possible second before the closing of an auction. It has been empirically shown that such a strategy is beneficial to the buyer [5]. Examples of those tools are Auction Sentry [6], Auction Navigator [7], BayGenie [8] and dozens of others. While our application in its "resellers" use case is capable of alerting its users of potential last-second wins, its primary functionality is to monitor new buy-it-now items. There is a smaller number of tools capable of alerting buyers of new buy-it-now items; those include AuctionSleuth [9], SearchDigger [10], and others. As mentioned earlier, these are only capable of alerting the buyer of all items that match some static search criteria – these tools do not attempt to make any kind of decision whether an item is a good buy or not.

The majority of research in price prediction deals either with exploratory study of past trades from an economic standpoint, or with prediction of time-series, such as stocks, futures, or globally-traded commodities. As for the auctions and online marketplace domain, there are a number of studies analyzing the factors that affect the final price. One such study is descriptive research by Lucking-Riely et al. [11] of data on eBay trades of collectible 1-cent pennies. This work focuses on naming and quantifying the features that impact the final price of a penny, but the
authors do not attempt to predict the price; rather they give statistical significance measures to attributes of a particular penny, and attributes of its seller.

Some auction price prediction work was done implicitly as a part of the Trading Agent Competition [12] that took place in 2002. This competition required its participants to predict prices for airfares and hotels, and build intelligent tools that would bid on those, competing with each other. The dataset used for this competition was entirely artificial, and the domain was explicitly restricted to airfares and hotels. There are numerous quirks specific to this domain that may make it difficult to port to other kinds of products.

In recent years there has been some research in explicit price prediction for online auctions. Ghani et al. [13,14] explore a couple of machine learning techniques (regression, binary and discrete classification) and their application to predicting the final price of a particular item sold on eBay. They extract item-specific features (condition, memory etc.) from the auction’s title. That approach has a number of drawbacks: sellers may not list them in the title or list them in a format that the extraction routine would not parse. Additionally, they gather data and test their approach on a single item type (the Palm Zire 21 PDA). Their results are good – 96% accuracy on hitting a correct $5 interval for an (on average) $55 item – but again, just for one item.

In "Auction Advisor" [15] Gregg & Walczak make use of an agent-based framework to build an application that assists auction traders in both buying and selling. As part of this application they attempt to predict the market price for items offered for sale by collecting past trades data from multiple sources: eBay, Amazon, and Yahoo. They find the median final selling price for all trades from all these sources for a specified item. This simple approach might work if there are many trades for a particular item, but it suffers from a number of drawbacks:

- If an item is rare, there are few or no trades, and using a median would produce incorrect results.
- If the item’s price changes over time, past trade data may not be applicable (they handled that by using only the data from the past 2 weeks).
If different items are sold under the same name (frequently the case with laptops, which may have wildly different components under the same name), using a median of their prices may not be sufficient.

Van Heijst et al. [16] improves upon these two approaches by mining product descriptions within auctions and using boosting-based machine learning algorithms to predict the final price. As a result, they achieve much better generality than either of the two approaches above. They run their experiments on entire product categories (Canon digital cameras and Nike’s men shoes). While their approach is more general than what this work accomplishes (since they do not have to explicitly specify category-specific features to be extracted), our application tends to perform better on the one category of our choice. Their Mean Relative Error was 34% on Nike’s shoes and 58% on Canon cameras (where Mean Relative Error is defined as how much, percent-wise, the prediction is off the real price on average). We use the same metric to measure the accuracy of our results, and in Chapter 4 we define the metric precisely and compare our results to theirs.
Chapter 3

Project description

Our intention is to develop a software product that will assist computer users who are not necessarily familiar with machine learning or how machine learning techniques can be applied to their trading activities. This application consists of two primary components: a web-based part written in PHP, and an offline part 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 training the model using a machine learning algorithm of our choice. A database stored on the web server is the single point of contact between the two components of the software. A more detailed description follows.

3.1 Global overview

Our application performs the following duties:

1. Periodically, once every hour, gathers data for past laptop auctions from eBay that have ended in the last hour.
2. Puts that data in the database.
3. Periodically runs a training algorithm on that data and builds a price prediction model.
4. Stores this model in the database.

For casual buyers use case:

1. Presents a public web form that allows anyone to enter desired laptop parameters.
2. These parameters are evaluated as a test case against the stored model, and a price estimate is presented to the user.
For resellers use case:

1. Periodically (once every minute) 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.
2. Evaluates each one of these items as a test instance against the stored model.
3. For ending auctions, checks if current auction price is $\delta$ below the price estimate, where $\delta$ is a value specified by the reseller. If it is, the application emails the reseller.
4. For new buy-it-now items, checks if buy-it-now price is $\delta$ below the price estimate, where $\delta$ is a value specified by the reseller. If it is, emails the reseller.

An overview of the entire system is presented in Fig. 1.

![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.](image-url)
Some of the web-based components run periodically using cron\(^1\) job scheduling. The local training module is run manually every week to update the model in accordance with market changes. One week interval was chosen in an *ad-hoc* manner, and the update likely happens more frequently than it is needed, since relative weighting only changes when buyers start valuing a particular feature differently – for instance, when high-definition optical drives were introduced, or when quad-core laptops become widespread. In essence, the retraining should only be necessary when a new technology becomes available on the market. Also, periodic retraining would take care of large-scale market trends, like a gradual increase in netbook sales. The following is the temporally-ordered list of information exchange in our system:

1. Hourly data gatherer sends a request to eBay using eBay trading API asking for auctions that ended in the past hour.
2. eBay returns past auctions data, which we use as the training data.
3. Data gatherer stores this information in the database.
4. Training algorithm (ran on a local computer) requests training data from the database. Since laptop prices go down rapidly, we only use the latest 10000 collected instances (roughly corresponds to 50 days of data gathering). Our statistics show that laptop prices drop by 6% in this timeframe (remarkable correspondence with Moore’s Law: 6% in 50 days translates to 50% price drop in 1.5 years).
5. Training algorithm learns patterns in the data and builds a prediction model.
6. This model is sent back to the web server and stored in the database.

