Mach Learn (2012) 88:369–398
DOI 10.1007/s10994-012-5295-6

How to reverse-engineer quality rankings

Allison Chang · Cynthia Rudin · Michael Cavaretta · Robert Thomas · Gloria Chou

Received: 23 June 2011 / Accepted: 3 May 2012 / Published online: 3 June 2012

Abstract A good or bad product quality rating can make or break an organization. However, the notion of "quality" is often defined by an independent rating company that does not make the formula for determining the rank of a product publicly available. In order to invest wisely in product development, organizations are starting to use intelligent approaches for determining how funding for product development should be allocated. A critical step in this process is to "reverse-engineer" a rating company's proprietary model as closely as possible. In this work, we provide a machine learning approach for this task, which optimizes a certain rank statistic that encodes preference information specific to quality rating data. We present experiments on data from a major quality rating company, and provide new methods for evaluating the solution. In addition, we provide an approach to use the reverse-engineered model to achieve a top ranked product in a cost-effective way.

Keywords Supervised ranking · Quality ratings · Discrete optimization · Reverse-engineering · Applications of machine learning

1 Introduction

Many organizations depend on the top ratings given to their products or services by quality rating companies. For instance, the reputations of undergraduate and graduate programs at colleges and universities depend heavily on their *U.S. News and World Report* rankings. Similarly, the mortgage industry hinges on the models of credit rating agencies like *Standard*

Editor: Thorsten Joachims.

A. Chang (✉)
Operations Research Center, Mass. Institute of Technology, Cambridge, MA 02139, USA
e-mail: aachang@mit.edu

C. Rudin
MIT Sloan School of Management, Mass. Institute of Technology, Cambridge, MA 02139, USA

M. Cavaretta · R. Thomas · G. Chou
Ford Motor Company, Dearborn, MI 48124, USA

& Poor's, Moody's, Dun & Bradstreet, and *Fitch Ratings*. Mutual funds rely on *Morningstar* and *Lipper* ratings. For electronics, rating companies include *CNET* and *PCMag*; and for vehicles, they include *What Car? J.D. Power, Edmunds, Kelley Blue Book*, and *Car and Driver*. Most of these rating companies use a formula to score products, and few of them make their complete rating formulas public. Moreover, the exact values of the input data to the formula are also often kept confidential. If organizations were able to recreate the formulas for quality rating models, they would better understand the standards by which their products were being judged, which would potentially allow them to produce better products. Furthermore, rating companies that are aware of reverse-engineering may be motivated to re-evaluate the accuracy and fairness of their formulas in representing the quality of products.

In this work, we introduce a method for reverse-engineering product ranking models, and apply it to over a decade's worth of data from a major quality rating company. Our method integrates knowledge about the way many such models are commonly constructed, which can be summarized as follows:

- *Point 1 (Linear scoring functions):* The rating company states publicly that its product rankings are based on real-valued scores given to each product, and that the score is a weighted linear combination of a known set of factors. The precise values for some factors can be obtained directly, but other factors have been discretized into a number of "stars" between 1 and 5 and are thus noisy versions of the true values. For example, the National Highway Traffic Safety Administration discretizes factors pertaining to vehicle safety ratings.
- *Point 2 (Category structure):* Products are organized into *categories*, and within each category there are one or more *subcategories*. For example, a computer rating company may have a laptop category with subcategories such as netbooks and tablets. Products within a category share the same scoring system, but the ranking of each product is with respect to its subcategory.
- *Point 3 (Ranks over scores):* The scores themselves are not as meaningful as the ranks, since consumers pay more attention to product rankings than to scores or to differences in score. Moreover, sometimes the scores are not available at all, and only the ranks are available.
- *Point 4 (Focus on top products):* Consumers generally focus on top-ranked products, so a model that can reproduce the top of each subcategory's ranked list accurately is more valuable than one that better reproduces the middle or bottom of the list.

Reverse-engineering product quality rankings is a new application for machine learning, and the algorithm we provide for this task matches the application in conforming to the points above. We use linear combinations of the same factors used by the rating company, and generate a separate model for each category, in accordance with Points 1 and 2. The reverse-engineered model for a given category is provided by a supervised ranking algorithm that uses discrete optimization to force the ranks produced by our algorithm to be similar to the ranks from the rating company; note that the algorithm reproduces the ranks, not the scores, as in Point 3. Specifically, the model is constructed to obey certain preference relationships in accordance with Point 4, that is, within each subcategory, the rankings of the rating companies' top-k products should match the top-k rankings from our model. When there are not enough data within a category to completely determine the ranking model for that category, our algorithm draws strength across categories, by using data from other categories as a type of regularization. Our experimental results on product quality ratings data indicate an advantage in sharing information across product categories, modeling ranks rather than scores, and using discrete optimization to maximize the exact rank statistic of interest rather than a convex proxy as is typical of conventional machine learning methods.

Note that even though Point 1 makes the assumption of known factors, it is also possible to use our method for problems in which the factors are unknown. As long as the factors in our model encompass the information used for the rating system, our algorithm can be applied regardless of whether or not the factors are precisely the same as those used by the rating company. For instance, a camera expert might know all of the potential camera characteristics that could contribute to camera quality, which we could then use as the factors in our model.

After the model has been reverse-engineered, we can use it to determine the most cost-effective way to increase product rankings, and we present discrete optimization algorithms for this task. These algorithms can be used independently of the reverse-engineering method. That is, if the reverse-engineered formula were obtained using a different method from ours, or if the formula were made public, we could still use these algorithms to cost-effectively increase a product's rank.

We describe related work in Sect. 2. In Sect. 3, we derive a ranking quality objective that encodes the preference relationships discussed above. In Sect. 4 we provide the machine learning algorithm, based on discrete optimization, that exactly maximizes the ranking quality objective. In Sect. 5, we establish new measures that can be used to evaluate the performance of our model. In Sect. 6, we derive several baseline algorithms for reverse-engineering, all involving convex optimization. Section 7 contains results from a proof-of-concept experiment, and Sect. 8 provides experimental results using rating data from a major quality rating company. Section 9 discusses the separate problem of how to cost-effectively increase the rank of a product. Finally, we conclude in Sect. 10. The main contributions of the paper are: the application of machine learning to reverse-engineering product quality rankings; our method of encoding the preference relationships in accordance with Points 1 through 4 above; using data from other product categories as regularization; the design of novel evaluation measures; and the mechanism to cost-effectively achieve a highly ranked product.

2 Related work

Reverse-engineering and approximation of rating models has been done in a number of industries, albeit not applied to rankings for consumer products with the category/subcategory structure. The related work we have found is published mostly within blogs. These works deal mostly with the problem of approximating the ranking function with a smaller number of variables, rather than using the exact factors in the rating company's formula. For instance, Chandler (2006) approximated the U.S. News and World Report Law School rankings using symbolic regression to obtain a formula with four factors, and another with seven factors; currently the formula for the law school rankings is completely public and based on survey results, but the approximated versions are much simpler. In the sports industry, there has been some work in reverse-engineering Elias Sports Bureau rankings, which are used to determine compensation for free agents (Bajek 2008). The search engine optimization (SEO) industry aims to be able to boost the search engine rank of a web page by figuring out which features have high influence in the ranking algorithm. For instance, Su et al. (2010) used a linear optimization model to approximate Google web page rankings. As a final example, Hammer et al. (2007) approximated credit rating models using Logical Analysis of Data. As far as we know, our work is the first to present a specialized machine learning algorithm to reverse-engineer product ratings.

If the ratings are accurate measures of quality, then making the ratings more transparent could have a uniformly positive impact: it would help companies to make better rated products, it would help consumers to have these higher quality products, and it would encourage rating companies to receive feedback as to whether their rating systems fairly represent quality. If the ratings are not accurate measures of quality, many problems could arise. Unethical manipulation of reverse-engineered credit rating models heavily contributed to the 2007–2010 financial crisis (Morgenson and Story 2010). These ratings permitted some companies to sell "junk bonds" with very high ratings. Rating companies were blamed for "performing the alchemy that converted the securities from F-rated to A-rated."[1]

Rating systems can also be arbitrary—even some well-established, heavily trusted rating systems can be inconsistent from product to product. There has been some controversy also over the Motion Picture Association of America movie rating system, discussed in the documentary "This Film Is Not Yet Rated."[2] The MPAA rating system sorts movies into categories based on how appropriate they are for certain audiences. The documentary demonstrates that the rating system was inconsistent between different types of films, and that the MPAA directly lied to the public regarding the way these ratings are constructed. This can be difficult for movie makers, whose profits may depend on getting an "R" rating rather than an "NC-17" rating, and it also causes problems for moviegoers, who want to know whether the movie is suitable for them.

Our reverse-engineering problem could potentially be useful in the area of conjoint analysis in marketing (Green et al. 2001). Conjoint analysts aim to model how a consumer chooses one brand over another, with the goal of learning which product characteristics are most important to consumers.

We have considered the reverse-engineering task as a problem of *supervised ranking*. Supervised ranking originated to handle problems that occur mainly in the information retrieval domain (see, for instance, the LETOR compilation of works[3]). The vast majority of work on supervised ranking considers problems that are specific to information retrieval (e.g., Cao et al. 2007; Matveeva et al. 2006; Lafferty and Zhai 2001; Li et al. 2007) or give insight into how to approximately solve versions of extremely large ranking problems quickly (Tsochantaridis et al. 2005; Freund et al. 2003; Cossock and Zhang 2006; Joachims 2002; Burges et al. 2006; Xu et al. 2008; Le and Smola 2007; Ferri et al. 2002; Ataman et al. 2006). For the task of reverse-engineering ranking models, fast computational speed is not essential, and the extra time needed to compute a better solution is worthwhile. This, coupled with the fact that the size of the dataset is not extremely large, permits us to use mixed-integer optimization (MIO). MIO preserves our encoding of exactly the desired preference structure, where we have incorporated membership into categories and subcategories. If we remove regularization and do not concentrate on the top ranks, then the problem is a generalization of Area Under the Curve (AUC) maximization (Freund et al. 2003; Joachims 2002). Most works on AUC maximization use a smoothed approximation of the 0-1 loss within the AUC. If we were to use a smoothed approximation for the reverse-engineering problem, it is possible that the algorithm would miss the best solutions to the 0-1 optimization problem. The ℓ_pRE relaxation algorithm we introduce in Sect. 6 is one such approximation. The work of Bertsimas et al. (2010, 2011) also discusses in depth the benefits of exact solutions over relaxations.

[1] http://www.bloomberg.com/apps/news?sid=ah839IWTLP9s&pid=newsarchive by Elliot Blair Smith, September 24, 2008.

[2] Directed by Kirby Dick, 2006.

[3] http://research.microsoft.com/en-us/um/beijing/projects/letor/paper.aspx.

Clearly, reverse-engineered ranking models can affect design decisions in a variety of applications. To the best of our knowledge, our work is the first to show the most cost-effective way to increase the rank of a new product.

