Optimal Pricing in E-Commerce Based on Sparse and Noisy Data

Josef Bauer and Dietmar Jannach
TU Dortmund, Germany
email: josef.bauer@ku.de, dietmar.jannach@tu-dortmund.de

Abstract

In today's transparent markets, e-commerce providers often have to adjust their prices within short time intervals, e.g., to take frequently changing prices of competitors into account. Automating this task of determining an "optimal" price (e.g., in terms of profit or revenue) with a learning-based approach can however be challenging. Often, only few data points are available, making it difficult to reliably detect the relationships between a given price and the resulting revenue or profit. In this paper, we propose a novel machine-learning based framework for estimating optimal prices under such constraints. The framework is generic in terms of the optimality criterion and can be customized in different ways. At its core, it implements a novel algorithm based on Bayesian inference combined with bootstrap-based confidence estimation and kernel regression. Simulation experiments show that our method is favorable over existing dynamic pricing strategies. Furthermore, the method led to a significant increase in profit and revenue in a real-world evaluation.

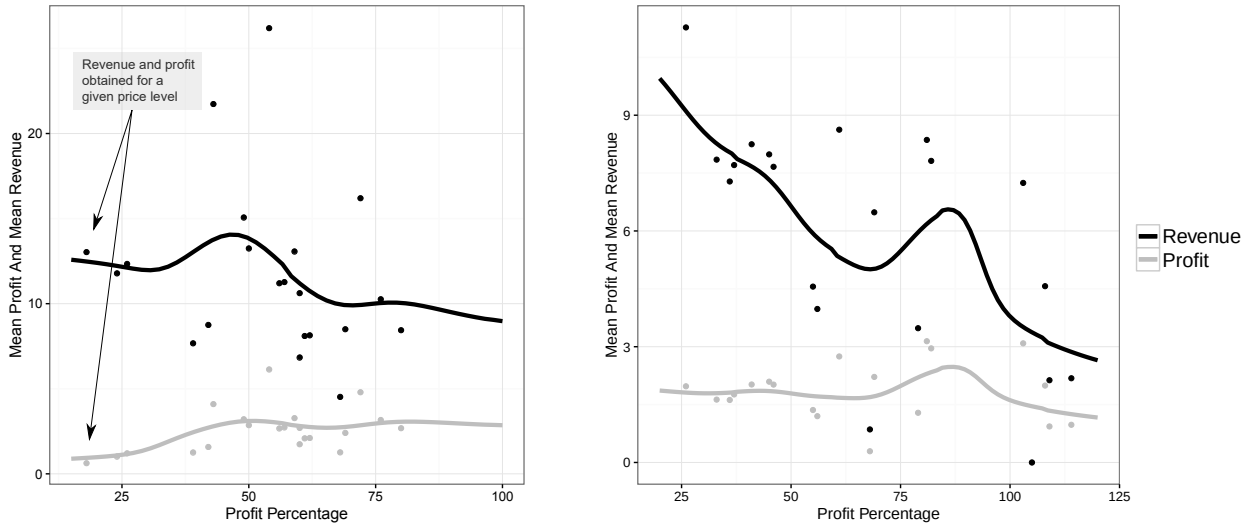
Keywords: Dynamic pricing, e-commerce, machine learning, data mining

1. Introduction

Pricing is a crucial factor for companies to be competitive on today's transparent online markets. Since manual pricing strategies (e.g., based on expert heuristics) do not scale well for larger e-commerce sites and pricing based on simple rules (e.g., adding a certain profit margin per category to the purchase price) might be too coarse-grained, automated decision support systems can represent a valuable tool that help companies find "optimal" prices for their products and can thereby play a key role for business success [1]. A common problem in particular of learning-based approaches is that the of amount past observations can be very limited, i.e., we only have a small set of data points that show how a price change affected sales. Furthermore, sometimes small changes in the price can lead to some unexpectedly large changes in the sales numbers. These two key practical challenges have, however, not been deeply explored in the literature before.

In this work, we propose a novel learning-based framework for *dynamic pricing* which determines optimal sales prices based on observing effects of past price changes for a given article *and for related products*. The framework comprises a combination of techniques to deal with the mentioned practical challenges and at its core implements a pricing algorithm based on Bayesian inference in combination with bootstrap-based confidence estimation and kernel regression. While this particular choice of techniques is specific, the design of the framework itself is general and customizable to different applications, e.g., with respect to the optimality criterion. At the same time, additional side information like competitor prices can be easily integrated into the learning process.

Next, in Section 2, we describe the specifics of our problem scenario. Related works are reviewed in Section 3. After the presentation of technical details in Section 4 we discuss the outcomes of different experimental evaluations along with managerial implications in Section 5.



(a) Product A. (b) Product B.

Figure 1: Raw data points and estimation of the mean revenue (black) and profit (grey) for two products. The curves are obtained via extrapolation with kernel regression, as described in Section 4.2.

2. Detailed Problem Illustration and Baseline Methods

We illustrate the specific problem using data that was collected for two products on the e-commerce shop on which we later on conducted the field studies. Figure 1a shows the observed relationships between different sales prices (expressed as profit percentage[1] on the X-axis) and the resulting mean daily revenue and profit for product A. Each point in the figure corresponds to an observation where product A was sold for a certain price for a number of days. The figure shows that higher prices do not necessarily lead to higher profit and revenue. Increasing the price yields a higher profit per product, but can lead to a lower demand on the market. As Figure 1a shows, the data is also fluctuating strongly and small differences in the price can lead to huge differences in the mean profit and revenue.

Learning-based dynamic pricing is generally concerned with determining a function that captures the relationship between the price and the profit or revenue from the data. The specific problem we address in our work is to find a method that does *not overfit* the function to the given sparse and fluctuating data. One main ingredient of our approach is the use of kernel regression. The curves in the figure are based on this method and indicate its general suitability.

Figure 1b shows the same plot for another product (B). Here, we can observe the additional challenge that the underlying dependencies can have a complex nature. In particular, the learned function for revenue shown in Figure 1b has two local optima. Note that the different pattern for profit compared to revenue is due to the fact that a slight increase in the price has a higher effect on profit per sale than on revenue per sale. The existence of multiple local optima contradicts traditional price sensitivity assumptions but is in line with with previous findings of [2]. One possible explanation for this phenomenon is that there might be two types of customers. One

[1]The profit percentage is calculated as $(((\text{net profit per unit})/(\text{cost price per unit})) * 100)$. We use profit percentages instead of absolute prices to make the estimation for different prices comparable. The profit percentage is also less correlated with the unit cost, and using less correlated predictors is usually preferable.

group consists of price sensitive customers only buy the product if its price does not surpass a certain threshold, e.g., a competitor's price. The other group does not compare prices and accepts higher prices to a certain degree. Only when the threshold of this latter group is reached, the overall profit and revenue start decreasing again. The curves in Figure 1b show that our proposed learning method is capable of detecting such phenomena in the data.