Selection of training algorithm and model class determines the complexity of model serialization and storage scheme.

Resellers use case:

---

\(^1\) Cron is a unix scheduling program that allows users to execute scripts automatically at the specified time.
7. The "Sniper" module queries eBay for ending auctions and new buy-it-now items.
8. EBay returns data for ending auctions and new buy-it-now items.
9. This data is used as test instances in the "Tester" module
10. "Tester" retrieves prediction model from the database
11. "Tester" evaluates data gathered by "Sniper" against the model and provides an estimate of the real market price back to the "Sniper".
12. "Sniper" compares this estimate with the current price (for an auction item) or with the buy-it-now price, and if it finds the real price to be higher by $\delta$, it sends an automated email alert to the reseller.

Casual buyers use case:
13. Buyer goes to a public URL and enters laptop parameters he or she is interested in.
14. Web form forwards those parameters as a test instance to the "Tester" module.
15. "Tester" evaluates this data and provides a price estimate back to web form.
16. Price estimate is displayed to the buyer.

3.2 Data acquisition and selection

The data was accumulated using eBay search, the eBay market research feature [17] and the Terapeak eBay research tool [4]. Where applicable, these statistics assume a $200 minimum price for all laptops – it is introduced to remove laptop accessories listed in the wrong eBay category. Without that threshold these accessories will skew the statistics, since there is a very large number of them – more than the laptops themselves.

As noted earlier, the daily amount of trading in the laptop category is quite large. 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 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 necessary reflect the real market price. EBay marketplace research [17] confirms this assumption: the average price of a brand new laptop sold on auction is about $500, while the average price of a brand new laptop sold at fixed price is roughly $625. It may seem that listing at a fixed price should always be preferred, but this is not the case: while fixed-priced laptops sell for more, they sell much less often. Terapeak report [4] shows that roughly 78% of laptops listed on auctions sell (i.e. receive at least one bid), while this number is only 31% for buy-it-now laptops and 22% for store inventory items.

Of 4000 laptops listed daily, 2500 are listed on auctions. This is our potential training data. However, as mentioned earlier, only 78% of them sell, and those that don’t sell do not interest us. This leaves about 2000.

We may choose to store all of these 2000 laptops, and use all of them in the training algorithm, hoping that the amount of “good” data overwhelms the amount of “bad” data during the training process, and the training results will reflect that. Alternatively, we may be highly selective in which of these 2000 laptops to store. In this case we will have little to no “bad” data, but much less “good” data as well, unless we find a way to separate the two perfectly, which is unlikely. Ultimately, the choice between these two approaches is defined by our learning algorithm (explained in detail in Section 3.3). Since our learning algorithm is somewhat susceptible to outliers, and takes a lot of time to train, we have decided to employ the high selectivity approach. Van Heijst et al. in their work [16] also perform some selection on the data, although to a lesser extent: they set lower and upper bounds on item’s price. An overview of data preprocessing strategies can be found in [18].
Figure 2. Training data for eBay item #290254696414. We will extract everything from the attribute array on the bottom, “Latitude” family and “D610” model number from the title on the top; some additional features like auction duration and seller’s feedback rating are not shown.

Figure 2 presents parts of a screenshot of an eBay item #290254696414, which demonstrates the training data we use. Most of it is gathered from an eBay-defined set of parameters that is specific to the laptop category and can be seen in the rectangular box. In addition to these, we extract some information from the title and also record some auction- and seller-related data. Here is the full list of features we use:
<table>
<thead>
<tr>
<th>Feature name</th>
<th>Type of feature</th>
<th>Allowed range</th>
<th>Feature extracted from</th>
</tr>
</thead>
<tbody>
<tr>
<td>Laptop brand</td>
<td>Nominal</td>
<td>Any</td>
<td>Attribute set</td>
</tr>
<tr>
<td>Laptop family</td>
<td>Nominal</td>
<td>From a predefined set, or missing</td>
<td>Title</td>
</tr>
<tr>
<td>Laptop series</td>
<td>Nominal</td>
<td>From a predefined set, or missing</td>
<td>Title</td>
</tr>
<tr>
<td>CPU type</td>
<td>Nominal</td>
<td>Any</td>
<td>Attribute set</td>
</tr>
<tr>
<td>CPU multi-core</td>
<td>Nominal</td>
<td>Any</td>
<td>Attribute set</td>
</tr>
<tr>
<td>configuration</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPU speed, MHZ</td>
<td>Numerical</td>
<td>0-4096, not missing</td>
<td>Attribute set</td>
</tr>
<tr>
<td>RAM size, MB</td>
<td>Numerical</td>
<td>0-8192, not missing</td>
<td>Attribute set</td>
</tr>
<tr>
<td>Hard disk size, GB</td>
<td>Numerical</td>
<td>0-2048, not missing</td>
<td>Attribute set</td>
</tr>
<tr>
<td>LCD size, inches</td>
<td>Numerical</td>
<td>4-22, not missing</td>
<td>Attribute set</td>
</tr>
<tr>
<td>Operating System</td>
<td>Nominal</td>
<td>Any</td>
<td>Attribute set</td>
</tr>
<tr>
<td>Optical drive type</td>
<td>Nominal</td>
<td>Any</td>
<td>Attribute set</td>
</tr>
<tr>
<td>Item Condition</td>
<td>Nominal</td>
<td>Any</td>
<td>Attribute set</td>
</tr>
<tr>
<td>Seller feedback rating</td>
<td>Numerical</td>
<td>20-200000</td>
<td>Seller data</td>
</tr>
<tr>
<td>Seller feedback</td>
<td>Numerical</td>
<td>93-100</td>
<td>Seller data</td>
</tr>
<tr>
<td>percentage</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Seller powerseller</td>
<td>Nominal</td>
<td>Any</td>
<td>Seller data</td>
</tr>
<tr>
<td>level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Auction duration,</td>
<td>Numerical</td>
<td>1-10</td>
<td>Auction data</td>
</tr>
<tr>
<td>days</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1 presents all the features we use for training. Notice that there is a mix of nominal and numerical attributes, and also that some of the features have restrictions on their range. These restrictions are one part of our data selection.
process, and, for example, they will disallow the item from Fig. 2 being used for training, since it is missing its *Hard disk size* feature.