3 Encoding preferences for quality ranking data

We derive a specialized rank statistic that serves as our objective for reverse-engineering. Maximizing this objective yields estimates of the weights on each of the factors in the rating company's model. Our starting point is the case of one category with one subcategory, that is, there is only a single ranked list. Then, we generalize this statistic to handle multiple categories and subcategories. Our method can be used to reverse-engineer quality rankings whether or not the underlying scores are made available; we need only to know the ranks.

3.1 One category, one subcategory

Let n denote the number of products to be ranked. We represent product i by a vector of d factors $x_i \in \mathcal{X}$, where $\mathcal{X} \subset \mathcal{R}^d$. The rating company gives a score $\zeta_i \in \mathcal{R}$ to each product i, which translates into a rank. Higher scores imply higher ranks, so that a product with rank 0 is at the bottom of the list with the lowest quality. For all pairs of products, let the preference function $\pi : \mathcal{X} \times \mathcal{X} \to \{0, 1\}$ capture the true pairwise preferences according to the scores ζ_i. That is, let:

$$\pi(x_i, x_k) := \pi_{ik} := \mathbf{1}_{[\zeta_i > \zeta_k]},$$

where $\mathbf{1}_q$ is the indicator function that equals 1 if condition q holds and 0 otherwise. In other words, if product i is ranked higher than product k by the rating company, then π_{ik} is 1. Even if the ζ_i's are not available, we can derive the π_{ik}'s because we know which products are ranked higher than which other products. Our goal is to generate a scoring function $f : \mathcal{X} \to \mathcal{R}$ that assigns real-valued scores $f(x_i)$ to each product x_i such that the π_{ik} values match as closely as possible with our model preferences $\mathbf{1}_{[f(x_i) > f(x_k)]}$.

Let $\Pi = \sum_{i=1}^{n} \sum_{k=1}^{n} \pi_{ik}$. We first consider a rank statistic that generalizes the area under the ROC curve (AUC):

$$\mathrm{AUC}_\pi(f) := \frac{1}{\Pi} \sum_{i=1}^{n} \sum_{k=1}^{n} \pi_{ik} \mathbf{1}_{[f(x_i) > f(x_k)]}. \tag{1}$$

This statistic is related to the disagreement measure introduced by Freund et al. (2003), as well as Kendall's τ coefficient (Kendall 1938). That is, in the absence of ties, the disagreement measure is $1 - \mathrm{AUC}_\pi(f)$ and Kendall's τ is $2\mathrm{AUC}_\pi(f) - 1$. The highest possible value of $\mathrm{AUC}_\pi(f)$ is 1, which is achieved if the scoring function f satisfies $f(x_i) > f(x_k)$ for all pairs (x_i, x_k) such that $\pi_{ik} = 1$.

$\mathrm{AUC}_\pi(f)$ does not put any emphasis on the top of the ranked list; a product at the bottom of the ranked list can contribute the same amount to $\mathrm{AUC}_\pi(f)$ as a product at the top. However, as noted in Point 4 in Sect. 1, it is often more important to accurately reproduce rankings at the top of the list than in the middle or at the bottom. Suppose we want to concentrate on the top $\bar{T}$ products within the subcategory. In particular, we want to weigh the top $\bar{T}$ products $1 + \theta$ times more than the rest of the list, where $\theta \geq 0$. To do this, we

first define the rank of product i, with respect to scoring function f, to be the number of products it is scored strictly above:

$$\text{rank}_f(x_i) := \sum_{k=1}^{n} \mathbf{1}_{[f(x_i) > f(x_k)]}.$$

The top $\bar{T}$ products have rank at least $T := n - \bar{T}$. For example, if $n = 10$, then assuming no ties in rank, the top $\bar{T} = 4$ products have ranks at least $T = 6$, that is, their ranks are 6, 7, 8, and 9. We consider the objective function:

$$\text{AUC}_{\pi}^{\text{top}}(f) := \frac{1}{\Pi(\theta)} \sum_{i=1}^{n} \sum_{k=1}^{n} \pi_{ik} \mathbf{1}_{[f(x_i) > f(x_k)]} (1 + \theta \mathbf{1}_{[\text{rank}_f(x_i) \geq T]}),$$

where we normalize by

$$\Pi(\theta) = \sum_{i=1}^{n} \sum_{k=1}^{n} \pi_{ik} (1 + \theta \mathbf{1}_{[\sum_{k=1}^{n} \pi_{ik} \geq T]}).$$

Note that $\text{AUC}_{\pi}^{\text{top}}(f)$ varies between 0 and 1 since the largest possible value of the summation in $\text{AUC}_{\pi}^{\text{top}}(f)$ is $\Pi(\theta)$, which is achieved if f ranks all pairs (x_i, x_k) correctly. Each pair of products (x_i, x_k) contributes $\frac{1}{\Pi(\theta)} \pi_{ik} \mathbf{1}_{[f(x_i) > f(x_k)]} (1 + \theta)$ to the objective if the rank of x_i is at least T, and contributes $\frac{1}{\Pi(\theta)} \pi_{ik} \mathbf{1}_{[f(x_i) > f(x_k)]}$ otherwise. If either $\theta = 0$ or $T = 0$, then maximizing this objective is equivalent to maximizing $\text{AUC}_{\pi}(f)$, which does not focus at the top.

3.2 Multiple categories and subcategories

We assume from Sect. 1 that different categories have different ranking models. Even so, these models may be similar enough that knowledge obtained from other categories can be used to "borrow strength" when there are limited data in the category of interest. Thus, as we derive the objective for reverse-engineering the model f for one prespecified category, we use data from all of its subcategories as well as from the subcategories in other categories.

Let S_{sub} be the set of all subcategories across all categories, including the category of interest, and let there be n_s products in subcategory s. Similar to our previous notation, $x_i^s \in \mathcal{R}^d$ represents product i in subcategory s, $\zeta_i^s \in \mathcal{R}$ is the score assigned to product i in subcategory s, and π_{ik}^s is 1 if $\zeta_i^s > \zeta_k^s$ and is 0 otherwise. The threshold T_s defines the top of the list for subcategory s.

The general objective we will optimize is a weighted sum of the $\text{AUC}_{\pi}^{\text{top}}(f)$ values over all subcategories:

$$\begin{aligned} &\text{AUC}_{\pi}^{\text{top,sub}}(\theta, C)(f) \\ &\quad = \sum_{s \in S_{\text{sub}}} \frac{C_s}{\Pi_s(\theta)} \sum_{i=1}^{n_s} \sum_{k=1}^{n_s} \pi_{ik}^s \mathbf{1}_{[f(x_i^s) > f(x_k^s)]} (1 + \theta \mathbf{1}_{[\text{rank}_f^s(x_i^s) \geq T_s]}), \end{aligned} \tag{2}$$

where

$$\text{rank}_f^s(x_i^s) = \sum_{k=1}^{n_s} \mathbf{1}_{[f(x_i^s) > f(x_k^s)]}. \tag{3}$$

The normalization constants are

$$\Pi_s(\theta) = \sum_{r \in \text{cat}(s)} \sum_{i=1}^{n_r} \sum_{k=1}^{n_r} \pi_{ik}^r (1 + \theta \mathbf{1}_{[\sum_{k=1}^{n_r} \pi_{ik}^r \geq T_r]}), \tag{4}$$

where $\text{cat}(s)$ denotes the category to which subcategory s belongs. The values C_s determine how much influence each subcategory has on the model. It is logical in general for C_s to be the same for all subcategories within a certain category. If there is a sufficient number of rated products in the category of interest, relative to the total number d of factors, then we can train the model with only these data. In that case, we would set $C_s = 1$ for subcategories within the category of interest and $C_s = 0$ for subcategories in all other categories. On the other hand, if the number of products in the category of interest is too small to permit the model to generalize, then we can regularize by setting $C_s \in (0, 1]$ for subcategories of other categories, choosing the values of C_s by cross-validation or heuristics.

Note that $\Pi_s(\theta)$ is the same for all subcategories s within the same category, instead of being proportional to the size of the subcategory. This is because we want each pair of products within the same category to have the same influence on the objective function. Consider if the normalization constants were alternatively

$$\Pi_s(\theta) = \sum_{i=1}^{n_s} \sum_{k=1}^{n_s} \pi_{ik}^s (1 + \theta \mathbf{1}_{[\sum_{k=1}^{n_s} \pi_{ik}^s \geq T_s]}).$$

Then for a particular category, there may be subcategories with large values of $\Pi_s(\theta)$ and others with small values. But in this case, assuming C_s is the same for all subcategories in this category, a misranked pair lowers the objective by much more in the subcategories with small $\Pi_s(\theta)$ values than with large values (since $\Pi_s(\theta)$ is in the denominator). Thus, in some sense, normalizing this way puts more weight on accurately ranking within the smaller subcategories. To avoid this issue, we use (4) to normalize. Conventional ranking methods do not address the subcategory/category structure of our product ranking problem in this manner; in fact it can be difficult to take the normalization into account accurately if the learning algorithm is limited to convex optimization. We show in Sect. 4 how our algorithm incorporates, in an exact way, this form of normalization.

We assume a linear form for the model, in accordance with Point 1 in Sect. 1. That is, the scoring function has the form $f(x) = w^T x$, so that $w \in \mathcal{R}^d$ is a vector of variables in our formulation, and the objective in (2) is a function of w. Note that we can capture relatively complex nonlinear rating systems using a linear model with nonlinear factors. For instance, we could introduce extra factors to accommodate "necessity" constraints, where products that do not have a certain property will always get a low score. To do this, we would add a binary factor to the model that is 1 if the product does not possess the property, and the learning algorithm should discover a large negative weight for that factor.

4 Optimization

We now provide an algorithm to reverse-engineer quality rankings that exactly maximizes (2). The algorithm is called MIO-RE—Mixed Integer Optimization for Reverse-Engineering, and expands on a technique due to Bertsimas et al. (2010, 2011) for supervised ranking in machine learning. In this work, the authors develop a type of approach with an advantage over other machine learning techniques in that it exactly optimizes the objective.

This advantage is counterbalanced by a sacrifice in computational speed, but for the rating problem, new data come out occasionally (e.g., yearly, monthly, weekly) whereas the computation time is generally on the order of hours, depending on the number of products in the training data and the number of factors. In this case, the extra computation time needed to produce a better solution is worthwhile.

In MIO, it is important to note that even though there are often various correct formulations to solve the same problem, not all valid formulations are equally strong. In fact, the ability to solve an MIO problem depends critically on the choice of formulation (see Bertsimas and Weismantel 2005, for details). This is not true of linear optimization, where a good formulation is simply one that correctly captures the model and is small in terms of the number of variables and constraints. In linear optimization, the choice of formulation is not crucial for solving a problem. However, when there are integer variables, it is typical to reformulate multiple times to achieve the best model. Essentially, a formulation is stronger if it cuts off extra unnecessary parts of the region of feasible solutions. Below we present a strong MIO formulation that we have found to work well empirically, and we discuss the logic behind its derivation.