3. Related Work

A number of strategies for dynamic pricing were proposed in the past. We will first discuss common strategies from the literature which we also empirically compared with our proposed method. Then, we will summarize the characteristics and applicability of other comparable approaches for our application scenario. Finally, we review other works in the broader context of dynamic pricing.

3.1. Competition-based pricing

In practice, e-commerce companies often determine their prices based on those of the competition [3, 4]. Such a simple strategy will in many cases lead to prices that are not optimal with respect to profit and sometimes not even in terms of revenue (e.g., for price-insensitive products a larger profit might be better). A general problem of competition-based pricing is that so-called price wars can arise, where competitors match each other's prices, resulting in either a cycling pattern or a downward spiral if both competitors keep on pricing below the other vendor [5].

3.2. Derivative-Following and Model Optimizer Algorithm

In our simulation experiments, we will contrast our method with the *Derivative-Following Algorithm* and the *Model-Optimizer Algorithm* [6, 7], since both methods are generally suited for our application scenario, as will be discussed later.

- *Derivative-Following Algorithm (DF).* This algorithm starts with a price change in one direction. If the profit increases compared to the value of the previous price point, the same direction is used for setting the next price point. Otherwise, the reverse direction is taken.
- *Model-Optimizer Algorithm (MO).* This more sophisticated algorithm considers several historical prices with optional weights that express their relevance based on subjective beliefs. A higher-order polynomial is then fit to the data in order to determine the optimal point.

Both algorithms are generally intuitive at a first glance, but we can identify some limitations that will be confirmed in our experiments later on. The Derivative-Following Algorithm only considers the previous price instead of a longer history. It starts from scratch every time and the amplitude of the price change is also not known in advance, but must be determined by a separate shrinkage or expansion. Ignoring past evidence can also cause the algorithm to get stuck in a wrong pricing area for long. The reason is that large changes in revenue or profit can be observed in reality, even when the sales price was only slightly changed (e.g. due to noise). Thus, the Derivative-Following Algorithm is highly prone to overfitting. Furthermore, as there can be multiple local optima, the algorithm can get stuck in one of them.

Overfitting is also a potential problem of the Model-Optimizer Algorithm, since it uses a higher order polynomial for fitting. Again, the noise and the resulting high variability of the data points might lead to models with a limited generalization ability and poor predictions. Furthermore, it relies on particular parametric assumptions which might be inappropriate for the given setting.

Having these potential limitations in mind, our goal was therefore to design a method that is robust against overfitting and does not rely on specific parametric assumptions. Furthermore, the

resulting framework should be able to accommodate highly relevant side information like competitor prices, as well as past data from other related products to counteract the issue of data sparsity.

3.3. Comparison with Other Pricing Methods for the Given Problem

Besides the DF and MO methods we compared of a number of other existing works ([8], [9], [10], [11], [12], [13], [14]) with ours in different dimensions and analyzed their applicability for the given problem scenario. The results of this analysis are summarized in the following list.

- From the mentioned approaches, only the DF and MO methods ([6, 7]), as well as [8] were specifically designed for e-commerce scenarios.
- The use of additional predictor variables, as supported in our method, is a feature that is also present in [8], but for example not possible in the approaches presented in [9], [10] or for the DF and MO methods.
- Only some methods, like ours, make full use of the available historical data. Others like DF or [10], in contrast, only use a subset of the available information.
- With respect to the model assumptions, a number of previous works rely on restrictive assumptions about the distributions, e.g., [8], [9], [11], [12], [13], [14], which is a potential problem in the targeted application scenario.
- When using our method (like when using DF, MO, or the method from [8]), no detailed information about the customers is required. Other works, e.g., [10], [11], however rely on the assumption that such information is available.
- Finally, among the examined works, our method is the only one that explicitly deals with data sparsity and is specifically designed to avoid overfitting. Furthermore, no other method tries to incorporate data of similar products from the same subcategory, has a customizable optimization function, and relies, at least partially, on models that are interpretable by a domain expert. Finally, none of the discussed methods was evaluated in a field test.

3.4. Dynamic Pricing as Reinforcement Learning: Multi-armed Bandits and Regret Minimization

Iteratively determining new prices in an automated way based on insights from past prices can also be framed as a reinforcement learning problem. Specifically, multi-armed bandit based approaches might represent a principled approach for pricing problems as they are designed to target the exploration-exploitation trade-off, i.e., they are concerned with balancing the payoff maximization based on historical data with the exploration of yet unknown actions (e.g., the effect of using a new price in this setting). To the best of our knowledge, there is however no work that presents a method that could be directly applied to our specific problem setting.

The work that is closest to our research in that context is [15]. This paper proposes the application of multi-armed bandits for dynamic pricing, but it is tailored to different assumptions. In particular, it is a customer-centric approach that is based on the assumption that customer visits are recorded and that the price determines whether a buying action happens. Technically, correlations between arms are introduced by relying on the assumption that if a customer accepts a certain price, he or she will probably accept a cheaper price and vice versa. However, in our application we do not assume to have visitor data available and our work is not based on assumptions about which individual customer accepts which price. In our setting, using visit data is also of limited help, since a majority of the customers monitors price comparison platforms and only visits

the site to make the purchase. Our method was therefore designed to work based only on sales and corresponding revenue/profit data, without the need to have record information.

Nonetheless, we consider the extension or reformulation of parts of our method based on the established concepts of regret minimization and/or multi-armed bandits as a promising avenue for future work. One aspect to consider here is that there is a correlation between different price points in our setting, while most methods in these areas do not make use of such correlations. An exception is the above mentioned paper [15], which introduces correlated arms by relying on visits and customer buying behavior. Since this is not applicable in our setting, an alternative approach is needed. We consider the recently released preprint [16] as a good starting point to approach our problem with multi-armed bandits and regret minimization. Although theoretically optimal strategies for both methods are derived in this work, it does not address the topic of pricing and the specific requirements and extensions needed for this purpose. Furthermore, a common assumption is that the distributions are from the one-parameter exponential family, an assumption which is not needed in our current framework. This assumption also implies that all price points must have the same distribution, just with different mean parameter. In contrast, we empirically found that this is not always the case in practice. Since also the aforementioned correlations between prices are not considered in this work, such an extension is more involved. Nevertheless, [16] provides important theorems that can be used as a basis for an extension of our framework to include regret minimization and multi-armed bandits.

3.5. Existing Approaches in Related Areas

The promise of dynamic pricing has attracted attention in several areas other than traditional e-commerce. In particular, works were proposed in the area of the digital music streaming business [17], for products offered by electronic component suppliers [18], for pricing strategies in the context of cloud computing [19], for electronic markets with customizable products [20], for dynamic pricing for low cost carriers [21], and in the context of auctions [22, 23]. All these methods are however designed to deal with different domain-specific problem situations. It is therefore unclear to which extent they are suitable for the specifics of our problem setting that are common in various e-commerce scenarios and which we address with our method.