### 3.2.1 Data acquisition

Our data acquisition process is largely predetermined by eBay procedures and its APIs [19]. Specifically, eBay allows searching for items through its APIs and gives an XML-based set of search results, but these results are not complete: they have some of the data for the items, but not all of it. To acquire all the required information, we need to send an individual get-item-by-id request for every laptop we are interested in. Additionally, eBay limits the number of daily requests to 5000 per developer, unless the application is manually approved by eBay staff. Furthermore, eBay only allows searching items that are being sold at the moment of the request, not the auctions that have already ended. However, if you request an ended auction by an item id, eBay will return all the necessary information, including the final selling price. All these considerations result in the following data acquisition routine, which is run hourly as a cron job:

1. Foreach ItemID in ending_items list do:
   - Request detailed information from eBay for this ItemID.
   - Parse incoming XML for the item.
   - If all second-layer data selectivity conditions are satisfied, add item data to the database.

2. Query eBay for a list of items in the laptop category that satisfy all first-layer selectivity conditions.
   - Parse incoming XML for the list of items
   - Put ItemIDs into ending_items list and store it to be used in an hour in step 1

Figure 3. Data acquisition process, mostly determined by eBay APIs. During first hour, ask eBay for list of ending laptops. The next hour, query for the same laptops and get their final price.
3.2.2 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}”. The “for” token was included because it almost always comes in conjunction with “for parts”, or in “<component> for <laptop model>” context: search for “for” produces 1100 results, while search for “for –parts –repair” produces 158 results, most of which are laptop components. This query is only applied to the auction title, not the item descriptions; it allows us to remove most of the broken laptops from our dataset, however it does not remove all of them: for some of the laptops their disrepair is only mentioned in descriptions. We cannot apply the query to the description, because even if we remove such common words as “for”, “no” and “not” from the query, many laptop descriptions indicate that the item is “not damaged”, or “if dead on arrival we will accept returns”. Also, this query does remove some “good” data, specifically laptops that say “no reserve” or “for students” are the most common cases of good data being removed. However, for an overall majority of laptops, the presence of these words in the title indicates some kind of a problem. Some additional restrictions are also specified in this step: item must have at least one bid, item price should not exceed $2500, item must not be listed as a lot, and the seller must accept PayPal as a payment method. Previously we indicated that at this moment we have 2000 laptops as our potential daily training set. This step leaves about 1500 of them.

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 in Table 1: 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%. These parameter limits were chosen
based primarily on personal experience. Additionally, the free-form laptop
description is first 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 those are present, the laptop is removed. 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. In particular, all the mis-
categorized laptop accessories are removed in this step since they never have all the
numerical parameters specified.

The final selectivity layer is applied right before the training algorithm is
started (and, to some degree, during its execution). First of all, it imposes all the
range conditions specified in Table 1 (we will go over special Family and Series
features conditions shortly). Secondly, for the various nominal features it counts the
number of unique values encountered. If a particular nominal value is encountered
fewer than a predefined number of 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. Then the training algorithm builds an initial prediction
model for laptop prices and tests all of the dataset instances against this model. The
instances which are far away from any other instances are removed from the
dataset as outliers. 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
that is our final daily training data.

The entire process is summed up in Fig.4
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. The justification for even adding these features and for the considerable amount of work associated with retrieving them is the fact that the best way for a human buyer or seller to find a laptop similar to a given one is to type its model name or its series in eBay's search string. Only if the laptop of the same model cannot be found would a human actually start looking for similar laptops.
based on their internal hardware. So we surmise that this feature should have a considerable degree of importance in price prediction.

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” (see Fig. 2). 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 manufacturer’s own cryptic naming convention.

Our extraction algorithm consists of two components. First, we have compiled a table of every single laptop model released by all major manufacturers. This database table has following fields:

1. Brand: One of the eBay-predefined features, extracted from the attribute set of the laptop listing.
2. Family: Defined by manufacturers, such as “Latitude”, “Satellite Pro” or “Compaq” (which may be either a family or a brand).
3. Series: A common name for all similar laptops, for example “D610”.
4. Model: A regular expression representing all 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 (described later)
6. Count: For words added to the table automatically, count of how many times a particular word was encountered during data acquisition (described later).

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).

A more detailed algorithmic specification follows:

```
Extract Brand feature from the attribute set
Query the laptop table for all families in this Brand
Foreach returned Family
    Check if the Family is present in the title
    If Family is found in the title
        Query the laptop table for all Models, Series in this Family
        Assign the Family feature
    Else
        Query the laptop table for all Models, Series in this Brand
        Foreach returned Model regexp
            Check if Model regexp is present in the title
        If Model regexp is found in the title
            Assign Series feature
        Else
            If Family is known //family is specified, but maybe incorrectly
                Query the laptop table for all Models, Series in this Brand
                Foreach returned Model regexp
                    Check if Model regexp is present in the title
                If Model regexp is found in the title
                    Assign Series feature
                    Unassign Family feature
            If Family is not known, but the Series is known
                Query the laptop table for the Family matching this Series, Brand
                Assign the Family feature
```
If Series is not known
Remove all non-alphanumeric symbols from the title, keep “-” and “.”
Remove all combinations like “2.4 GHZ” with the regular expression:
\s\d{0,4}(\d{0,2})? ?(mhz|ghz|gb|gig|mb|"|in|inch|inches)(\s|$)
Remove all blacklisted common words from the title
Foreach remaining word
    If it’s not Brand and not one of the families for that Brand
        Query the laptop table for that word
        Increase its count in the laptop table

Once a month:
Order the laptop table by descending count and manually inspect top rows
If the row corresponds to an actual laptop model, switch its status to 1
Else, add the word to blacklist and remove the row from the table

Figure 5. Laptop model extraction routine. EBay traders typically estimate market prices using the prices of sold laptops of the same model, so this model feature is likely quite important for training. Laptop model is successfully extracted from 83% of all auction titles.