Modern solvers typically produce a bound (upper for maximization problems, lower for minimization problems) as they search for better integer feasible solutions, and when the bound matches the objective value of an integer solution, the solution has reached provable optimality. However, it is common for a solver to find an optimal solution relatively quickly, but to take much longer in proving optimality, that is, in bringing the bound closer to the optimal objective value. See Bertsimas et al. (2011) for an introduction to MIO that discusses in particular the strength of a formulation and also the progress in MIO technology over the last few decades. Due to advances in both hardware and MIO algorithms, computational speed has been increasing exponentially, allowing us today to solve large scale MIO problems that would have been impossible only a few years ago. MIO will be progressively more powerful as this exponential trend continues.

In Sects. 7 and 8, our experimental results show that our MIO algorithm performs well on both training and test data. Considering generalization bounds from statistical learning theory, there are two ways to achieve better test performance: one is to decrease the training error, and the other is to decrease the complexity term to prevent overfitting. Using MIO, we can decrease the training error, and we control the complexity by using regularization across categories as shown in Sect. 3.2.

4.1 Model for reverse-engineering

The variable v_i^s represents the model's score $w^T x_i^s$ of product i in subcategory s, and the binary variable z_{ik}^s captures the decision $\mathbf{1}_{[v_i^s > v_k^s]}$, as in (3). The strict inequality is numerically defined using a small positive constant ε, that is:

$$z_{ik}^s = \mathbf{1}_{[v_i^s - v_k^s \geq \varepsilon]}. \tag{5}$$

Thus $\text{rank}_f^s(x_i^s) = \sum_{k=1}^{n_s} z_{ik}^s$. To keep track of which products are in the top, we want the binary variable t_i^s be 1 only if $\text{rank}_f^s(x_i^s)$ is at least T_s:

$$t_i^s = \mathbf{1}_{[\sum_{k=1}^{n_s} z_{ik}^s \geq T_s]}. \tag{6}$$

Also, we want the binary variable u_{ik}^s to be 1 only if both $v_i^s - v_k^s \geq \varepsilon$ and $\text{rank}_f^s(x_i^s) \geq T_s$, which is equivalent to:

$$u_{ik}^s = \min\left\{z_{ik}^s, t_i^s\right\}. \tag{7}$$

Using these decision variables, we can rewrite the objective (2) as

$$\sum_{s\in S_{\text{sub}}} \frac{C_s}{\Pi_s(\theta)} \sum_{i=1}^{n_s}\sum_{k=1}^{n_s} \pi_{ik}^s \left(z_{ik}^s + \theta u_{ik}^s\right). \tag{8}$$

To capture (5) through (7), we use the following constraints:

$$z_{ik}^s \le v_i^s - v_k^s + 1 - \varepsilon, \tag{9}$$

$$T_s t_i^s \le \sum_{k=1}^{n_s} z_{ik}^s, \tag{10}$$

$$u_{ik}^s \le z_{ik}^s, \tag{11}$$

$$u_{ik}^s \le t_i^s. \tag{12}$$

If $v_i^s - v_k^s \ge \varepsilon$, then the right-hand side of (9) is at least 1, so the solver sets $z_{ik}^s = 1$ because we are maximizing z_{ik}^s. Otherwise, the right-hand side is strictly less than 1, so the solver sets $z_{ik}^s = 0$. Similarly, if $\sum_{k=1}^{n_s} z_{ik}^s \ge T_s$, then (10) implies $t_i^s = 1$; note that since we are maximizing u_{ik}^s in (8), we are also automatically maximizing t_i^s because of (12). And if both $v_i^s - v_k^s \ge \varepsilon$ and $\text{rank}_f^s(x_i^s) \ge T_s$, then $z_{ik}^s = t_i^s = 1$, so (11) and (12) imply $u_{ik}^s = 1$. We do not need to explicitly specify u_{ik}^s as a binary variable because u_{ik}^s is the minimum of two binary variables; if either z_{ik}^s or t_i^s is 0, then u_{ik}^s is 0, and otherwise it is 1. Here is the MIO formulation that maximizes (2):

$$\begin{aligned}
\max_{w,v,z,t,u} \quad & \sum_{s\in S_{\text{sub}}} \frac{C_s}{\Pi_s(\theta)} \sum_{i=1}^{n_s}\sum_{k=1}^{n_s} \pi_{ik}^s \left(z_{ik}^s + \theta u_{ik}^s\right) \\
\text{s.t.} \quad & v_i^s = w^T x_i^s, \quad \forall s, i, \\
& z_{ik}^s \le v_i^s - v_k^s + 1 - \varepsilon, \quad \forall s, i, k, \\
& T_s t_i^s \le \sum_{k=1}^{n_s} z_{ik}^s, \quad \forall s, i, k, \\
& u_{ik}^s \le z_{ik}^s, \quad \forall s, i, k, \\
& u_{ik}^s \le t_i^s, \quad \forall s, i, k, \\
& 0 \le w_j, u_{ik}^s \le 1, \quad \forall j, s, i, k, \\
& z_{ik}^s, t_i^s \in \{0, 1\}, \quad \forall s, i, k.
\end{aligned} \tag{13}$$

We enforce that the weights $\{w_j\}_{j=1}^d$ are nonnegative, in accordance with our knowledge of how most quality ratings are constructed. If there is a case in which a factor is negatively correlated with rank, then we would simply use the negative of the factor, so that the corresponding weight would be positive. Also, if w^* maximizes (2), then so does γw^*, for any constant $\gamma > 0$; thus we can constrain each w_j to be in the interval $[0, 1]$ without loss of generality. The primary purpose of this constraint is to reduce the size of the region of feasible solutions, which is intended to speed up the computation. There is a single parameter $\varepsilon > 0$ that the user specifies. Since increasing ε tends to increase runtimes, we choose ε to be just large enough to be recognized as nonzero by the solver.

After the optimization problem (13) is solved for our category of interest, we use the maximizing weights w^* to determine the score $f(x) = w^{*T} x$ of a new product x within the same category.

5 Evaluation metrics

In the case of our rating data, one goal is to predict, for instance, whether a new product that has not yet been rated would be among the top-k products that have already been rated. That is, the training data are included in the assessment of test performance. This type of evaluation is contrary to common machine learning practice in which evaluations on the training and test sets are separate, and thus it is not immediately clear how these evaluations should be performed.

In this section, we define three measures that are useful for supervised ranking problems in which test predictions are gauged relative to the training set. These measures are intuitive, and more closely represent how most industries would evaluate ranking quality than conventional rank statistics. The measures are first computed separately for each subcategory and then aggregated over the subcategories to produce a concise result. We focus on the top $\bar{T}_s$ products in subcategory s, and use the following notation, where $f(x) = w^T x$ is a given scoring function.

ζ_i^s	=	true score for product x_i^s (training or test)
ζ^s	=	true score of product in position $\bar{T}_s$ within the training set, where products are ranked according to true scores ζ_i^s
f_w^s	=	model score of product in position $\bar{T}_s$ within the training set, where products are ranked according to model scores $w^T x_i^s$
S_{train}^s	=	$\{i : \text{product } x_i^s \text{ is in the training set}\}$
S_{test}^s	=	$\{j : \text{product } x_j^s \text{ is in the test set}\}$
S_{all}^s	=	$S_{\text{train}}^s \cup S_{\text{test}}^s$
$S_{\text{train,top}}^s$	=	$\{i : i \in S_{\text{train}}^s \text{ and } \zeta_i^s \geq \zeta^s\}$
$S_{\text{test,top}}^s$	=	$\{j : j \in S_{\text{test}}^s \text{ and } \zeta_j^s \geq \zeta^s\}$
$S_{\text{all,top}}^s$	=	$S_{\text{train,top}}^s \cup S_{\text{test,top}}^s$

Note that ζ^s and f_w^s are computed from only the training data.

Measure 1: fraction of correctly ranked pairs among top of ranked list

This is the most useful and important of the three measures because it specifically captures ranking quality at the top of the list. Using the same notation as in (13), let $\pi_{ik} = 1$ if $\zeta_i^s > \zeta_k^s$ and 0 otherwise, and $z_{ik} = 1$ if $w^T x_i > w^T x_k$ and 0 otherwise. The evaluation measures for the training and test data are:

$$\text{M1}_{\text{train}}(s) = \frac{\sum_{i,k \in S_{\text{train,top}}^s} \pi_{ik} z_{ik}}{\sum_{i,k \in S_{\text{train,top}}^s} \pi_{ik}},$$

$$\text{M1}_{\text{test}}(s) = \frac{\sum_{i,k \in S_{\text{all,top}}^s} \pi_{ik} z_{ik} - \sum_{i,k \in S_{\text{train,top}}^s} \pi_{ik} z_{ik}}{\sum_{i,k \in S_{\text{all,top}}^s} \pi_{ik} - \sum_{i,k \in S_{\text{train,top}}^s} \pi_{ik}}.$$

The M1 metric does not require the actual values of the true scores ζ_i^s; it suffices to know the pairwise preferences π_{ik}. Note that $\text{M1}_{\text{test}}(s)$ is the fraction of correctly ranked pairs among both training and test products, excluding pairs for which both products are in the training set.

Measure 2: fraction of correctly ranked pairs over entire ranked list

This measure is similar to Measure 1, except that instead of considering only the top of the ranked list, it considers the entire list.

$$\mathrm{M2}_{\mathrm{train}}(s) = \frac{\sum_{i,k\in S^s_{\mathrm{train}}} \pi_{ik} z_{ik}}{\sum_{i,k\in S^s_{\mathrm{train}}} \pi_{ik}},$$

$$\mathrm{M2}_{\mathrm{test}}(s) = \frac{\sum_{i,k\in S^s_{\mathrm{all}}} \pi_{ik} z_{ik} - \sum_{i,k\in S^s_{\mathrm{train}}} \pi_{ik} z_{ik}}{\sum_{i,k\in S^s_{all}} \pi_{ik} - \sum_{i,k\in S^s_{\mathrm{train}}} \pi_{ik}}.$$

Note that $\mathrm{M2}_{\mathrm{train}}$ is the same as AUC_π in (1).

Measure 3: fraction of correctly classified products

This evaluation metric is the fraction of products that are correctly classified in terms of being among the top of the list:

$$\mathrm{M3}_{\mathrm{train}}(s) = \frac{1}{|S^s_{\mathrm{train}}|} \sum_{i\in S^s_{\mathrm{train}}} (\mathbf{1}_{[\zeta^s_i \geq \zeta^s \text{ and } w^T x_i \geq f^s_w]} + \mathbf{1}_{[\zeta^s_i < \zeta^s \text{ and } w^T x_i < f^s_w]}),$$

$$\mathrm{M3}_{\mathrm{test}}(s) = \frac{1}{|S^s_{\mathrm{test}}|} \sum_{j\in S^s_{\mathrm{test}}} (\mathbf{1}_{[\zeta^s_j \geq \zeta^s \text{ and } w^T x_j \geq f^s_w]} + \mathbf{1}_{[\zeta^s_j < \zeta^s \text{ and } w^T x_j < f^s_w]}).$$

Although $\mathrm{M3}_{\mathrm{test}}(s)$ measures quality on the test set, the values ζ^s and f^s_w depend on the true scores and model scores from the training set. If the true scores ζ^s_i are not available, then it suffices to know the rank of each product relative to the product in position $\bar{T}_s$ in the training set in order to compute this metric.