On the other hand, there are a few works that address issues that *are* relevant for our application scenario as well but are not yet covered by our approach. In [24], for example, the important relation between pricing and *shipping costs* in the online retail industry is analyzed. In our work, we did not consider shipping costs so far but consider this in the future. The work of [25] puts special emphasis on *fair price discrimination* strategies in e-commerce. One of the key findings of this work is that it is worthwhile to consider that customers might not buy immediately, but could *wait for a cheaper price*. Such effects are not taken into account in our approach so far as well. The work of [26] focuses on dynamic pricing in the fashion industry by developing a method which is based on click-through rates for buying actions. The use of clickstream data could be a helpful addition, with the limitations mentioned above, but was unavailable in our application.

A different class of approaches relies on the estimation of *price elasticity*. In [27], for example, several approaches of that type from the literature are discussed in the context of pricing for software. Various measures for estimating demands are considered, including the Amazon sales rank. The method for determining new prices is then based on the *price elasticity of demand*. Roughly speaking, this measure describes the relative change in demand (like the sold quantity) resulting from a relative change of the price. While such approaches are important and insightful from a conceptual point of view, the underlying assumptions are often not met on a granular product level in practice. As discussed above, this kind of real data can be very sparse and noisy, making reliable estimation of such quantities difficult. And methods based on price elasticity usually

assume a single price optimum, while we found that products can often have several local price optima. We have described probable reasons for this phenomenon in Section 2.

Generally, we further refer to [28] and [29] for an overview of pricing methods in the literature.

4. Proposed Method

4.1. Overview

The main rationale of the different steps of the dynamic pricing algorithm of our framework can be summarized as follows.

(i) Obtain past sales data for the product for which a new price is sought. Calculate the quantities of interest, such as profit or revenue, for the different past sales prices.

(ii) For each historical price, calculate the probability that it is optimal based on the observations for other prices in the past. We do this by considering the variability in profit or revenue that is associated with each historical price. Technically, we use a bootstrap approach which, roughly speaking, considers the daily fluctuations within a restricted time frame in which the price is fixed to assess the reliability of the past observations (Section 4.3). The resulting values can be thought of as confidence estimates that the respective price points are truly the best prices in terms of profit or revenue.

(iii) Since we usually only have a limited number of such past observations, we use a kernel regression approach to learn a model to predict the performance for new price values based on the past outcomes in a robust way (Section 4.2).

(iv) As an additional means to deal with the data sparsity problem, we consider two types of additional information to infer general pricing rules at a higher level: a) past sales data of similar products, e.g., those of the same subcategory and b) competitor prices. We use decision trees to infer such higher-level pricing rules (Section 4.4.3).

(v) We combine the outputs of step 3 and 4 by enriching the predictions at the product level with those of the subcategory level. We apply a Bayesian inference approach and include the higher-level predictions through a prior function in the prediction model (Section 4.4 and Section 4.4.2).

(vi) Finally, we apply an adapted Metropolis-Hastings algorithm to sample new price points for the given product (Section 4.5).

In the following sections, we discuss our technical approach in more depth.

4.2. Using Kernel Regression for Robust Estimates of Fluctuating Sales-based Target Measures

Given a product, our goal is to predict a real-valued target measure, such as the expected profit or revenue, depending on different predictor variables. We model this target measure by a random variable and consider its dependency on the profit percentage p (and thus the price) as well as additional predictor variables which are denoted by a vector $\mathbf{z}$. Estimating the dependency of the target variable on *multiple predictor variables* is challenging in our setting due to data sparsity and noise. In a first step, we therefore propose to consider only the dependency on the profit percentage.

The examples in Section 2 show that we can expect the resulting functions describing this dependency to be complex, potentially having multiple local optima and exhibiting different forms from product to product. Parametric approaches are unlikely to yield good results in this case

and we therefore use a nonparametric method that does not require the target function to have a predetermined form.

Specifically, we propose to use kernel regression because this class of models can be used to estimate complex functions, in particular multimodal ones like the one in Figure 1b. As a specific regression method, we use the Nadaraya-Watson estimator (c.f., [30], Section 6.1, p. 192 f.), because it has the advantage that only one parameter (the bandwidth) has to be tuned. Other nonparametric approaches, such as local polynomial regression, could be used as well in our general framework. Such models, while being theoretically more expressive, usually require more data points because more hyperparameters have to be tuned.

4.2.1. Formalization of the Problem Using the Nadaraya-Watson Estimator

Suppose we are given data points p_n, $n \in \{1, ..., N\}$, where p_n denotes the profit percentage at time period n, with corresponding observed outcomes y_n, $n \in \{1, ..., N\}$.[2] In our problem setting, we consider (sales) observations from a number of consecutive days during which the price was not changed, i.e., a time point n corresponds to a number of days or a week. This allows us to make more reliable estimates. The number of days to be considered for a single observation depends on the specific sparsity situation and can be adapted depending on the outcome of the bootstrap-based confidence estimation process described later in Section 4.3.

Now, for a given Kernel K, the Nadaraya-Watson estimator is defined as

$$f_q(p) := \widehat{Y}_q(p) = \frac{\sum_{n=1}^{N} y_n \cdot 1/h \cdot K((p - p_n)/h)}{\sum_{n=1}^{N} K((p - p_n)/h)}, \tag{1}$$

where h is the bandwidth that has to be estimated from the data. In our experiments, we used a Gaussian kernel K, which is a common choice that exhibits consistency.

The quantity $p - p_n$ measures the distance of a new price point (given by the profit percentage) p and the data point p_n. The underlying principle is that for every point p a locally weighted regression (in the case of Nadaraya-Watson with constant functions) is performed, giving more weights to nearby points, according to the term $K((p - p_n)/h)$. Note that the right-hand side of Equation (1) has to be calculated separately for every point of interest p. This is however not a computational issue, as the number of data points is low and the number of potential new price points p is restricted. This restriction is based on reasonable price ranges and the fact that there is only a finite number of different prices, since they are rounded to cents.

The bandwidth h can be seen as a smoothing parameter addressing the bias-variance tradeoff. If the bandwidth is too low, the variance of the resulting function is high, potentially leading to overfitting to random noise. If the bandwidth is too high, not all characteristics in the data might be captured (underfitting).

To determine the parameter with the best generalization performance, a leave-one-out cross-validation optimization procedure can be used. While such an approach is computationally prohibitive in many applications, it is easily feasible in the given setting, since the range of possible bandwidth values is restricted and the number of data points is usually at most of order 10^2 (e.g., if one data point represents one week, 100 data points correspond to a history of almost two years). Using the Nadaraya-Watson estimator results in a standard quadratic complexity in the number of data points (more precisely, it is $\mathcal{O}(N \cdot n_p)$, where n_p denotes the number of prices for which predictions should be made), but faster binning algorithms exist that reduce it to a linear complexity

[2] We use the notation n to describe different periods in time, while we use t to denote different time points (days) within such time periods in the following sections.

[31]. With an increasing number of data points, the computation demand only increases linearly, so it is scalable. The overall computation time is between minutes and hours, depending on the hardware, the history and the total number of products.