3.3 Training algorithm

The training data gathered through the process described in Section 3.2 has a number of properties that informs the choice of the learning algorithm. These properties include:

1. A mix of numerical/continuous and nominal discrete attributes
2. Possible missing data.
3. Numerical data is not normalized and has different ranges.
4. Nominal attributes are unordered, and the distance between different values for nominal attributes is not defined. For example, we cannot explicitly define a distance between "IBM" and "Toshiba" brands.
5. Possible incorrect data. The seller may misspell, or enter a wrong number.
6. New data is added to the dataset every hour.

To a large extent, our algorithm is determined by these traits; however, there are other items to consider. Firstly, the project’s structure and purpose demand that the testing routine should be algorithmically fast, since it is implemented in PHP, and a rapid response is required for the reseller to be able to buy the laptop before
anyone else. This also implies a need for a relatively simple model storage scheme, so that it can be retrieved from the database quickly. Secondly, we believe that the data selection process will have a much larger effect on the algorithm’s accuracy than will the choice of the algorithm itself, so we have dedicated a much larger amount of time to that process than to tuning our algorithm to perfection.

We have used a feature-weighted $k$-Nearest Neighbor algorithm [20] for this project. This algorithm aligns well with our data: it has nice facilities to deal with mixed or missing data, normalization is not required (due to weights assigned to features), and it does not require frequent retraining as new instances are added to the dataset, since the algorithm is largely instance-based, and instances themselves comprise the model; the weights also do not change frequently, as mentioned in section 3.1. As with any Machine Learning algorithm a number of decisions need to be made on the developer’s part as to the specifics of its implementation:

- 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. We do not allow missing numerical attributes (see table 1).
- The algorithm is distance-weighted; the relative importance of a particular neighbor is proportional to its distance from the tested instance.
- We use $L_1$ distance norm to speed up calculations. While we saw slightly better results for smaller values of the norm, the training time increases significantly.
- We experimented with different values of $k$, and found the results to be the best for $k$ in the range of 5 to 8. We use $k = 5$ to speed up computation.

The standard $k$-NN algorithm weights all the features equally, suffering from degraded performance if features have different degrees of importance. We believe that some attributes are more important than others (such as CPU speed versus auction duration, for example), so we weight attributes differently. There are a number of approaches to do this, which are summarized in Wettschereck and Aha’s work “Weighting features” [21]. One approach uses a gradient descent technique to
iteratively converge to optimal weights \cite{22}; another assigns weights based on single-feature performance \cite{23}. We have used evolutionary-based search \cite{24} to find the feature weights that maximize the accuracy.

Genetic search is an iterative feedback-based method: the result of a previous iteration of \( k \)-NN testing serves to modify some parameters for the next iteration. This means that this algorithm runs \( k \)-NN against the dataset over and over again, modifying the weight assignments on every iteration. As a result, the training time is quite long, because the dataset is growing every hour, and because of \( k \)-NN's \( O(n^2) \) complexity. This promotes the need for some optimization. While the weights change significantly during the search, 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 before we initialize the algorithm, and during the algorithm execution only look for nearest neighbors in this cached set, not in the whole dataset. This approach greatly speeds up algorithmic execution, while having almost no effect on accuracy (see Results Chapter for more details).

Genetic algorithms consist of randomly generating a sizeable “population” and then “evolving” it in accordance with a “survival of the fittest” principle. In our case, the population is a set of weight vectors, and the fittest vector is the one that gives the best accuracy on the training data. During every generation, all members of the population are evaluated against the fitness criterion, and the more fit ones then produce “offspring”, i.e. members of the population for the next generation. The “child” is usually some kind of perturbation of its “parents”. Additionally, parents themselves may survive to the next generation if they are well fit; also, random mutations are sometimes introduced to better simulate biological evolution and improve the variability of the offspring. As with \( k \)-NN, a number of decisions need to be made regarding the specifics of the implementation:

- In genetic algorithms children are usually produced via some sort of permutation that attempts to simulate a real-life DNA crossover mechanism. A short survey of variations of crossovers and their performance related to \( k \)-NN can be found in \cite{25}. For every child we randomly choose one of three crossover mechanisms: child weights may be set to equal a single parent’s
weights, multiplied by a random “mutation factor”; child weights may be randomly picked to be either of the two parents’ weights, again multiplied by a mutation factor; or child weights can be set to be the average of the parents’ weights, multiplied by a mutation factor.

- Genetic algorithms employ a probabilistic approach to determine which members of the previous generation are going to produce offspring: as the fitness gets worse, the mating probability decreases. We sort our population by its MRE (see Chapter 4), and use the following formula to determine if a particular member of the population survives to the next generation:

\[
P = \left(1 - \frac{\text{Rank}}{\text{Size}}\right)^{\frac{1}{\text{Size}}} S
\]

Where \( P \) is a probability of survival, \( \text{Rank} \) is the index of this member of the population in an array sorted by MRE (increasing), \( \text{Size} \) is population size, \( S \) is a survival rate, which we determine by experiment. The formula has a number of properties: the best member always survives, for an \( S \) of 0.5 the survival chance of the middle member of the population is 0.5, smaller \( S \) results in fewer survivors. The remaining spots in the population are filled with children, whose parents are chosen at random from the survivors with equal probability.