Aggregation of measures

To produce a single numerical evaluation for each of the three measures, we aggregate by taking a weighted sum of the measures over subcategories in a given category, where the weights are proportional to the sizes of the subcategories. The three evaluation measures defined above all have the form:

$$\mathrm{M}(s) = \frac{\mathrm{numer}(s)}{\mathrm{denom}(s)}.$$

The version of evaluation measure M aggregated over subcategories for either the training or test set is:

$$\mathrm{M} = \frac{\sum_s \mathrm{numer}(s)}{\sum_s \mathrm{denom}(s)} = \frac{\sum_s \mathrm{denom}(s)\mathrm{M}(s)}{\sum_s \mathrm{denom}(s)}.$$

6 Other methods for reverse-engineering

We developed several baseline algorithms for our experiments that also encode the points in the introduction. The first set of methods are based on least squares regression, and the second set are convex relaxations of the MIO method. These algorithms could be themselves useful, for instance, if a fast convex algorithm is required.

6.1 Least squares methods for reverse-engineering

The organization that provides our rating data currently uses a proprietary method to reverse-engineer the ranking model, the core of which is very similar to least squares regression on the scores. If the scores were not available—for instance, when working with data from a different rating company—the organization would conceivably use least squares regression on the ranks. Thus, our baselines are variations on least squares regression, minimizing:

$$\sum_{s\in S_{\text{sub}}} \frac{C_s}{N_s} \sum_{i=1}^{n_s} \left(y_i^s - \left(w_0 + w^T x_i^s\right)\right)^2,$$

where N_s is the number of products in the category to which subcategory s belongs:

$$N_s = \sum_{r\in \text{cat}(s)} n_r,$$

and y_i^s can be one of three quantities:

1. the true score ζ_i^s for product x_i^s (method LS1),
2. the rank over all training products, that is, the number of training products that are within subcategories r such that $C_r > 0$ and are ranked strictly below x_i^s according to the true scores ζ_i^s (method LS2),
3. the rank within the subcategory, that is, the number of training products in the same subcategory as x_i^s that are ranked strictly below x_i^s according to the true scores ζ_i^s (method LS3).

6.2 The ℓ_p reverse-engineering algorithm

As another point of comparison, we introduce a new method called "ℓ_p Reverse-Engineering" (ℓ_pRE) that generalizes the P-Norm Push algorithm for supervised ranking, developed by Rudin (2009). This algorithm minimizes an objective with two terms, one that "pushes" low-quality products to the bottom of the list, and another that "pulls" high-quality products to the top. To derive this algorithm, we first consider the following loss function:

$$\text{Loss}_{s,p,\text{low},0-1}(f) := \left(\sum_{k=1}^{n_s}\left(\sum_{i=1}^{n_s} \pi_{ik}^s \mathbf{1}_{[f(x_i^s)\le f(x_k^s)]}\right)^p\right)^{1/p}.$$

In order to interpret $\text{Loss}_{s,p,\text{low},0-1}(f)$, consider that $\sum_{i=1}^{n_s} \pi_{ik}^s \mathbf{1}_{[f(x_i^s)\le f(x_k^s)]}$ is the number of products i that should be ranked higher than k (that is, $\pi_{ik}^s = 1$), but are ranked lower by f (that is, $\mathbf{1}_{[f(x_i^s)\le f(x_k^s)]}$). This quantity is large when k is a low-quality product that is near the top of the ranked list. In other words, the largest terms in the sum $\sum_{k=1}^{n_s}(\sum_{i=1}^{n_s} \pi_{ik}^s \mathbf{1}_{[f(x_i^s)\le f(x_k^s)]})^p$ correspond to low quality products that are highly ranked. Thus, minimizing $\text{Loss}_{s,p,\text{low},0-1}(f)$ tends to "push" low-quality products towards the bottom of the list.

Instead of minimizing $\text{Loss}_{s,p,\text{low},0-1}(f)$ directly, we can minimize the following convex upper bound:

$$\text{Loss}_{s,p,\text{low}}(f) := \left(\sum_{k=1}^{n_s}\left(\sum_{i=1}^{n_s} \pi_{ik}^s e^{-(f(x_i^s)-f(x_k^s))}\right)^p\right)^{1/p}.$$

We reverse the sums over i and k to define another quantity:

$$\text{Loss}_{s,p,\text{high}}(f) := \left(\sum_{i=1}^{n_s}\left(\sum_{k=1}^{n_s}\pi_{ik}^{s}e^{-(f(x_i^s)-f(x_k^s))}\right)^{p}\right)^{1/p}.$$

Minimizing $\text{Loss}_{s,p,\text{high}}(f)$ tends to "pull" high-quality products towards the top of the list. The ℓ_pRE method uses both $\text{Loss}_{s,p,\text{low}}(f)$ and $\text{Loss}_{s,p,\text{high}}(f)$. The loss function minimized by ℓ_pRE is:

$$\sum_{s\in S_{\text{sub}}}\frac{C_s}{N_{s,p}}\big(\text{Loss}_{s,p,\text{low}}(f)+C_{\text{high}}\cdot\text{Loss}_{s,p,\text{high}}(f)\big),$$

where the normalization factor $N_{s,p}$ is:

$$N_{s,p}=\sum_{r\in\text{cat}(s)}\left(\left(\sum_{k=1}^{n_r}\left(\sum_{i=1}^{n_r}\pi_{ik}^{r}\right)^{p}\right)^{1/p}+C_{\text{high}}\left(\sum_{i=1}^{n_r}\left(\sum_{k=1}^{n_r}\pi_{ik}^{r}\right)^{p}\right)^{1/p}\right),$$

and C_s and C_{high} are user-specified parameters that control the relative importance of each subcategory, and the importance of $\text{Loss}_{s,p,\text{high}}(f)$ relative to $\text{Loss}_{s,p,\text{low}}(f)$ respectively. We use $p=1$ and $p=2$, and denote the corresponding methods ℓ_1RE and ℓ_2RE respectively.

7 Proof of concept

As a preliminary experiment, we tested the methods using an artificial dataset that we have made publicly available.[4] Figure 1 shows for each of the five factors of this dataset, a scatterplot of the factor values versus the scores. The sixth plot in the figure shows all five factors versus the scores in the same window. For each factor, there is one set of products for which there is perfect correlation between the factor values and scores, another set for which there is perfect anti-correlation, and the remainder for which the factor value is constant. By constructing the dataset in this manner, we expect there to be significant variation in the ranking performance of the different methods.

There is only one category with one subcategory. There are 200 products total, and we randomly divided the data into 100 products for training and 100 products for testing. We tested five methods: LS1, LS2, ℓ_1RE, ℓ_2RE, and MIO-RE; LS3 is equivalent to LS2 since there is only one subcategory.[5] We ran the methods for three cases: concentrating on the top 60, the top 45, and the top 25, that is, $T=40$, $T=55$, and $T=75$ respectively. We ran ℓ_1RE with $C_{\text{high}}=0$; ℓ_2RE with $C_{\text{high}}=0, 0.5$, and 1; and MIO-RE with $\theta=9$. MIO-RE found the final solutions within three minutes for each case, and the other methods ran within seconds. Tables 1, 2, and 3 show the results. The highest training and test measures across the methods are highlighted in bold. LS1, ℓ_1RE, and ℓ_2RE (with $C_{\text{high}}=0, 0.5$, and 1) always produced the same values for the three evaluation measures.

[4] Dataset available at: http://web.mit.edu/rudin/www/ReverseEngineering_Flex_Data.csv.

[5] All least-squares methods were implemented using R 2.8.1, and all ℓ_pRE methods were implemented using MATLAB 7.8.0, on a computer with an Intel Core 2 Duo 2 GHz processor with 1.98 GB of RAM. MIO-RE was implemented using ILOG AMPL 11.210 with the Gurobi 3.0.0 solver on a computer powered by two Intel quad core Xeon E5440 2.83 GHz processors with 32 GB of RAM. We always used $\varepsilon=10^{-6}$ for MIO-RE.

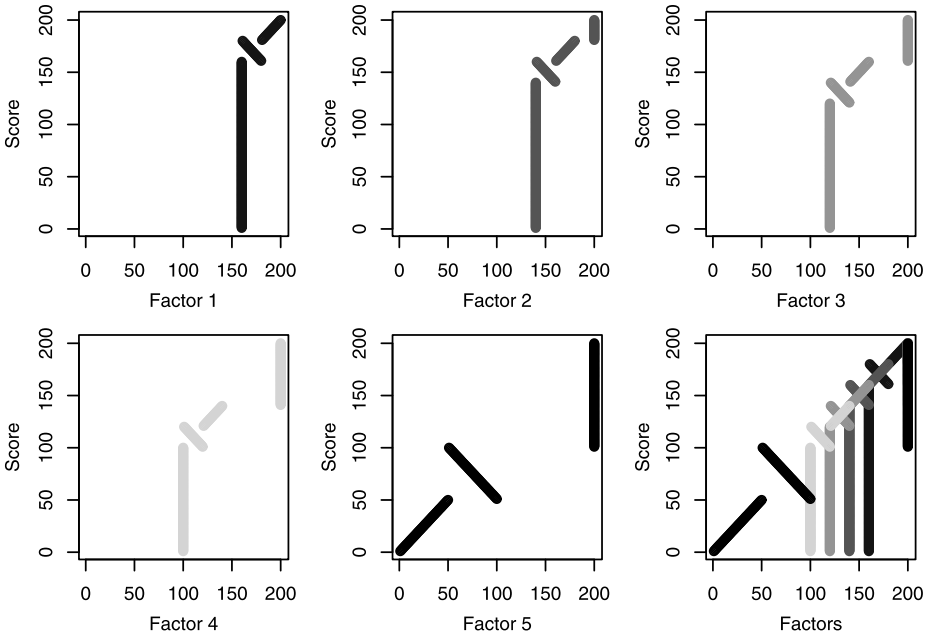

Fig. 1 Factor values vs. scores for artificial dataset

Table 1 Training and test values for M1, M2, and M3 on artificial dataset (top 60)

Algorithm		M1	M2	M3
LS1, ℓ_1RE, ℓ_2RE	train	0.878	0.912	**0.780**
	test	0.892	0.909	**0.770**
LS2	train	0.909	0.923	**0.780**
	test	0.915	0.918	**0.770**
MIO-RE	train	**0.925**	**0.928**	**0.780**
	test	**0.943**	**0.929**	**0.770**