Furthermore, since the patterns in the data can change over time, it can be useful to only consider restricted time windows, i.e., only using the newest points p_n. As an alternative to consider changes over time, we could modify Equation (1) and apply time-based decay weights.

The examples in Figures 1a and 1b were based on Nadaraya-Watson kernel regression. One can indeed observe that the estimator was able to smooth the large fluctuations between neighboring points while at the same time it captured the multimodal nature of the data in Figure 1b.

4.3. Bootstrap-based Confidence Estimation

So far, we have discussed how to estimate the mean daily profit and the mean daily revenue dependent on the profit percentage of a product. However, there can be substantial variability, and thus uncertainty, in each observation of profit or revenue. To obtain more reliable predictions of the optimal prices, our goal is therefore to quantify this uncertainty. In this section, we present a bootstrap-based approach to derive confidence estimates that a certain price point is optimal. Correspondingly, we adapt our goal in a way that we no longer aim to predict quantities like revenue or profit, but instead aim to estimate the probability that a price point is optimal. Based on these probability estimates for single points, we are able to infer the probabilities for the whole price range by using the kernel regression approach that we described in the previous section.

4.3.1. Background

Given the variability in the data, which can depend on the length of the considered time frame and can vary across different price points, the ideal case would be to have confidence intervals for each observation. However, the challenge is that the true underlying distribution is unknown. Using common assumptions like a normal distribution with its respective confidence intervals is unlikely to capture the true distribution of the data points, which is often zero-inflated.

Bootstrapping is a generic way of addressing such problems. The general principle of a bootstrap approach is to approximate the true distribution by drawing *with replacement* from the empirical distribution [32]. The sample size of each bootstrap replicate is the same as the sample size of the empirical distribution, where the empirical distribution is given by the underlying data points, i.e., each day in such a time frame is a data point and all data points determine the empirical distribution. This enables to measure the variability in the data.

In order to improve the prediction accuracy, we used a minimum number of 7 to 14 days per observation in our experiments. Moreover, we will average the bootstrap samples by kernel regression to increase the reliability of the results, as will be described in more detail below. In the following, we will focus on the percentile bootstrap method and derive a statistical test for determining whether a given price point is optimal.

4.3.2. A Bootstrap-based Algorithm to Estimate the Probability that a Price is Optimal

Choosing a Target Measure. Which measure we seek to optimize, e.g., revenue or profit, typically depends on the domain and business requirements. In our application scenario, we are interested in a combination of revenue and profit. Focusing only on revenue might lead to profits that are close to zero. Focusing on profit alone, on the other hand, can lead to lower revenues which is bad for company growth. We can deal with such a trade-off situation in our framework by defining a measure that combines profit and revenue as follows:

$$y := \lambda \frac{profit}{max.profit} + (1 - \lambda) \frac{revenue}{max.revenue},$$

where $max.profit$ and $max.revenue$ denote the (empirical) maximum of profit and revenue, respectively, and $\lambda \in [0, 1]$ is a trade-off parameter determining the weight of profit and revenue. We have used $\lambda := 1/2$ in our field test. Note that our method is flexible and in general independent of this choice. The measures, including the trade-off, can be chosen in accordance with business objectives.

Testing a Price for Optimality. Given the target measure, our goal is to design a statistical test that tells us if a certain profit percentage is likely the best. Note that according to the problem that we want to solve, it is not necessary to precisely estimate the value y for each point. In fact, we are only interested in how probable it is that a given price is optimal based on the evidence we obtained so far. Specifically, we want to know how probable it is that the value is at least as high as the largest value that we have already observed in the data. Our final algorithm will then draw prices based on these estimated probabilities.

Formally, the objective of our test is to estimate

$$c_n := \mathbb{P}\left(Y\left(p_n\right) \geq \widetilde{y}\right), \tag{2}$$

where $\widetilde{y}$ denotes the best value that was achieved in the past. The higher the value for c_n, the more confident we are that this is the best point to choose, since c_n describes the probability that the value is higher than the best achieved value in the past. The most straightforward choice for $\widetilde{y}$ is to use $\widetilde{y} := \max_{n=1,\ldots,N} y_n$. However, in practice it can be helpful to lower this threshold a bit, since the empirical maximum can be the result of special circumstances. For instance, additional sales could have happened because of a cross-selling promotion at a given time. When the threshold $\widetilde{y}$ is lowered, more exploration is done by the algorithm, thus avoiding to get stuck early in a suboptimal point. One way to find this threshold is to use the τ-th sample quantile for a high value of τ and to treat τ as a hyperparameter. Its best value can be determined with a separate validation, by splitting the data randomly into folds belonging to a different threshold. Applying the algorithm to each of them and comparing the relative gains yields an estimate for the best threshold (in our application, $\tau = 0.9$ worked well in most cases). The key reason for using this criterion is the fact that uncertainty due to random variations is taken into account. Only considering an empirical average can be highly biased by a small sample size. However, we cannot directly calculate c_n, since the corresponding distribution is unknown. For this reason, we use the aforementioned bootstrap-based statistical test which can be summarized as follows: *(i)* For every price in the past indexed by n, get the target values $y_{n,t}$ for each day t with this price, which determine the empirical distribution. *(ii)* Take a sample of the same size with replacement form these data points, record the mean value and repeat this many times (e.g., for 1000 replications). *(iii)* Count the percentage of bootstrap samples whose mean value is greater or equal to the threshold $\widetilde{y}$. This quantity is our estimate for the confidence c_n. Note that according to the central limit theorem, the distribution of the mean has a limiting normal distribution. For a sufficient number of data points, the variance around the means gets arbitrarily small. This allows for an accurate estimation, which can be achieved by the bootstrap approach.

In order to increase stability, only prices with a minimum number of observations (e.g., for 7 to 14 days) are considered and our resulting algorithm only changes the prices after this period. The final algorithm will use these confidence values and applies kernel regression to extrapolate to yet unseen prices, following the intuition that new prices are more likely good prices if they are close to well-performing prices. Note that the bootstrap procedure increases the computation time by a constant, but has a linear complexity in the number of prices and in the number of days per price.

Illustrative Example. In Figure 2, which is based on the same data as Figure 1a, we plot the estimated confidence values as the dependent variable and show the curve that resulted from the

application of the Nadaraya-Watson estimator. Comparing Figure 2 and Figure 1a, we see that the points for the confidence-based plot are more concentrated than in the plot that is based on revenue and profit. For most of the points the value is zero, because the observed revenue and profit in the past was so low that the estimated probability of being optimal is negligibly small.

Note that kernel regression is used for smoothing since the variability of neighboring points can still be high and averaging will give the correct result in the limit for sufficient data.