- The “mutation multiplier” is introduced to increase the variability of the population. This number can take random values around 1, the possible range of the values is determined by a “variability” parameter, and is in the range \((0, 2)\). In our experiments we found that decreasing the mutation multiplier over time yields the best results – at first we are looking for a broad good fit, but as time goes on, we are trying to nail down the absolute best possible set of weights. This is comparable to the “temperature” parameter in Simulated Annealing algorithms. In general, the mutation multiplier is chosen randomly from the range \((1-v^i; 1+v^i)\), where \( v \) is the variability parameter and \( i \) is the current generation.
The genetic algorithm is summarized in the following pseudocode:

```
Initialize population array CURRENT of weight sets, setting weights to random values between 0.001 and 1.
Set MUTATION_RATE to predefined variability parameter
For a predefined number of generations
  Foreach member M of CURRENT
    Test performance of M on the training set
  Order CURRENT by decreasing performance
  Foreach member M of CURRENT
    Calculate survival probability P based on M’s index in CURRENT; Append M to NEXT with probability P
  Foreach remaining spot in NEXT
    Randomly select F from PARENTS, let F’s weights be W_fi
    Initialize child C
    Select between CR1, CR2, CR3 with equal probability:
    CR1 (single parent mutation):
      Foreach weight W_c1 of C
        Set MUTATION_MULT = Random value in [-MUTATION_RATE; MUTATION_RATE]
        Set W_c1 = W_fi * MUTATION_MULT
    CR2 (2-parent multi-point crossover):
      Randomly select M from PARENTS, with weights W_mi
      Foreach weight W_c1 of C
        Set MUTATION_MULT = Random value in [-MUTATION_RATE; MUTATION_RATE]
        Randomly set W_c1 to either W_fi or W_m1
        Multiply W_c1 = W_c1 * MUTATION_MULT
    CR3 (2-parent average):
      Randomly select M from PARENTS, with weights W_m1
      Foreach weight W_c1 of C
        Set MUTATION_MULT = Random value in [-MUTATION_RATE; MUTATION_RATE]
        Set W_c1 = (W_fi + W_m1) / 2 * MUTATION_MULT
    Set MUTATION_RATE = MUTATION_RATE * variability
```

Figure 6. Genetic algorithm pseudocode. The genetic algorithm is used to iteratively search for weights that result in the lowest MRE on the training data. The type of crossover is selected randomly between three options.
Chapter 4

Results

Our intention was to build a fully-functional, usable software product, not just to explore a theoretical new application of machine learning methods. If we wish to know how useful it can be, not only do we need to evaluate the accuracy of our algorithms but also how much profit can be earned with their help. Our validation mechanism needs to be two-fold: it should consist of standard statistical measures to evaluate prediction accuracy, and it should also gauge the profits that could be made with the application’s assistance.

We will use the Mean Relative Error (MRE) metric to measure our prediction accuracy:

\[
MRE = \frac{1}{N} \sum_{i=1}^{N} \frac{|P_i - O_i|}{O_i}
\]

Where \( N \) is a number of instances in the test set, \( P_i \) is the \( i \)th predicted price and \( O_i \) is the \( i \)th observed price. The metric measures the percentage of how much, on average, our prediction is off the actual price. This is a fairly standard metric used in a number of works; in particular, Van Heijst et al. report MRE scores from 0.34 to 0.58 for their approach, depending on the product category they use it on [16].

We first gather a large amount of data — a few thousand instances. Then we perform 10-fold cross validation testing on this dataset. This involves partitioning the dataset into training and test subsets, repeated 10 times, each time using a different portion of the original set as a test subset. We find the best \( k \)-NN weights for each of the training subsets, and then run the corresponding test subset against the models. We measure the MRE of each of the 10 tests, and report the average as our result.

Usability testing of the second component is somewhat more complicated. We run the application in its "resellers" mode for a period of time and actually perform the actions of an eBay trader; we accept buying suggestions from our application and make actual buying decisions. We measure the total time spent making those decisions. We then resell all the laptops we bought and calculate the total profit. Then, we act as an eBay trader that does not have the benefit of such an application, but is willing to
dedicate themselves to manually checking eBay for new buy-it-now laptops for two hours every day. Again, we actually perform the actions of a trader, buying the laptops and then reselling them. We compare the profit made with and without using our system, as well as the total time spent on eBay. Unfortunately it is impossible to compare performance on the same dataset, but we believe that a relatively long testing period, as well as a large amount of trading in the laptop category offsets random factors and makes our comparison meaningful. A “virtual” comparison commonly adopted for stock trading evaluation is infeasible for our problem, since that requires all laptops virtually bought by us to be virtually sold back, but most of the laptops bought on eBay are not sold back, so it is impossible to measure the “real” auction selling price, unless we are actually performing the sale.

The following is an outline of the remainder of this chapter:

- In 4.1, we perform preliminary measurements to establish various biasing parameters for our algorithms, such as $k$ for $k$-NN, the outlier elimination criterion for data selection, variability for the genetic algorithm, etc.
- In 4.2 we evaluate our final MRE result.
- In 4.3 we execute the usability validation and measure the profit/hour improvement that our application brings to the resellers.

### 4.1 Preliminary measurements

There are a large number of parameters that affect the performance of our algorithms. In this section we try to determine the combination of these parameters that yields accurate results, while not being too time-consuming. More specifically, in determining these parameters we look for three broad goals that are somewhat mutually-exclusive:

1. The algorithms need to have high accuracy.
2. The algorithms need to be as general as possible, and should work on as many laptops as possible.
3. The training process should take a reasonable amount of time.
While we are somewhat flexible as to the numerical representation of these requirements, all three need to be satisfied.

Our parameter tuning process consists of fixing all parameters but one, then measuring the performance for different values of this single parameter, and determining the best value for it. We do a 10-fold cross validation on the dataset to find each parameter; with a 10% test data and 90% training data split. Afterwards, we proceed to perform 20-fold cross-validation with a combination of the best values for all parameters. It can be argued that this process is flawed if various parameters are not completely independent, but we feel it gives a reasonable enough approximation, and building a second-level meta-learning system to determine the best parameter combination is well beyond the scope of this thesis.

The following is a list of parameters we determine experimentally:

**K**: The number of neighbors for \(k\)-NN to take into account.

**L\_n-norm**: \(k\)-NN distance norm used for calculation. \(L\_n\)-norm of 1 means “Manhattan distance”, 2 is a standard Euclidian distance, and in general for two points \(A, B\) with coordinates \((A_1, A_2, ..., A_F)\) and \((B_1, B_2, ..., B_F)\), and \(L\_n\)-norm \(n\),

\[
\text{distance}(A, B) = \left( \sum_{i=1}^{F} \left( |A_i - B_i| \right)^n \right)^{\frac{1}{n}}
\]

We explore values of \(n\) from 0.2 to 3.0.