Table 2 Training and test values for M1, M2, and M3 on artificial dataset (top 45)

Algorithm		M1	M2	M3
LS1, ℓ_1RE, ℓ_2RE	train	0.880	0.912	**0.920**
	test	0.898	0.909	**0.930**
LS2	train	0.935	0.923	**0.920**
	test	0.942	0.918	**0.930**
MIO-RE	train	**0.964**	**0.928**	**0.920**
	test	**0.994**	**0.929**	**0.930**

The methods all performed similarly according to the classification measure M3. MIO-RE had a significant advantage with respect to M2, no matter the definition we used for top of the list (top 60 in Table 1, top 45 in Table 2, or top 25 in Table 3). For M1, MIO-RE

Table 3 Training and test values for M1, M2, and M3 on artificial dataset (top 25)

Algorithm		M1	M2	M3
LS1, ℓ_1RE, ℓ_2RE	train	0.907	0.912	**1.000**
	test	0.899	0.909	0.980
LS2	train	0.907	0.923	**1.000**
	test	0.899	0.918	0.980
MIO-RE	train	**1.000**	**0.928**	**1.000**
	test	**1.000**	**0.929**	**1.000**

performed substantially better than the others, and its advantage over the other methods was more pronounced as the evaluation measure concentrated more on the top of the list. One can see this by comparing the M1 column in Tables 1, 2, and 3. In Table 3, MIO-RE performed better than the other methods by 10.3 % on training and 11.3 % on testing. Using exact optimization rather than approximations, the MIO-RE method was able to find solutions that none of the other methods could find. This study demonstrates the potential of MIO-RE to substantially outperform other methods.

8 Experiments on rating data

For our main experiments, the dataset contains approximately a decade's worth of rating data from a major rating company, compiled by an organization that is aiming to reverse-engineer the ranking model. The values for most of the factors are discretized versions of the true values, that is, they have been rounded to the nearest integer. The rating company periodically makes ratings for new products available, and our goal is to predict, with respect to the products that are already rated: where each new product is within the top-k (M1), where it is in the full list, even if not in the top-k (M2), and whether each new product falls within the top-k (M3). We generate a scoring function for one category, "Category A," regularizing with data from "Category B." Category A has eight subcategories with a current total of 209 products, and Category B has eight subcategories with a total of 212 products. There are 19 factors.

This dataset is small and thus challenging to deal with from a machine learning perspective. The small size of the training sets causes problems with accurate reverse-engineering. The small size of the test sets causes problems with evaluating generalization ability. That is, for all algorithms, the variance of the test evaluation measures is high compared to the difference in training performance, so it is difficult to evaluate which algorithm is better in a robust way. The worst performing algorithm in training sometimes has the best test performance, and vice versa. What we aim to determine is whether MIO-RE has consistently good performance, as compared with other algorithms that sometimes perform very poorly.

8.1 Experimental setup

For this set of experiments, we divided the data for Category A into four folds, and used each fold in turn as the test set. The first fold had 53 products, and the other three folds each had 52 products. Our experiment was as follows, where M1, M2, and M3 refer to the three aggregate evaluation measures, computed using just data from Category A and not Category B, though data from both categories were used for training:

Table 4 Parameter values tested for each algorithm

Algorithm	Parameter1	Parameter2
LS1	$C = 0$, 0.1, or 0.2	
LS2	$C = 0$, 0.1, or 0.2	
LS3	$C = 0$, 0.025, or 0.05	
ℓ_1RE	$C = 0$ or 0.1	$C_{\text{high}} = 0$
ℓ_2RE	$C = 0$ or 0.1	$C_{\text{high}} = 0$, 0.5, or 1
MIO-RE	$C = 0$ or 0.5	$\theta = 0$ or 9

1. For each set of parameters, perform three-fold cross-validation using the first three folds as follows:
   a. Train using folds 1 and 2, and Category B, and validate using fold 3. Compute M1, M2, and M3 for training and validation.
   b. Train using folds 1 and 3, and Category B, and validate using fold 2. Compute M1, M2, and M3 for training and validation.
   c. Train using folds 2 and 3, and Category B, and validate using fold 1. Compute M1, M2, and M3 for training and validation.
   d. Compute the average over the three folds of the training and validation values for each of M1, M2, and M3.

   Note that when we compute M1, M2, and M3 on validation data, this also takes into account the training data, as in Sect. 5.
2. Sum the three average validation measures, and choose the parameters corresponding to the largest sum.
3. Train using folds 1, 2, and 3, and Category B, together with the parameters chosen in the previous step, and test using fold 4. Compute M1, M2, and M3 for training and testing.
4. Repeat steps 1 through 3 using folds 1, 2, and 4 for cross-validation and fold 3 for the final test set.
5. Repeat steps 1 through 3 using folds 1, 3, and 4 for cross-validation and fold 2 for the final test set.
6. Repeat steps 1 through 3 using folds 2, 3, and 4 for cross-validation and fold 1 for the final test set.

We followed this experimental procedure for each algorithm, repeating the same steps four times to avoid the possibility that by chance our results would be good or bad because of our choice of training data.

For all algorithms, we set $C_s = 1$ for all subcategories s in Category A. The regularization parameter $C_s = C$ for all subcategories s in Category B varied for each method such that the contribution in the objective function from Category B was smaller than the contribution from Category A. Table 4 shows the different parameter values tested for each algorithm. For ℓ_1RE, the two terms of the objective function are identical, so we chose C_{high} to be 0. For ℓ_2RE, we chose C_{high} to be 0, 0.5, or 1. For MIO-RE, we chose θ to be 0 or 9, so that the top of the list was weighed by a factor of 1 or 10 respectively.

In total, for the cross-validation step, there were $6 \times 3 = 18$ problems to solve for LS1, LS2, and LS3; $6 \times 2 = 12$ problems for ℓ_1RE, $6 \times 2 \times 3 = 36$ problems for ℓ_2RE, and $6 \times 2 \times 2 = 24$ problems for MIO-RE. (For each method, the total number of problems was the number of different parameter settings times six, which is the number of ways to choose two out of four folds for training.) For the test step, there were an additional four problems for each method. This set of experiments required approximately 163 hours of computation time.

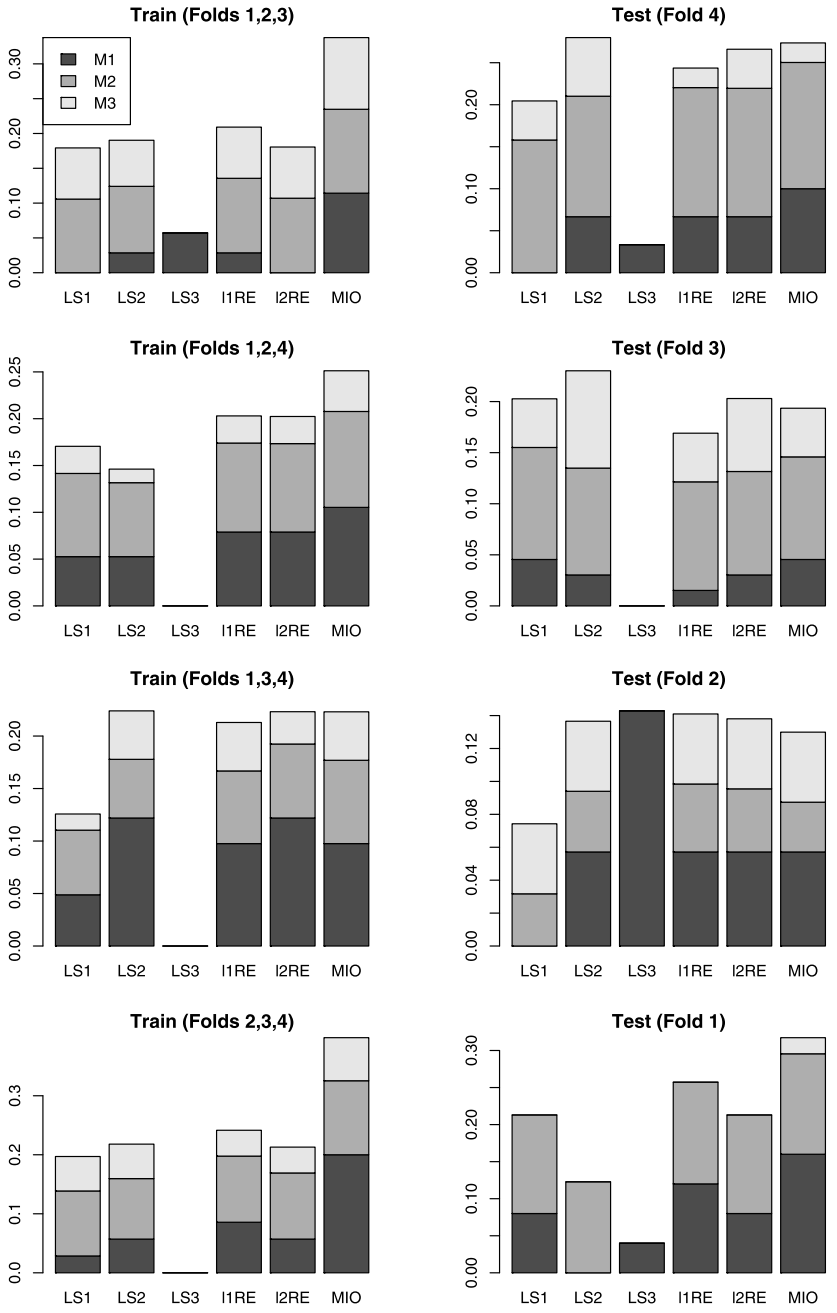

Fig. 2 Barplot summary of results from four rounds of training on three folds and testing on the fourth: M1 (*dark*), M2 (*medium*), and M3 (*light*)

Table 5 Average of M1 metric over four rounds for each algorithm

Algorithm	M1 (train)	M1 (test)
LS1	0.767	0.794
LS2	0.792	0.798
LS3	0.752	0.811
ℓ_1RE	0.797	0.820
ℓ_2RE	0.792	0.814
MIO-RE	**0.836**	**0.840**

8.2 Results

There are four rounds of the experiment in which we train on three folds and test on the fourth fold (step 3 in the procedure above), with the parameter values found through cross-validation. Tables 17, 18, 19, and 20 in the Appendix show the training and test values of M1, M2, and M3 in each of these four rounds. Figure 2 is a visualization of Tables 17 through 20 and shows barplots of M1, M2, and M3 from each of the four rounds; note that for each algorithm, the bars for the three measures have been stacked for compactness. The bar heights are relative instead of absolute; for example, the bar heights for the dark bars (M1) in the top left plot were computed as follows:

1. Let $\mathrm{M1}_m$ be the value of M1 for method m, where m is either LS1, LS2, LS3, ℓ_1RE, ℓ_2RE, or MIO-RE. Note that these are the M1 values from training on folds 1, 2, and 3.
2. Let $\mathrm{M1}_{\min}$ be the minimum of the six $\mathrm{M1}_m$ values.
3. The bar height for method m is the percentage increase of $\mathrm{M1}_m$ from $\mathrm{M1}_{\min}$:

$$\frac{\mathrm{M1}_m - \mathrm{M1}_{\min}}{\mathrm{M1}_{\min}}.$$

The method for which $\mathrm{M1}_m = \mathrm{M1}_{\min}$ has bar height 0. The other bar heights are computed similarly; for each measure, there is at least one method for which the bar height is 0, corresponding to the worst performing method(s). Thus, it is easy from the figure to see, within each barplot, the relative magnitudes of the three measures across all algorithms. For instance, in the top left barplot, MIO-RE clearly is largest in terms of dark bars (M1) and light bars (M3), though it is about the same as all other algorithms in terms of medium bars (M2).