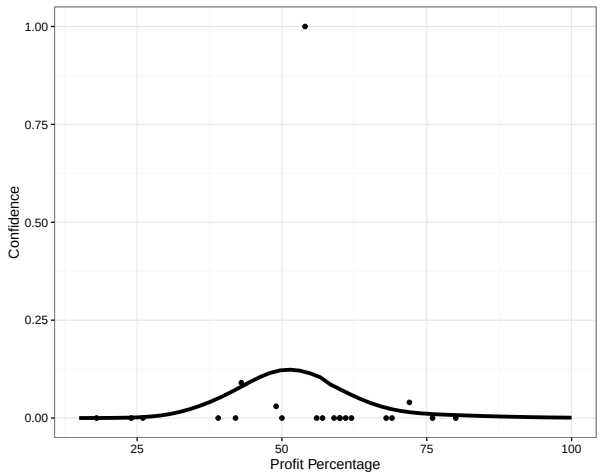


Figure 2: Bootstrap-based Confidence Estimates for Product A.

4.4. Incorporating Additional Information Through a Bayesian Approach

So far, our estimates are based on past sales information for individual products. However, the number of sales per product can be very low for a larger number of the catalog items (long tail problem) even after months. Even worse, for new items in the shop, no past sales data exists at all. To address these issues, we propose to determine optimal price estimates also for *entire subcategories* of the shop, as such estimates can be based on more data. These "higher-level" estimates that are derived for the entire subcategory can then be combined with the estimates at the level of the individual product.

4.4.1. General Idea

The underlying intuition of the approach is to assume that shop items of the same (sub)category may exhibit similar patterns regarding the effects of different sales prices. For example, the pattern observed for board games can be similar for the whole subcategory, not depending as much on an individual product. Also, it is reasonable to assume that the unit cost for the vendor and the prices of the competitions are important predictors. Taking these types of information into account at the product level is however challenging due to the limited amount of data points. But this might not be the case when we consider entire subcategories. As a result, we are no longer analyzing the effects of selling at a cheaper price for a particular product, but rather for the category of board games as a whole. If enough data is available, we can alternatively break down the category in different groups based on their attributes, e.g., into costly and cheap board games.

To combine the models obtained on the product and category level, we propose to use a method based on Bayesian inference. Specifically, our goal is to use the information that we can derive on the higher (category) level as a *prior* when we make the estimates on the product level.

The idea of our Bayesian inference approach is to use the price estimate for the subcategory only as a starting point (prior) and incrementally decrease the importance of this estimate when more data for the individual product becomes available. In case the pattern for the individual product deviates from the pattern for the subcategory, the information obtained from the subcategory level should be downweighted. This is precisely what we will achieve in the next section.

4.4.2. Formal Bayesian Model

Let f denote the function obtained from the kernel regression estimate based on the bootstrap confidence relation (2), i.e.,

$$f_q(p) := \widehat{Y}_q(p) = \frac{\sum_{n=1}^{N} c_n \cdot 1/h \cdot K((p-p_n)/h)}{\sum_{n=1}^{N} K((p-p_n)/h)}, \tag{3}$$

so in case of sufficient data it approximately holds that

$$f_q(p) \propto \mathbb{P}(Y(p) \geq \widetilde{y}). \tag{4}$$

We now assume that in case of insufficient data this target can be approximated by a function which does not explicitly depend on the given product q, but rather on the corresponding subcategory and on other predictors summarized in a vector $\mathbf{z}$. We denote this function by $g(p, \mathbf{z})$, which can be thought of acting as a prior. Together we have:

$$\mathbb{P}(Y(p) \geq \widetilde{y}) = \frac{f_q(p) \cdot g(p, \mathbf{z})}{\int f_q(p) \cdot g(p, \mathbf{z})\, dp}. \tag{5}$$

The factor $g(p, \mathbf{z})$ can be seen as a guidance to prices which are more likely to be optimal in case of insufficient data for a single product. As we can deduce from plugging (2) into (3), in the limit $N \to \infty$ the estimated function f_q converges to a function which equals 1 at optimal price points and is zero elsewhere (i.e., the spikes become more and more concentrated around the optimal points). This implies that the influence of $g(p, \mathbf{z})$ vanishes as long as $g(p, \mathbf{z}) > 0$, which we require in the following section.

4.4.3. Estimating the Prior Function with Decision Trees

We now discuss the problem of finding the modulation factor $g(p, \mathbf{z})$ in more depth. While there are many different machine learning methods that could be used to estimate the prior function g, we propose the usage of decision trees, namely regression trees or conditional inference trees. Decision trees have two main advantages. First, such trees can model nonlinear relationships which can likely occur in our problem setting. In particular, no manual encoding of application specific rules is needed; instead, these rules are learned from the data. Second, a learned tree model can be understood and interpreted, since its output is given by a set of rules. Therefore, the result can be verified by domain experts and new insights into the underlying dependencies can be obtained.

Decision trees are in general constructed as follows. Given a set of independent variables, a *decision tree learning algorithm* greedily seeks to find a best *split variable* and *split point* in each step and approximates the target by piecewise constant functions. The result is a partition of the data in the form of a set of multiple binary splits corresponding to rules.

A simple example of such a tree can be seen in Figure 3. It is based on a specific subcategory and was obtained in our experiments. Each underlying data point corresponds to one day on which a product in this subcategory group was offered at a certain price and for which the resulting average total profit per product and day were measured. In particular, we have used the CART algorithm in our experiments.

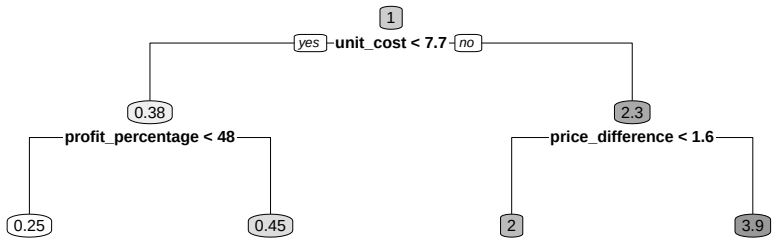


Figure 3: Regression tree for the total mean profit per product and day within a subcategory

As we can see in the tree, the first and thus most important split was done with respect to the cost of products. The variable `price_difference` is defined as the minimum competitor price minus the own price and is important for the next split. While such tree-based rules could in principle already be used to determine prices for the products, they would ignore the patterns for individual products, which can differ from the pattern that was observed for the subcategory. This is why we propose to include these tree-based predictions as nonzero prior weight functions $g(p, \mathbf{z})$ in Equation (5). Note that these predictions are updated each time new data becomes available.

4.5. A Metropolis-Hastings-based Algorithm for Probabilistic Sampling of Optimal Prices

In this section we bring together the different pieces of our method and describe the resulting algorithm for sampling new price points.

4.5.1. Preliminaries

In the previous sections, we have seen how we can determine a confidence value for every price point that reflects how likely it is that the given price is optimal. However, choosing the price with the highest expected confidence based on past observations can lead to undesired stationarity. If not many different prices were tested, it can happen that the best price is in an unexplored region.