**Allowed missing values**: the maximum number of missing nominal values for an instance to remain in the dataset.

**Rare values threshold**: as mentioned in Section 3.2.2, if a certain nominal value is encountered in the dataset fewer than this number of times, all instances containing this value will be eliminated from the dataset.

**Cache size**: number of potential neighbors to store for every instance in the dataset, so that for later iterations only those neighbors will be considered as possibly nearest. Using a cache greatly speeds up the genetic algorithm but has a small negative effect on the accuracy.

**Outlier elimination factor**: again mentioned in Section 3.2.2, we first build an approximate model, and then use this model to remove the outliers before
constructing the final model. We measure an average distance from an instance to its nearest neighbor, and multiply this distance by the outlier elimination factor. If for any instance in the dataset, its nearest neighbor is farther than the resulting number, this instance is removed.

**Population size:** for the genetic algorithm, the number of weight sets in a population.

**Survival rate:** for the genetic algorithm, the factor that determines the chance of survival of a particular member of the previous generation. See Section 3.3 for a complete description.

**Variability:** for the genetic algorithm, the boundaries on the number of random mutations during every generation. For example, variability of 0.6 means that mutation rate can be a random value in the range (1-0.6, 1+0.6), which in turn means that for the first type of crossover (See Section 3.3), children weights are set to equal parent’s weights, multiplied by a random number from (0.4, 1.6).

**Generations:** for the genetic algorithm, the number of generations before completion.
4.1.1 Determining $k$ for $k$-Nearest Neighbor

<table>
<thead>
<tr>
<th>$k$</th>
<th>$L_n$ norm</th>
<th>Allowed missing values</th>
<th>Rare values threshold</th>
<th>Cache size</th>
<th>Outlier elimination percentage</th>
<th>Population size</th>
<th>Survival rate</th>
<th>Variability</th>
<th>Generations</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>1</td>
<td>Any</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>

$k$ of 5 produces the best results for an unweighted $k$-NN.

Figure 7. Determining $k$ for $k$-Nearest Neighbor. $k$ of 5 results in the lowest MRE.

$k$ of 5 produces the best results for an unweighted $k$-NN.
4.1.2 Determining optimal $L_n$ norm

<table>
<thead>
<tr>
<th>$k$</th>
<th>$L_n$ norm</th>
<th>Allowed missing values</th>
<th>Rare values threshold</th>
<th>Cache size</th>
<th>Outlier elimination distance</th>
<th>Population size</th>
<th>Survival rate</th>
<th>Variability</th>
<th>Generations</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>?</td>
<td>Any</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>Not applicable</td>
<td>Unweighted k-NN.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 8. Determining optimal distance norm. A value of 1 was chosen as a compromise solution between accuracy and time.

While an $L_n$-norm of 0.2 results in the smallest MRE, calculation time needs to be taken into account: it takes 3 times as long to process a norm of 0.2 than a norm of 2.8. We decided to use the $L_n$-norm of 1 as a compromise solution, especially since it removes the need to invoke the $\text{pow()}$ function. You can see the outlier at $x = 1$ on the second graph.
4.1.3 Determining allowed missing values

<table>
<thead>
<tr>
<th>( k )</th>
<th>( | L_n |_\text{norm} )</th>
<th>Allowed missing values</th>
<th>Rare values threshold</th>
<th>Cache size</th>
<th>Outlier elimination factor</th>
<th>Population size</th>
<th>Survival rate</th>
<th>Variability</th>
<th>Generations</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1</td>
<td>?</td>
<td>None</td>
<td>50</td>
<td>None</td>
<td>50</td>
<td>0.2</td>
<td>0.88</td>
<td>50</td>
</tr>
</tbody>
</table>

Figure 9. Determining allowed missing values. Having a maximum number of missing values provides no benefit, likely because the dataset size gets severely decreased.

For this part we have decided to evolve weights with a genetic algorithm first, since we were unsure how much effect missing values would have on the evolutionary procedure. The initial parameters of the genetic algorithm were set in an *ad hoc* manner to values that tended to produce reasonable results during the development process; these are also used later in subsections 4.1.4 – 4.1.9. The performance of the algorithm does not change significantly if the number of missing values is above 2. We have decided to allow at most two missing nominal values per instance, since this minimizes the dataset size, while not affecting performance. The decrease in accuracy with fewer allowed missing values may seem counter-intuitive. We assume this happens due to a significant decrease of the training set size: if we allow no missing values, more than half of the instances are rejected.
### 4.1.4 Determining rare values threshold

<table>
<thead>
<tr>
<th>k</th>
<th>L_n norm</th>
<th>Allowed missing values</th>
<th>Rare values threshold</th>
<th>Cache size</th>
<th>Outlier elimination factor</th>
<th>Population size</th>
<th>Survival rate</th>
<th>Variability</th>
<th>Generations</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1</td>
<td>2</td>
<td>?</td>
<td>50</td>
<td>None</td>
<td>50</td>
<td>0.2</td>
<td>0.88</td>
<td>50</td>
</tr>
</tbody>
</table>

![Determining rarity threshold](image)

Figure 10. Determining rare values threshold. As with the number of missing values, having this parameter does not provide any benefits, probably due to decreased dataset size.

Our experiments have shown that actually having a rare values threshold is not beneficial. While the accuracy on the training data improves with a higher rare values threshold, it actually worsens on the test subset. This is true for both unweighted $k$-NN and genetically evolved weighted $k$-NN. As with the missing values, we assume that the negative effects of a smaller dataset overwhelm the positive effects of having fewer rare items.
4.1.5 Determining cache size

<table>
<thead>
<tr>
<th>$k$</th>
<th>$L_n$ norm</th>
<th>Allowed missing values</th>
<th>Rare values threshold</th>
<th>Cache size</th>
<th>Outlier elimination factor</th>
<th>Population size</th>
<th>Survival rate</th>
<th>Variability</th>
<th>Generations</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>?</td>
<td>None</td>
<td>50</td>
<td>0.2</td>
<td>0.88</td>
<td>50</td>
</tr>
</tbody>
</table>

Figure 11. Determining cache size. For each instance in the training set, its cache is a set of its $k'$ closest neighbors during unweighted $k'$-NN. $k'$ is the cache size. During the training process for weighted $k$-NN, only instances from this set are evaluated as potential nearest neighbors. Having a cache significantly decreases the training time.