As stated in Sect. 5, we are most interested in M1, which measures ranking quality at the top of the ranked list. Figure 2 shows that with respect to M1, though not always the best, MIO-RE performed consistently well for both training and testing. In contrast, the other algorithms may have performed well for some training or test cases but also performed poorly for other cases. Table 5 shows just the M1 metric from Tables 17 through 20, averaged for each algorithm over the four rounds. MIO-RE has a clear advantage over the other methods according to these sums.

To view the results in a nonparametric way, for each of M1, M2, and M3 and each training or test set, we assigned each of the six algorithms a rank equal to the number of algorithms that performed strictly worse; if all algorithms performed differently for a certain metric, then the ranks would be 0, 1, 2, 3, 4, and 5. There are four sets of ranks corresponding to the four rounds of training and testing. These ranks are shown in Tables 17 through 20 in the Appendix. In Table 6, we sum up the ranks over the four rounds. The consistently high performance of MIO-RE is also reflected in this table, particularly in its advantage in terms of training and testing for M1.

Table 6 Sums of ranks over four rounds for each algorithm

		LS3	LS1	LS2	ℓ_2RE	ℓ_1RE	MIO-RE
Train	M1	4	3	9	9	11	17
	M2	0	8	4	13	13	20
	M3	0	8	8	7	8	18
Total		**4**	**19**	**21**	**29**	**32**	**55**
Test	M1	7	6	5	7	8	15
	M2	0	14	8	11	18	8
	M3	0	5	11	8	3	8
Total		**7**	**25**	**24**	**26**	**29**	**31**

Note that LS1 has an inherent advantage over the other five methods in that it uses information—namely the true scores—that is not available to the other methods that use only the ranks. As discussed earlier, in many cases the true scores may not be available if the rating company does not provide them. Even if the scores are available, our experiment demonstrates that it is possible for methods that encode only the ranks, such as MIO-RE, to have comparable or better performance than methods that directly use the scores. For example, in all but the third round of our experiment, it appears that there was a particularly good solution that none of the approximate methods found, but that MIO-RE did, similar to the results in Sect. 7. This is the major advantage of exactly optimizing the objective function rather than using a convex proxy.

8.3 Example of differences between methods on evaluation measures

It is not immediately clear how a difference in evaluation measures corresponds to differences between ranked lists. In order to show this correspondence, we directly compare ranked lists corresponding to the test set in the fourth round (train on folds 2, 3, and 4; test on fold 1). The ranked lists shown in Table 7 were generated by scoring the products using MIO-RE and LS3, and are divided into the eight subcategories in Category A. For confidentiality purposes, the actual product names have been replaced by the names of various wineries in eight different regions of California.[6]

As indicated by the test measures, reproduced in Table 8, MIO-RE and LS3 were comparable in terms of correctly classifying products as either in the top or not in the top (M3). However, MIO-RE performed much better in terms of pairwise rankings (M1 and M2). For example, MIO-RE correctly ranked all products in the Lake County subcategory while LS3 switched the first and third products; MIO-RE switched the first two products in the Southern California subcategory while LS3 also switched the last two; and the MIO-RE rankings for the Central Valley subcategory were not inaccurate by more than three places for any product while LS3 ranked the second product in the eighth position and the eighth product in the third position. There are several other differences between the ranked lists that help to explain the differences in the evaluation measures.

[6] http://www.cawinemall.com/region.shtml.

Table 7 Example of ranked lists produced by different algorithms, corresponding to metrics in Table 8

True	MIO-RE	LS3
LakeCounty		
Brassfield	Brassfield	Wildhurst
Langtry	Langtry	Langtry
Wildhurst	Wildhurst	Brassfield
NorthCoast		
Alpen	Alpen	Alpen
Fieldbrook	Fieldbrook	Fieldbrook
Winnett	Winnett	Winnett
SouthCali		
Faulkner	Lenora	Lenora
Lenora	Faulkner	Faulkner
Peralta	Peralta	Peralta
Salerno	Salerno	Thompkin
Thompkin	Thompkin	Salerno
Mendocino		
Baxter	Navarro	Navarro
Goldeneye	Baxter	Baxter
Navarro	Goldeneye	Goldeneye
Skylark	Skylark	Skylark
CentralCoast		
Blackstone	Blackstone	Morgan
Estancia	Estancia	Blackstone
Jenkins	Morgan	Ronan
Morgan	Parsonage	Estancia
Newell	Newell	Ventana
Parsonage	Jenkins	Jenkins
Ronan	Ronan	Newell
Ventana	Ventana	Parsonage

True	MIO-RE	LS3
CentralVal		
Accardi	Accardi	Accardi
Baywood	Baywood	Mariposa
Cantiga	Mariposa	Trimble
Harmony	Cantiga	Harmony
Mariposa	Omega	Cantiga
Omega	Watts	Omega
Trimble	Harmony	Watts
Watts	Trimble	Baywood
SierraFoot		
Auriga	Auriga	Auriga
Chevalier	Chevalier	Paravi
Dillian	Paravi	Chevalier
Fitzpatrick	Dillian	Solomon
Hatcher	Fitzpatrick	Oakstone
Montevina	Hatcher	Hatcher
Oakstone	Montevina	Fitzpatrick
Paravi	Oakstone	Dillian
Renwood	Solomon	Renwood
Solomon	Renwood	Montevina
Venezio	Venezio	Venezio
NapaValley		
Carter	Falcor	Falcor
Falcor	Carter	Carter
Ilsley	Ilsley	Kelham
Kelham	Kelham	Ilsley
Mason	Mason	Mason
Oberon	Oberon	Oberon
Quintessa	Relic	Quintessa
Relic	Quintessa	Trefethen
Sawyer	Sawyer	Relic
Trefethen	Varozza	Sawyer
Varozza	Trefethen	Varozza

9 Determining a cost-effective way to achieve top rankings

Having reverse-engineered the ranking model, it is useful to investigate the following: given a current product x, how can its features be cost-effectively modified so that the new product achieves a top ranking? For instance, suppose we would like to find the most cost-effective way to achieve a top ranking point-and-shoot digital camera. In particular, let there be L ways to change a current product, where multiple changes could potentially be made simultaneously. For example, we can change a current digital camera by enlarging the battery and

Table 8 Comparison of MIO-RE and LS3 (train on folds 2, 3, and 4; test on fold 1), corresponding to ranked lists in Table 7

Algorithm	M1	M2	M3
MIO-RE	0.967	0.904	0.887
LS3	0.867	0.796	0.868

by making it out of heavier material. Let the decision variable α_ℓ encode whether change ℓ is implemented. The α_ℓ are binary, that is, either the change is implemented or not:

$$\alpha_\ell = \begin{cases} 1, & \text{if change } \ell \text{ is implemented,} \\ 0, & \text{otherwise.} \end{cases}$$

If change ℓ is implemented, then there is an associated cost, denoted c_ℓ, and factor j of product x will increase by an amount $\delta_{j\ell}(x)$:

$$\alpha_\ell = 1 \quad \Longrightarrow \quad x_j \leftarrow x_j + \delta_{j\ell}(x).$$

It is possible that implementing change ℓ can affect more than one factor. Making a digital camera out of heavier material affects its weight and perhaps also its ability to handle shake, for example. Moreover, some of the $\delta_{j\ell}$ values and costs may be negative, as the most cost-effective way to increase the ranking of a product may be to decrease some factors while increasing others. That is, it might be economical to spend less on one factor and instead fund another change that contributes more to increasing the score. The total change in factor j of product x is

$$\sum_{\ell=1}^{L} \alpha_\ell \delta_{j\ell}(x).$$

There may be possible changes that conflict with each other, and we take this into account as follows: let there be M index sets of changes where at most one of these changes is allowed, and let S_m denote the mth set. Then we have the exclusivity constraints

$$\sum_{\ell \in S_m} \alpha_\ell \leq 1, \quad \forall m = 1, \ldots, M.$$

For instance, we cannot increase a camera's resolution both by one megapixel and by two megapixels; at most one of these two changes can occur.

Let the current score of product x be

$$v_0(x) = w^T x = \sum_{j=1}^{d} w_j x_j.$$

For a given vector of changes $\alpha \in \{0, 1\}^L$, the new score of product x after the changes are made is

$$\begin{aligned} v_{\text{new}}(x) &= \sum_{j=1}^{d} w_j \left(x_j + \sum_{\ell=1}^{L} \alpha_\ell \delta_{j\ell}(x) \right) = \sum_{j=1}^{d} w_j x_j + \sum_{j=1}^{d} w_j \left(\sum_{\ell=1}^{L} \alpha_\ell \delta_{j\ell}(x) \right) \\ &= v_0(x) + \sum_{j=1}^{d} w_j \sum_{\ell=1}^{L} \alpha_\ell \delta_{j\ell}(x) = v_0(x) + \sum_{\ell=1}^{L} \alpha_\ell \sum_{j=1}^{d} w_j \delta_{j\ell}(x) \\ &= v_0(x) + \sum_{\ell=1}^{L} \alpha_\ell W_\ell(x), \end{aligned}$$

where $W_\ell(x) = \sum_{j=1}^{d} w_j \delta_{j\ell}(x)$. Note that $W_\ell(x)$ is the change in score that would result from making change ℓ. Then

$$v_{\text{diff}}(x) = v_{\text{new}}(x) - v_0(x) = \sum_{\ell=1}^{L} \alpha_\ell W_\ell(x)$$

is the total score difference. The total cost associated with the changes in α is

$$c_{\text{diff}}(\alpha) = \sum_{\ell=1}^{L} c_\ell \alpha_\ell.$$

The cost trades off with the change in score. In what follows, we show how to both maximize the change in score on a fixed budget, and how to minimize the cost to achieve a certain change in score.

9.1 Two formulations

We directly provide the formulations for, first, achieving a cost-effective increase in score, and, second, minimizing cost for a fixed target score.