Our goal is to design a method that samples new price points in a probabilistic manner, proportional to the estimated confidence that a given price is optimal. For this purpose, we use a tailored variant of the basic Metropolis-Hastings Algorithm [33]. The general idea of these kind of algorithms is to construct a Markov Chain whose stationary distribution coincides with a distribution of interest. In our case, this is the distribution described by Equation (5) in Section 4.4.2. Roughly speaking, in the algorithm we iteratively create new sample candidates which are either accepted or rejected depending on whether they are likely to be optimal.

To determine how a new sample is drawn based on a current sample, the Metropolis-Hastings method uses a so-called *proposal distribution*. A Gaussian proposal distribution works well in many cases in practice. However, for our case, we found that a truncated normal distribution (cf. [34]) is more suitable. The reason is that the region of possible prices is restricted and a truncated normal distribution directly takes this asymmetry into account. The truncation is determined by prescribed price ranges (i.e., category dependent margin thresholds specified by the business owner). Moreover, MCMC methods require a burn-in period, which means that samples are only used after a certain number of iterations n_{it} in order to ensure convergence to the true distribution. A value of $n_{it} = 1000$ iterations is often used in the literature, but we found that in our case 100 iterations are already sufficient.

4.5.2. Final Algorithm

The details of our method for sampling new prices are shown in Algorithm 1. Let $\tau(.|p)$ denote the density of the truncated normal distribution with mean p. The truncation is determined by the lowest and the highest possible prices. Using a standard deviation of $\sigma = (p_{max} - p_{min})/4$ worked well in our cases. [3] The truncated normal distribution ensures that only points within the permitted range are sampled.

A central step of the algorithm is the computation of the *acceptance ratio* a, which compares the probability of a newly sampled price point with the probability of the previously sampled price point.

5. Evaluation

5.1. Simulation-based Evaluation

We compared our method with the Derivative-Following and the Model Optimizer algorithms as introduced in Section 3.2 in a simulation experiment. We created a simple but realistic scenario that was inspired by observations made for real-world data in our field test. Specifically, the data that was made available for the algorithms is sparse and rapid changes in sales could occur when a certain price threshold, e.g., the competition's minimal price, was crossed. At the same time, we tried to keep the simulation simple, such that it can be easily replicated by other researchers.

5.1.1. Simulation Design

We considered a single product with a cost of \$100 and a price range between \$101 and \$200, i.e., a profit percentage between 1 and 100. Suppose that the minimal price of the competition is \$150 in this scenario. We assumed that up to this price the true demand for one day follows a Poisson distribution with parameter $\lambda = 1$. Poisson distributions are often used to model demands and the parameter value of 1 states that on average every day one piece of the product is sold. Note that due to the constant demand the total profit will increase up to \$150 in this setting, because of the increasing percentage in profit p for each sold item.

If the price exceeds \$150, it is reasonable to expect that the demand drops notably. In particular, we assumed that the demand follows a Poisson distribution with parameter $\lambda = 0.5 \cdot 50/p$, i.e., after an immediate drop the demand keeps decreasing up to the highest price \$200, for which the profit percentage is 100 and thus λ equals 0.25. We can verify either through a numerical analysis or through a simulation that a price of \$150 in the end yields the highest average total profit and that this price is therefore optimal.

We implemented a simulation process in which each of the pricing algorithms was used to compute a new price after periods of 14 days. For each of these days, the assumed demand was drawn randomly according to the respective Poisson distribution. Using this value, the total profit for all 14 days was calculated. The first two prices were drawn randomly and then 10 consecutive 14-days periods were simulated. For these periods, the new prices were generated by each of the algorithms. Each of the prices determined the respective Poisson distribution based on which the demand was sampled. The final predictions of the optimal price of each algorithm were then used

[3] We tested various alternative deviations in a grid search and analyzed the convergence behavior on a validation set.

Algorithm 1: Main algorithm for sampling new price points

for *every product* $q \in S$ **do**
  Apply a bootstrap-based method (Section 4.3.2) to calculate the confidence estimates c_n determined by Equation (2);
  Apply kernel regression to the confidences by the approach outlined in Section 4.2. In particular, use Equation (1) with $y_n := c_n$ to estimate the regression function $f_q(p)$.
end
Use the data of all products from subcategory S to generate the tree-based estimation function $g(p, \mathbf{z})$ on the subcategory level, as described in Section 4.4.3;
Apply the following adapted Metropolis-Hastings method for sampling a new price according to Equation (5):
for *every product* $q \in S$ **do**
  Initialize $p^{(0)}$;
  for $it \in \{1, ..., n_{it}\}$ **do**
    Draw $p^{(it)} \sim \tau\left(. \mid p^{(it-1)}\right)$;
    Compute the acceptance ratio $a := \frac{f_q\left(p^{(it)}\right) g\left(p^{(it)}, \mathbf{z}\right)}{f_q\left(p^{(it-1)}\right) g\left(p^{(it-1)}, \mathbf{z}\right)} \cdot \frac{\tau\left(p^{(it-1)} \mid p^{(it)}\right)}{\tau\left(p^{(it)} \mid p^{(it-1)}\right)}$;
    if $a \geq 1$ **then**
      $p^{(it)} := p^{(it)}$;
    else
      $u \sim Uniform(0, 1)$;
      if $u \leq a$ **then**
        $p^{(it)} := p^{(it)}$;
      else
        $p^{(it)} := p^{(it-1)}$;
      end
    end
  end
  Return the sample $p_q^{(new)} := p^{(n_{it})}$ as the new price point for product q;
end

to compare their performance, i.e., the final prices were used to determine the resulting demand and to calculate the corresponding total profit for 14 days. For consistency, we have also applied the same approach for revenue instead of profit as the objective.

5.1.2. Simulation Outcome

This process comprises a notable amount of randomness and therefore we repeated it 1000 times for each algorithm and averaged the result. We thus compare the mean total profit as well as the mean total revenue that was at the end generated by each algorithm. The results of the simulation are shown in Table 1.

Table 1: Simulation-based comparison of algorithms

Method	Total mean profit (left column) and revenue (right column) for one product in 14 days generated after 10 steps (including 95 % confidence intervals)	
Derivative-Following Algorithm	460.47 (451.47, 469.86)	1533.33 (1504.19, 1563.22)
Model Optimizer Algorithm	493.55 (483.69, 503.79)	1679.91 (1652.59, 1708.37)
Proposed Approach	553.94 (544.01, 563.97)	1740.99 (1711.48, 1775.05)

The results show that our method significantly outperforms both other methods in terms of the obtained mean profit and mean revenue. Note that the fact that the 95 % confidence intervals do not overlap is a stronger result and implies that the differences are statistically significant (at a test level of at least 0.05). The theoretical optimum for the profit is \$700, which would be obtained at a price of \$150. Because of the underlying randomness and rarity of sales, the theoretical optimum can however only be reached when a large number of observations is available. Notice that the performance differences for revenue are not as high as for profit. The reason for this is the setup of our simulation scenario, in which there is less variation in revenue than in profit between prices.