Interestingly enough, we found out that increasing cache size above 25 does not significantly affect performance, while taking much more time (cache size of 10: 500 seconds, cache size of 200: 3000 seconds).
4.1.6 Determining outlier elimination factor

<table>
<thead>
<tr>
<th>k</th>
<th>( L_n ) norm</th>
<th>Allowed missing values</th>
<th>Rare values threshold</th>
<th>Cache size</th>
<th>Outlier elimination factor</th>
<th>Population size</th>
<th>Survival rate</th>
<th>Variability</th>
<th>Generations</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>30</td>
<td>?</td>
<td>50</td>
<td>0.2</td>
<td>0.88</td>
<td>50</td>
</tr>
</tbody>
</table>

Figure 12. Determining outlier elimination factor. If the distance from an instance \( t \) to its nearest neighbor is larger than the outlier elimination factor multiplied by an average distance from an instance to its nearest neighbor, \( t \) is removed from the dataset. Our results have shown that this approach does not provide substantial accuracy benefits.

Our experiments have shown that using this outlier elimination mechanism does not lead to significant performance improvement on the testing set (contrary to the training set). The results tend to be the best around outlier elimination factor of 4.8, but that is likely not statistically significant because of the large variability (see Fig. 12).
4.1.7 Determining population size

<table>
<thead>
<tr>
<th>K</th>
<th>L_n norm</th>
<th>Allowed missing values</th>
<th>Rare values threshold</th>
<th>Cache size</th>
<th>Outlier elimination factor</th>
<th>Population size</th>
<th>Survival rate</th>
<th>Variability</th>
<th>Generations</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>30</td>
<td>4.8</td>
<td>?</td>
<td>0.2</td>
<td>0.88</td>
<td>50</td>
</tr>
</tbody>
</table>

Figure 13. Determining population size. Population size over 60 or so does not significantly increase accuracy, while taking linearly longer time to evolve.

The performance stabilizes around the population size of 50. We ran two sets of experiments: in the first one, we kept doubling the population size from 10 up to 640, in the second we linearly increased the population size from 10 to 80, hence the distribution of data points on the graph. We assume that increasing the population size over 60 or so does not yield accuracy improvements due to over-fitting.
4.1.8 Determining survival rate

<table>
<thead>
<tr>
<th>$k$</th>
<th>$L_n$ norm</th>
<th>Allowed missing values</th>
<th>Rare values threshold</th>
<th>Cache size</th>
<th>Outlier elimination factor</th>
<th>Population size</th>
<th>Survival rate</th>
<th>Variability</th>
<th>Generations</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>30</td>
<td>4.8</td>
<td>50</td>
<td>?</td>
<td>0.88</td>
<td>50</td>
</tr>
</tbody>
</table>

Figure 14. Determining survival rate. Survival rate defines how many members of the current generation survive till the next one; survival rate of 0.5 results in approximately 50% surviving. See section 3.3 for a complete description. Survival rates between 0.1 and 0.5 seem to provide the best tradeoff between variability and evolution speed.

Our experiments showed a slight increase in accuracy for survival rates in the range of $(0.1, 0.5)$. Survival rate of less than 0.1 may result in too little variability (a few survivors producing all the offspring), while survival rates higher than 0.5 may be slowing down the evolution.
4.1.9 Determining variability

<table>
<thead>
<tr>
<th>k</th>
<th>L_n norm</th>
<th>Allowed missing values</th>
<th>Rare values threshold</th>
<th>Cache size</th>
<th>Outlier elimination factor</th>
<th>Population size</th>
<th>Survival rate</th>
<th>Variability</th>
<th>Generations</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>30</td>
<td>4.8</td>
<td>50</td>
<td>0.3</td>
<td>?</td>
<td>50</td>
</tr>
</tbody>
</table>

Figure 15. Determining variability. The variability parameter defines the amount of mutation during each generation. Variability around 0.96 seems to provide the best results.

Variability parameters around 0.96 tended to provide the most accurate results. This means that on the first generation the child weights can range from $(0.04 \times w, 1.96 \times w)$, where $w$ is the parent’s weight or a combination of the parents’ weights, while on the $50^{th}$ generation this range is $(0.87 \times w, 1.13 \times w)$, and in general for the variability of $v$ and the $i^{th}$ generation it is $(1-v^i; 1 + v^i)$. 
### 4.1.10 Determining generations

![Determining the number of generations](image)

Increasing the number of generations beyond 80 did not seem to yield any accuracy improvements.

### 4.2 Final MRE measurement

We performed all our preliminary measurements on the same “offline” dataset of 5954 instances that passed the first two selectivity layers. By the time the measurements were completed the “online” dataset grew to 10521 instances. Since we are interested in the best set of weights that cover the most data, we will use the entire array of available data for the final measurement (including the original 5954-instance dataset used to establish parameters). Also, one could have noticed large variations in data points for similar parameters in our preliminary measurements (due to random factors and convergence to local minima instead of a global one). To somewhat alleviate this problem, we use 20-fold cross-validation instead of 10-fold.
Note that all of the data that we perform our measurements on is nearly "perfect" – it passed rigorous selection procedures. This is true for both training and test subsets during $n$-fold validations (except that test subsets were not subjected to any selections that required the knowledge of the model, for example outlier removal). For the usability validation part of the results we will 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 laptops listed with incorrect or incomplete information.