9.1.1 Maximizing score on a fixed budget

The first problem is to fix the budget for making changes and maximize the new score of product x, which is equivalent to maximizing v_{diff}. That is, we want to maximize $\sum_{\ell=1}^{L} \alpha_\ell W_\ell(x)$ while not exceeding some bound on the cost, denoted $\bar{c}$. The integer optimization formulation to solve this problem is given by:

$$\begin{aligned} \max_{\alpha} \quad & \sum_{\ell=1}^{L} \alpha_\ell W_\ell(x) \\ \text{s.t.} \quad & \sum_{\ell=1}^{L} c_\ell \alpha_\ell \le \bar{c}, \\ & \sum_{\ell \in S_m} \alpha_\ell \le 1, \quad \forall m = 1, \ldots, M, \\ & \alpha_\ell \in \{0, 1\}, \quad \forall \ell = 1, \ldots, L. \end{aligned} \tag{14}$$

9.1.2 Minimizing cost with a fixed target score

Suppose the target score is v_{tar}, so that the desired score difference is $v^*_{\text{diff}} = v_{\text{tar}} - v_0(x)$. The integer optimization formulation is given by:

$$\begin{aligned} \min_{\alpha} \quad & \sum_{\ell=1}^{L} c_\ell \alpha_\ell \\ \text{s.t.} \quad & \sum_{\ell=1}^{L} \alpha_\ell W_\ell(x) \ge v^*_{\text{diff}}, \\ & \sum_{\ell \in S_m} \alpha_\ell \le 1, \quad \forall m = 1, \ldots, M, \\ & \alpha_\ell \in \{0, 1\}, \quad \forall \ell = 1, \ldots, L. \end{aligned} \tag{15}$$

Table 9 Point-and-shoot digital camera factors

1	2	3	4	5
Resolution	Weight	Photo quality	Video quality	Response time
6	7	8	9	10
Handling shake	Versatility	LCD quality	Widest angle	Battery life

Table 10 Coefficients of scoring function for digital cameras

w_1	w_2	w_3	w_4	w_5
0.584	−0.571	4.342	2.926	3.769
w_6	w_7	w_8	w_9	w_{10}
1.137	1.442	2.896	0.005	0.001

Table 11 Scores of two example cameras

Camera	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	Score
1	14	5	5	5	5	5	5	5	35	500	88.38
2	12	5	4	4	4	3	4	4	30	300	69.41

By solving the first formulation for a range of budgets, or by solving the second formulation for a range of target scores, we can map out an efficient frontier of maximum score for minimum cost. This concept is best explained through an example, which we present in the next section.

9.2 Practical example

We use the example of finding the most cost-effective way to increase the rank of a point-and-shoot digital camera. Suppose there are 10 factors, shown in Table 9. Resolution is in number of megapixels, weight is in ounces, widest angle is in millimeters, and battery life is in number of shots. All other factors take values between 1 and 5, in increments of 0.5, with 1 representing poor quality and 5 representing excellent quality. Let the coefficients of the scoring function $f(x) = w^T x$ be as shown in Table 10. The coefficient corresponding to camera weight is negative since it is desirable to have a lighter camera. Table 11 shows the scores of two different cameras according to this scoring function.

There are twelve possible changes that we can make to a particular hypothetical digital camera x. Table 12 shows the cost of making each change in dollars per camera, as well as the effect $\delta_{j\ell}$ each change ℓ has on factor j. A dot indicates the effect is 0. Table 12 does not apply to all cameras, and the $\delta_{j\ell}$'s might need to be constructed individually for each camera. In particular, we assume that none of the six integer factors of camera x would exceed the upper bound of 5 if any of the changes were implemented. For instance, x could not be the first camera in Table 11 since factors 2 through 8 are already at their maximum possible value, but it could be the second.

Table 13 shows the conflict sets S_m. For instance, the changes "Add 1 Megapixel" (change 2) and "Add 2 Megapixels" (change 6) are mutually exclusive. These conflicts are incorporated in (14) and (15) in the exclusivity constraints. We represent the conflict

Table 12 Change information for a digital camera

	Change	δ_1	δ_2	δ_3	δ_4	δ_5	δ_6	δ_7	δ_8	δ_9	δ_{10}	Cost
1	Larger battery	·	·	·	·	·	·	·	·	·	50	2
2	Add 1 megapixel	1	·	·	·	·	·	·	·	·	·	3
3	Better LCD	·	·	·	·	·	·	·	0.5	·	·	4
4	More modes	·	·	·	·	·	·	1	·	·	·	4
5	Wider angle	·	·	0.5	·	·	·	·	·	2	·	5
6	Add 2 megapixels	2	·	0.5	·	·	·	·	·	·	·	5
7	Heavier material	·	1	·	·	·	1	·	·	·	·	5
8	Better video	·	·	·	1	·	·	·	·	·	·	6
9	Faster response	·	·	·	·	0.5	·	·	·	·	·	6
10	Better lens	·	·	0.5	1	·	·	·	·	·	·	7
11	Fastest response	·	·	·	·	0.5	1	·	·	·	·	7
12	Most modes	·	·	1	·	0.5	·	1	·	·	·	9

Table 13 Conflict sets ($M = 6$)

m	S_m
1	{2, 6}
2	{5, 6, 10, 12}
3	{8, 10}
4	{9, 11, 12}
5	{7, 11}
6	{4, 12}

between changes 2 and 6 as

$$\alpha_2 + \alpha_6 \leq 1, \quad \text{or} \quad \sum_{\ell \in S_1} \alpha_\ell \leq 1,$$

where $S_1 = \{2, 6\}$. Table 14 gives an alternative way to represent the conflicts and shows for each of the twelve changes, which of the other changes conflict with it.

Each point in Fig. 3 corresponds to one of the 512 feasible changes or combinations of changes, and its position indicates its cost and effect on the score. We can trace out a frontier of solutions that lead to maximum changes in score for minimum cost, also shown in Fig. 3. For example, suppose that we fix the maximum cost at 7. In Fig. 4, we see that for a cost of 7, the maximum difference in score is 5.097, which we find corresponds to the single change "Better Lens." Note that for a maximum cost of 8, the best solution stays the same. That is, even if we were willing to spend up to 8, the maximum difference in score would be achieved by the same solution as if we were willing to spend only up to 7. Extending this example for other costs, Table 15 shows the changes that lead to maximum differences in score for fixed budgets between 2 and 10. These results address the first problem in Sect. 9.1. To address the second problem in Sect. 9.1, Table 16 shows the changes that have minimum cost for lower bounded differences in score between 1 and 7. For instance, suppose that we specify that the difference in score is at least 2. The table shows that there are two ways to achieve this difference with the minimum cost of 5, namely by the changes "Wider Angle,"

Table 14 Conflicts between changes

	Change	Conflicts
1	Larger battery	·
2	Add 1 megapixel	6
3	Better LCD	·
4	More modes	12
5	Wider angle	6, 10, 12
6	Add 2 megapixels	2, 5, 10, 12
7	Heavier material	11
8	Better video	10
9	Faster response	11, 12
10	Better lens	5, 6, 8, 12
11	Fastest response	7, 9, 12
12	Most modes	4, 5, 6, 9, 10, 11

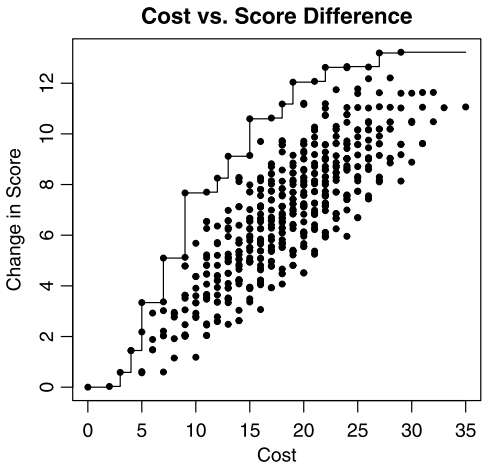

Fig. 3 Changes in score and corresponding costs for digital camera example

which corresponds to an actual score difference of 2.182, or "Add 2 Megapixels," which corresponds to a higher score difference of 3.339, also illustrated in Fig. 5.

Tables 15 and 16 demonstrate that we can generate look-up tables for a variety of schemes to enhance a product so that it reaches a higher ranking with minimal cost. The tables could, in fact, be expanded up to a cost of 35. The highest possible cost of any feasible combination of changes is 35, as shown in Fig. 3. For datasets that are much larger than this digital camera example, (14) and (15) provide an efficient way to generate the look-up tables. There may be multiple optima, but it is also straightforward to find all of them using the iterative algorithm shown in Fig. 6, which uses (14) or (15) as a subroutine. The algorithm finds all optimal solutions by iteratively solving (14) or (15), and adding a constraint in each iteration that makes the previous optimum infeasible, until the optimal cost changes. This algorithm finds the efficient frontier without having to enumerate all possible solutions, as we did in Fig. 3.

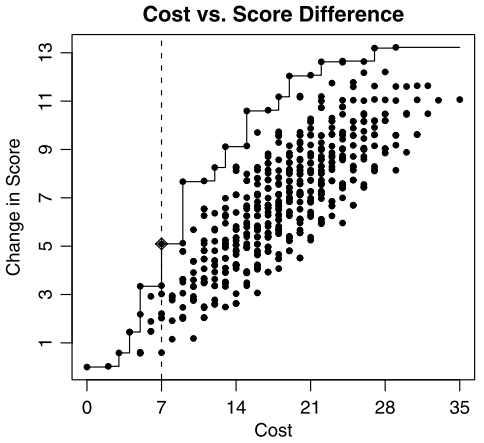

Fig. 4 If we fix the maximum allowed cost at 7 (*dotted line*), then the highest possible change in score is 5.097 (one optimum, indicated by a *diamond*)

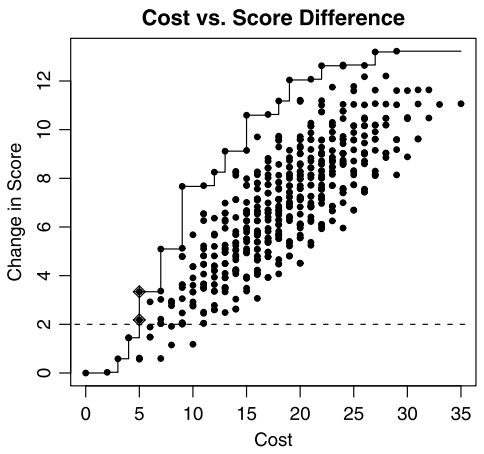

Fig. 5 If we fix the minimum allowed change in score at 2 (*dotted line*), then the lowest possible cost is 5 (two optima, indicated by *diamonds*)

Table 15 Lookup table for fixed budget

Max cost	Optimal change(s)	Score diff	Actual cost
2	Larger battery	0.030	2
3	Add 1 megapix	0.584	3
4	Better LCD	1.448	4
5	Add 2 megapix	3.339	5
6	Better video	2.926	6
7	Better lens	5.097	7
8	Better lens	5.097	7
9	Most modes	7.669	9
10	Most modes	7.669	9
⋮	⋮	⋮	⋮