It is not surprising that the Derivative-Following Algorithm yields the weakest result in this simulation, because it only considers the most current improvement without considering the past. Because of the noise in the random process it can therefore easily be misled in the wrong direction. In contrast to the other algorithms, our method proved to be more robust with respect to the variability in the observations. Note that for a fairer comparison, we did not include the competitor price as a variable in our model, since the other algorithms are not able to take it into account. We thus assumed it to be unobserved, having only an indirect impact on the resulting optimal profit curve. In practice, especially when several products in a category are observed, our algorithm can be expected to work even better, since the dependency on the minimum competitor price can be estimated by means of the prior function, as described in Section 4.4.

5.2. Online Evaluation

The proposed method was applied and tested for a number of shop categories of a large European e-commerce company focusing on products for children and families. The pricing algorithm is in productive use as part of the company's online system since 2015, was updated in 2016, and is still part of their e-commerce system today. The proposed method was successfully applied to 28 different categories, with products mainly belonging to the drugstore, toys and fashion category. In total, these categories comprise more than 4500 products. We must add that it has not yet been used for categories with higher-priced products and certain key items like food. In these categories customers are highly price-sensitive and there is usually almost no margin to vary the prices.

For these highly price-sensitive categories, the company uses a simpler pricing method based on the minimum competitor price. Prices are automatically updated on a daily basis and set to a value that is equal or slightly below the minimum competitor price. We used this method as an additional baseline in one of our experiments described below.

Our dynamic pricing algorithm runs on a daily basis, since daily aggregates are used due to data sparsity. For the same reason, a minimum number of days is calculated for which a price is kept fixed. This number of days is calculated based on the sales rate of a product. The computational requirement is at most 2 hours of CPU time (using 2 cores and 16 GB of RAM) every day.

We evaluated our method in two ways. First, we analyzed how much additional profit and revenue the method generated *over time*. Second, we used A/B testing to compare our method with the baseline based on the minimum competitor price described above.

5.2.1. Observed Improvements over Time

For this measurement, we considered a particular baseline month 0 as a starting point for the analysis. In month 0, the prices for all relevant products where either set to the minimum competitor price or set manually by category managers in the company. In the following months the prices were dynamically updated by our method. We then compared the profit and revenue that was obtained in each month across all products that were subject to the dynamic pricing strategy.

The results of this measurement over time are shown in Figure 4. Both revenue and profit increased gradually over time and reached a maximum increase of 28.04 % and 20.64 % in revenue and profit, respectively, after four months of measurement. A gradual increase is expected, since the proposed method is able to learn from additional data points that become available over time.

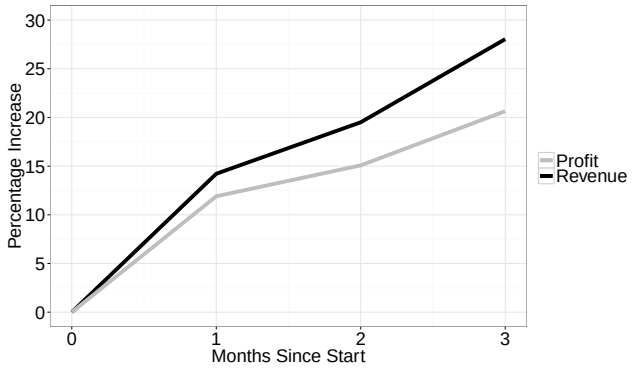


Figure 4: Improvements achieved by of our method over time.

The improvements of profit and revenue are both substantial. The relative improvement in terms of revenue is slightly larger than the improvement in profit. This indicates that for some products an increase in both profit and revenue was achieved when using a higher price, at least up to some threshold. For other products, a price reduction possibly helped to increase the overall profit to some extent because of the additional sales, but the corresponding increases in revenue were more pronounced in this case.

When interpreting the obtained results it is important to know that the overall sales figures of the company (across all categories, including those unaffected by our method) were almost constant for all four months in consideration, with a maximum deviation of less than 3 %. Therefore, the performance of these months can indeed be compared without adjustments. Likewise, there was no increase in the availability of the affected products in the evaluation period, which also ensures comparability.

5.2.2. Outcomes of an A/B Test

During the A/B test, which lasted for over one year, the products were randomly assigned to one of two methods – our algorithm and the one based on the minimum competitor price as mentioned above – and the prices for the products were determined by the respective methods. The products in the treatment and the control group were randomly selected, with the condition that they have a comparable sales rank (i.e., general popularity) in each subcategory. We used a 50:50 split for the field test. After the successful experiment, the company kept about 10% of the products in the control group to be able to continuously monitor the performance of the method.

The main result of the A/B test is shown in Figure 5. Our method lead to 17.79 % increase in terms of profit and a 3.02 % increase in revenue compared to the used baseline method.

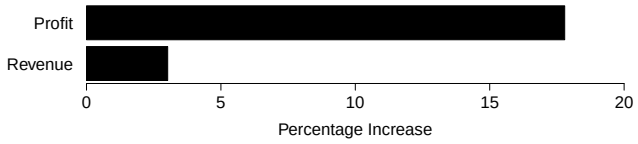


Figure 5: Profit and revenue improvements obtained in the A/B test.

The improvement in revenue is markedly lower than in the analysis that was done over time in the previous section. One probable explanation for this phenomenon is that in the time-based analysis not all products were priced based on the competition in month 0, but were set manually by category managers. This was not the case for the A/B test, which was exclusively based on minimum competitor prices. Intuitively, it appears plausible that competitive pricing is already a promising way to increase revenue. It is therefore difficult to improve upon this value.

The 17.79 % increase in profit can in contrast be considered to be substantial. Increasing profit is in fact usually a harder problem than increasing revenue, because a price reduction implies a smaller decrease in revenue per sale than in profit per sale. Thus, with increasing sales at a lower price, the total revenue can increase sharply, while total profit might even decrease. Finally, it is also remarkable that our method was able to increase the total profit substantially and at the same time led to a slight revenue increase. This indicates that our method of balancing profit and revenue in the optimization process was effective. Moreover, the performance difference in the A/B test was significant at a level of 5% (p-value < 0.031 for revenue and p-value $< 10^{-5}$ for profit).

5.3. Managerial Implications

The experiments in this section have shown that our method is effective in terms of automatically determining prices that lead to increases *both* in revenue and profit. Apart from this core functionality, the general design of the framework has a number of further important characteristics that are relevant in practice.

First, the proposed framework is flexible and adaptable in different ways – as discussed in the technical parts of the paper – and can thus be easily modified according to the needs of a particular company. For instance, the framework is not designed to work only with one particular target measure, but allows for a combination of profit and revenue or any other business metric. Moreover, additional variables that are important in a given domain can be easily incorporated, because the framework features a flexible method to incorporate any number of additional predictors (e.g., seasonality, stock status etc.). Finally, our method is suitable to aid in important managerial decisions regarding a goal-oriented pricing strategy. When decision trees are used to generate the prior functions, parts of the outcomes of our method can be easily understood and interpreted.