<table>
<thead>
<tr>
<th>$k$</th>
<th>$L_n$ norm</th>
<th>Allowed missing values</th>
<th>Rare values threshold</th>
<th>Cache size</th>
<th>Outlier elimination factor</th>
<th>Population size</th>
<th>Survival rate</th>
<th>Variability</th>
<th>Generations</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>30</td>
<td>4.8</td>
<td>50</td>
<td>0.3</td>
<td>0.955</td>
<td>80</td>
</tr>
</tbody>
</table>

![Figure 17. Final MRE measurement. The first column shows MRE over 20 folds of cross-validation. Second column shows MRE over the entire dataset for average normalized weights; these weights are used later during usability validation. Third column is Van Heijst et al. [16] reported result.](image)

The first column shows the average error on the test set over 20 folds of cross-validation. The second column results from averaging the normalized best weights produced over 20 folds, and testing the entire dataset with these weights. For comparison, the third column shows the results of Van Heijst et al. [16] on the Nike's shoes category, but again it needs to be mentioned that their method is more
general than ours. The 20-fold averaged weights are used later on in the automatic part of usability validation (Section 4.3.2).

4.3 Usability validation

While we may have been somewhat lenient on the user interface part, usability validation in our particular case actually does not have much to do with the product’s ease of use; rather, it is concerned with how profitable the product is. In the introduction we mentioned two potential uses of our product: one for casual buyers, one for resellers. It would be complicated to involve a large number of “casual” buyers with our software. Instead, we choose to focus exclusively on the reseller use case, since it does not require that more than one person be involved. Our primary metric is profit/time spent on eBay. The denominator only includes the amount of time it takes the reseller to make a buying decision. It does not include time for the following reseller tasks:

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

All these quantities are almost 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 that may be useful: total amount of profit made, and percent-wise return on investment; however we do not base our final conclusion on them.

While we are focusing our validation on resellers use case, we still provide a web form that anyone can access for free. The web form is currently located at http://ebay.xirax.net; a screenshot is presented on Fig. 18.
4.3.1 Manual benchmarking

Before we can proceed with testing our system, we needed to establish profit metrics for a reseller not using our system. For that benchmark we have two data points:

1) The more recent data point came from purposefully trading on eBay to establish that benchmark. For two weeks straight we spent two hours per day on eBay looking for under-priced buy-it-now laptops. As such, we know both the profit we made, and the time we spent making it.

2) The older data point came from prior eBay trading not for the purposes of this thesis. It spans a period of a couple of months, so there is a much larger...
number of laptops bought and sold, and consequently, the data should be more reliable. However, we did not directly measure the time that was spent trading. Nevertheless, if we assume that the time/purchase value is roughly equivalent between the two data points, we can extrapolate the time for the second data point from the first one.

3) We can also combine both of these data points, using

\[
\frac{\text{profit(data point 1)} + \text{profit(data point 2)}}{\text{real time(data point 1)} + \text{extrapolated time(data point 2)}}
\]

as our metric.

The results are summarized in the table:

<table>
<thead>
<tr>
<th>Data point</th>
<th>Number of laptops traded</th>
<th>Total profit made</th>
<th>Profit/hour</th>
<th>Return on investment</th>
<th>Total time spent</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>$443</td>
<td>$15.82</td>
<td>10.7%</td>
<td>28 hours</td>
</tr>
<tr>
<td>2</td>
<td>41</td>
<td>$2052</td>
<td>$17.87</td>
<td>10.9%</td>
<td>115 hours</td>
</tr>
<tr>
<td>Combined</td>
<td>51</td>
<td>$2495</td>
<td>$17.45</td>
<td>10.8%</td>
<td>143 hours</td>
</tr>
</tbody>
</table>

Table 2. Usability validation baseline. The first row is the data collected specifically for the purposes of this thesis. The second row is information from the earlier trades; we did not directly measure the time it took to perform those.

4.3.2 Automatic usability validation

Automatic usability 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 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 5 minutes or so in a few cases. The results are summarized in the table:

<table>
<thead>
<tr>
<th>Number of laptops traded</th>
<th>Total profit made</th>
<th>Profit/hour</th>
<th>Return on investment</th>
<th>Total time spent</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>$585</td>
<td>$98.04</td>
<td>10.3%</td>
<td>5 hours 58 minutes</td>
</tr>
</tbody>
</table>

Table 3. Automatic usability validation results. Our system provides dramatic profit/hour improvement and also a slight increase in the amount of overall profit.

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. If we factor in the auxiliary time that it takes to process an eBay resale (shipping, testing, repairing, packing, listing time), the relative increase is of course much smaller, but it is still significant. 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 just 2 hours for the manual benchmarking part. There is, however, room to grow. Assuming that these 2 hours are the busiest hours every day, 7-9 PM Mountain Time (which is close to reality), then during the manual validation we covered about 20% of all the laptops offered for sale. 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. The automatic system only achieves about one fourth of that value, due to two main factors: inability to be present by a computer the entire day 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 by easing the automatic alert requirements, but that would result in more time spent evaluating the alerts.
Chapter 5

Conclusion

This thesis presents a novel application of machine learning techniques to a real-life problem. Our system greatly speeds up trading activities 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 takes mere minutes of human time instead of hours. Additionally, our application directly boosts revenues, since contrary to a human trader, it does not require rest and 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 estimate faster than any other method, is completely free, and is capable of storing and evaluating more data than the Terapeak [4], which until now was the best tool on the market.

It is true that the market our system is currently operating in has a very small capacity: see Section 4.3.1 for an estimate, $2200 profit/two weeks. However, there are many opportunities for expansion. While we won’t be able to generate millions of dollars quickly, the potential for slow expansion is nearly limitless. Even 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. That will require additional involvement on the part of a human, since auction prices only become close to market prices during the last minutes of an auction; and even then, there are cases when the price doubles in the last 30 seconds. This will require extremely rapid response from the trader for every single received alert, but is potentially capable of greatly increasing the profits (see Section 3.2 for statistics: the number of auctioned laptops is roughly equal to the number of buy-it-now laptops).

Expanding the approach to other eBay categories is straightforward – we need to redesign our feature set in accordance with category specifics, but everything else is already in place. Even more, expanding to the desktop computers category and the Apple computers category requires no modification whatsoever, since the attribute sets are the same (with the possible exception of the
model/family extraction routine). When the tool is expanded to many categories, it may become feasible to sell access to the tool instead of, or in addition to, directly using it.

As the final proof of the usefulness of our system we would like to mention that we did not turn it off after all the necessary validation procedures were completed.
Bibliography