6. Summary and Future Work

We have presented a novel algorithmic framework for the estimation of optimal prices in e-commerce settings with sparse and noisy data. Our framework is generic and can be customized to the specific problem at hand. Experimental evaluations have shown that applying our framework can lead to substantial gains in terms of profit and revenue in practice.

There are several directions for future work. As mentioned in Section 5.2, this includes the application of our method in high-price segments. Customers tend to be more price-sensitive for such products, which requires further tuning of the allowed price range to avoid customer dissatisfaction. We also plan to apply our method to very frequently sold items in the food category, which also exhibit a high price sensitivity. Since demand dynamics can change very quickly in this case, we consider *online learning* combined with gradient boosting as a promising technical approach to this problem.

7. References

[1] M. D. Smith, J. Bailey, E. Brynjolfsson, Understanding the Digital Economy: Data, Tools, and Research, 2nd Edition, MIT Press Cambridge, 2002, Ch. Understanding digital markets: Review and assessment, pp. 99–136.

[2] J. D. Abbey, J. D. Blackburn, V. D. R. Guide, Optimal pricing for new and remanufactured products, Journal of Operations Management 36 (2015) 130–146.

[3] P. K. K. PK Kannan, Dynamic pricing on the internet: Importance and implications for consumer behavior, International Journal of Electronic Commerce 5 (3) (2001) 63–83.

[4] W. Zhang, Y. C. Zhu, L. H. Wu, Empirical analysis on price level changes and price adjustment policy selection: Evidence from B2C e-commerce market, International Journal of u-and e-Service, Science and Technology 9 (4) (2016) 69–76.

[5] M. Xie, J. Chen, Pricing and ordering strategies of e-retailers in electronic business environments, Tsinghua Science and Technology 10 (S1) (2005) 778–789.

[6] P. Dasgupta, R. Das, Dynamic pricing with limited competitor information in a multi-agent economy, in: Proceedings CoopIS '16, Springer, 2000, pp. 299–310.

[7] P. Dasgupta, L. E. Moser, P. M. Melliar-Smith, Electronic Business: Concepts, Methodologies, Tools, and Applications, IGI Global, 2008, Ch. Dynamic Pricing for E-Commerce, pp. 393 – 400.

[8] T. K. Ghose, T. T. Tran, A dynamic pricing approach in e-commerce based on multiple purchase attributes, in: Proceedings Canadian AI Conference AI '10, 2010, pp. 111–122.

[9] E. Cope, Bayesian strategies for dynamic pricing in e-commerce, Naval Research Logistics (NRL) 54 (3) (2007) 265–281.

[10] S. Ramezani, P. A. Bosman, H. La Poutré, Adaptive strategies for dynamic pricing agents, in: Proceedings WI/IAT '02, IEEE Computer Society, 2011, pp. 323–328.

[11] N. Nechval, M. Purgailis, K. Nechval, Weibull model for dynamic pricing in e-business, in: Conference on e-Business, e-Services and e-Society, Springer, 2011, pp. 292–304.

[12] V. F. Farias, B. Van Roy, Dynamic pricing with a prior on market response, Operations Research 58 (1) (2010) 16–29.

[13] V. R. Chinthalapati, N. Yadati, R. Karumanchi, Learning dynamic prices in multiseller electronic retail markets with price sensitive customers, stochastic demands, and inventory replenishments, IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 36 (1) (2006) 92–106.

[14] C. Kwon, T. L. Friesz, R. Mookherjee, T. Yao, B. Feng, Non-cooperative competition among revenue maximizing service providers with demand learning, European Journal of Operational Research 197 (3) (2009) 981–996.

[15] F. Trovo, S. Paladino, M. Restelli, N. Gatti, Multi–armed bandit for pricing, in: Proceedings of the European Workshop on Reinforcement Learning (EWRL), 2015.

[16] E. Kaufmann, A. Garivier, Learning the distribution with largest mean: two bandit frameworks.

[17] T. H. Ko, H. Y. Lau, A decision support framework for optimal pricing and advertising of digital music as durable goods, IFAC-PapersOnLine 49 (12) (2016) 277–282.

[18] W. Chung, S. Talluri, R. Narasimhan, Optimal pricing and inventory strategies with multiple price markdowns over time, European Journal of Operational Research 243 (1) (2015) 130–141.

[19] J. Huang, R. J. Kauffman, D. Ma, Pricing strategy for cloud computing: a damaged services perspective, Decision Support Systems 78 (2015) 80–92.

[20] R. Aron, A. Sundararajan, S. Viswanathan, Intelligent agents in electronic markets for information goods: customization, preference revelation and pricing, Decision Support Systems 41 (4) (2006) 764–786.

[21] S. Christ, Operationalizing dynamic pricing models: Bayesian demand forecasting and customer choice modeling for low cost carriers, Gabler Verlag, 2011.

[22] S. S. Reynolds, J. Wooders, Auctions with a buy price, Economic Theory 38 (1) (2009) 9–39.

[23] M. Schwind, Dynamic pricing and automated resource allocation for complex information services: Reinforcement learning and combinatorial auctions, Springer Berlin Heidelberg, 2007.

[24] Y. Yao, J. Zhang, Pricing for shipping services of online retailers: Analytical and empirical approaches, Decision Support Systems 53 (2) (2012) 368–380.

[25] J. Wu, L. Li, L. Da Xu, A randomized pricing decision support system in electronic commerce, Decision Support Systems 58 (2014) 43–52.

[26] T.-M. Choi, P.-S. Chow, T. Xiao, Electronic price-testing scheme for fashion retailing with information updating, International Journal of Production Economics 140 (1) (2012) 396–406.

[27] A. Ghose, A. Sundararajan, Evaluating pricing strategy using e-commerce data: Evidence and estimation challenges, Statistical Science 21 (2) (2006) 131–142.

[28] Y. Narahari, C. Raju, K. Ravikumar, S. Shah, Dynamic pricing models for electronic business, Sadhana 30 (2-3) (2005) 231–256.

[29] A. V. den Boer, Dynamic pricing and learning: historical origins, current research, and new directions, Surveys in Operations Research and Management Science 20 (1) (2015) 1–18.

[30] J. Friedman, T. Hastie, R. Tibshirani, The elements of statistical learning, Vol. 1 of Springer Series in Statistics, Springer, Berlin, 2001.

[31] V. C. Raykar, R. Duraiswami, L. H. Zhao, Fast computation of kernel estimators, Journal of Computational and Graphical Statistics 19 (1) (2010) 205–220.

[32] P. Hall, The Bootstrap and Edgeworth Expansion, Springer Series in Statistics, Springer, Berlin, 1992.

[33] C. P. Robert, G. Casella, Monte Carlo Statistical Methods, Springer Texts in Statistics, Springer-Verlag New York, 2005.

[34] J. Burkardt, The truncated normal distribution, Online at `http://people.sc.fsu.edu/%7Ejburkardt/presentations/truncated_normal.pdf` (2014), accessed October 2017.