Table 16 Lookup table for target score

Min diff	Optimal change(s)	Cost	Actual diff
1	More modes	4	1.442
	Better LCD		1.448
2	Wider angle	5	2.182
	Add 2 megapix		3.339
3	Add 2 megapix	5	3.339
4	Better lens	7	5.097
5	Better lens	7	5.097
6	Most modes	9	7.669
7	Most modes	9	7.669
⋮	⋮	⋮	⋮

Let formulation (14') = formulation (14) [or (15') = (15)]
Solve (14') [or (15')]
$z =$ optimal value of the objective function
$\alpha^* =$ optimal solution
$Z = z$
Initialize list of optima with α^*
While $Z = z$
$S_0 \leftarrow \{\ell : \alpha^*_\ell = 0\}$
$S_1 \leftarrow \{\ell : \alpha^*_\ell = 1\}$
Add constraint $\sum_{\ell \in S_0} \alpha_\ell + \sum_{\ell \in S_1} (1 - \alpha_\ell) \geq 1$ to (14') [or (15')]
Solve (14') [or (15')]
$\alpha^* \leftarrow$ optimal solution
$Z \leftarrow$ optimal value of the objective function
Add α^* to list of optima
Output: complete list of optima

Fig. 6 Algorithm to find multiple optima

10 Conclusion

We presented a machine learning approach to reverse-engineering ranking models, and an experiment on data from a rating company. The formulation encodes a specific preference structure and categorical organization of the products. Another contribution of our work is the introduction of evaluation measures that take into account the rank of a new product, relative to the products that have already been ranked. Finally, we showed how to use a reverse-engineered ranking model to achieve a high rank for a product in a cost-effective way.

This leads to many avenues for future work, for instance, it would be useful to develop an algorithm that solves the ranking problem while locating potential errors in the data. Another idea is to quantify the uncertainty in each of the coefficients in the reverse-engineered model.

Acknowledgements This material is based upon work supported by the MIT-Ford Alliance and the National Science Foundation under Grant No IIS-1053407. We would like to thank Dimitris Bertsimas from MIT, Brian Jahn and Larry Kummer from Ford, and Elaine Savage, John Leonard, and Ed Krause from the MIT-Ford Alliance.

Appendix

Tables 17, 18, 19, and 20 show the training and test values of M1, M2, and M3 in each of the four rounds of the experiment in Sect. 8. The highest training and test measures are highlighted in bold. The integer number next to each measure is the rank of the method, that is, the number of other methods below it for the particular measure and dataset (training or test). Note that 0 is the lowest possible rank by this definition.

Table 17 Training and test values of M1, M2, and M3 on ratings data, and ranks of algorithms (train on folds 1, 2, and 3; test on fold 4)

Algorithm		M1		M2		M3	
LS1 $C = 0$	train	0.686	0	0.920	2	0.930	2
	test	0.714	0	**0.922**	5	0.865	3
LS2 $C = 0.1$	train	0.706	2	0.911	1	0.924	1
	test	0.762	2	0.911	1	**0.885**	5
LS3 $C = 0.05$	train	0.725	4	0.832	0	0.866	0
	test	0.738	1	0.797	0	0.827	0
ℓ_1RE $C = 0.1$, $C_{\text{high}} = 0$	train	0.706	2	0.921	3	0.930	2
	test	0.762	2	0.919	4	0.846	1
ℓ_2RE $C = 0.1$, $C_{\text{high}} = 1$	train	0.686	0	0.921	3	0.930	2
	test	0.762	2	0.918	3	0.865	3
MIO-RE $C = 0.5, \theta = 0$	train	**0.765**	5	**0.932**	5	**0.955**	5
	test	**0.786**	5	0.916	2	0.846	1

Table 18 Training and test values of M1, M2, and M3 on ratings data, and ranks of algorithms (train on folds 1, 2, and 4; test on fold 3)

Algorithm		M1		M2		M3	
LS1 $C = 0$	train	0.833	1	0.925	2	0.904	2
	test	**0.784**	4	**0.922**	5	0.846	1
LS2 $C = 0.2$	train	0.833	1	0.917	1	0.892	1
	test	0.773	2	0.918	3	**0.885**	5
LS3 $C = 0.05$	train	0.792	0	0.850	0	0.879	0
	test	0.750	0	0.831	0	0.808	0
ℓ_1RE $C = 0.1$, $C_{\text{high}} = 0$	train	0.854	3	0.930	4	0.904	2
	test	0.761	1	0.919	4	0.846	1
ℓ_2RE $C = 0$, $C_{\text{high}} = 0$	train	0.854	3	0.930	3	0.904	2
	test	0.773	2	0.915	2	0.865	4
MIO-RE $C = 0.5, \theta = 0$	train	**0.875**	5	**0.937**	5	**0.917**	5
	test	**0.784**	4	0.914	1	0.846	1

Table 19 Training and test values of M1, M2, and M3 on ratings data, and ranks of algorithms (train on folds 1, 3, and 4; test on fold 2)

Algorithm		M1		M2		M3	
LS1 $C = 0.1$	train	0.843	1	0.913	2	0.841	1
	test	0.778	0	0.925	2	**0.942**	1
LS2 $C = 0$	train	**0.902**	4	0.908	1	**0.866**	3
	test	0.822	1	0.929	3	**0.942**	1
LS3 $C = 0.05$	train	0.804	0	0.860	0	0.828	0
	test	**0.889**	5	0.896	0	0.904	0
ℓ_1RE $C = 0.1$, $C_{\text{high}} = 0$	train	0.882	2	0.919	3	**0.866**	3
	test	0.822	1	**0.933**	5	**0.942**	1
ℓ_2RE $C = 0.1$, $C_{\text{high}} = 1$	train	**0.902**	4	0.920	4	0.854	2
	test	0.822	1	0.931	4	**0.942**	1
MIO-RE $C = 0.5, \theta = 0$	train	0.882	2	**0.928**	5	**0.866**	3
	test	0.822	1	0.923	1	**0.942**	1

Table 20 Training and test values of M1, M2, and M3 on ratings data, and ranks of algorithms (train on folds 2, 3, and 4; test on fold 1)

Algorithm		M1		M2		M3	
LS1 $C = 0$	train	0.706	1	0.932	2	0.929	3
	test	0.900	2	0.902	2	0.868	0
LS2 $C = 0.2$	train	0.725	2	0.925	1	0.929	3
	test	0.833	0	0.894	1	0.868	0
LS3 $C = 0.05$	train	0.686	0	0.839	0	0.878	0
	test	0.867	1	0.796	0	0.868	0
ℓ_1RE $C = 0.1$, $C_{\text{high}} = 0$	train	0.745	4	0.933	3	0.917	1
	test	0.933	4	**0.906**	5	0.868	0
ℓ_2RE $C = 0.1$, $C_{\text{high}} = 0.5$	train	0.725	2	0.933	3	0.917	1
	test	0.900	2	0.902	2	0.868	0
MIO-RE $C = 0.5, \theta = 0$	train	**0.824**	5	**0.944**	5	**0.942**	5
	test	**0.967**	5	0.904	4	**0.887**	5

References

Ataman, K., Street, W. N., & Zhang, Y. (2006). Learning to rank by maximizing AUC with linear programming. In *International joint conference on neural networks*.

Bajek, E. (2008). (Almost) How the Elias Sports Bureau rankings work. Unpublished blog entry at http://tigers-thoughts.blogspot.com/2008/07/almost-how-elias-sports-bureau-rankings.html.

Bertsimas, D., & Weismantel, R. (2005). *Optimization over integers*. Charlestown: Dynamic Ideas.

Bertsimas, D., Chang, A., & Rudin, C. (2010). A discrete optimization approach to supervised ranking. In *Proceedings of the 5th INFORMS workshop on data mining and health informatics (DM-HI 2010)*.

Bertsimas, D., Chang, A., & Rudin, C. (2011). Integer optimization methods for supervised ranking. MIT DSpace, Operations Research Center. Working paper available at http://dspace.mit.edu/handle/1721.1/67362.

Burges, C. J., Ragno, R., & Le, Q. (2006). Learning to rank with nonsmooth cost functions. In *Proceedings of neural information processing systems (NIPS)* (pp. 395–402).

Cao, Z., Qin, T., Liu, T.-Y., Tsai, M.-F., & Li, H. (2007). Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on machine learning (ICML)* (pp. 129–136).

Chandler, S. J. (2006). Analyzing US News and World Report rankings using symbolic regression. Unpublished blog entry at http://taxprof.typepad.com/taxprof_blog/files/analyzing_u.S. News & World Report Rankings Using Symbolic Regression.pdf.

Cossock, D., & Zhang, T. (2006). Subset ranking using regression. In *Proceedings of the 19th conference on learning theory (COLT)* (pp. 605–619).

Ferri, C., Flach, P., & Hernández-Orallo, J. (2002). Learning decision trees using the area under the ROC curve. In *Proceedings of the 19th international conference on machine learning (ICML)* (pp. 139–146).

Freund, Y., Iyer, R., Schapire, R. E., & Singer, Y. (2003). An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research*, *4*, 933–969.

Green, P. E., Krieger, A. M., & Wind, Y. (J.) (2001). Thirty years of conjoint analysis: Reflections and prospects. *Interfaces*, *31*(3), S56–S73.

Hammer, P. L., Kogan, A., & Lejeune, M. A. (2007). Reverse-engineering banks' financial strength ratings using logical analysis of data. Working paper available at http://www.optimization-online.org/DB_FILE/2007/02/1581.pdf.

Joachims, T. (2002). Optimizing search engines using clickthrough data. In *Proceedings of the ACM conference on knowledge discovery and data mining (KDD)*. New York: ACM Press.

Kendall, M. G. (1938). A new measure of rank correlation. *Biometrika*, *30*(1/2), 81–93.

Lafferty, J., & Zhai, C. (2001). Document language models, query models, and risk minimization for information retrieval. In *Proceedings of the 24th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 111–119). New York: ACM Press.

Le, Q., & Smola, A. (2007). Direct optimization of ranking measures. arXiv:0704.3359v1 [cs.IR].

Li, P., Burges, C. J., & Wu, Q. (2007). McRank: learning to rank using multiple classification and gradient boosting. In *Proceedings of neural information processing systems (NIPS)*.

Matveeva, I., Laucius, A., Burges, C., Wong, L., & Burkard, T. (2006). High accuracy retrieval with multiple nested ranker. In *Proceedings of the 29th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 437–444). New York: ACM Press.

Morgenson, G., & Story, L. (2010). Rating agency data aided wall street in deals. *New York Times*, Business section, April 24, 2010. Article at http://www.nytimes.com/2010/04/24/business/24rating.html.

Rudin, C. (2009). The P-Norm Push: a simple convex ranking algorithm that concentrates at the top of the list. *Journal of Machine Learning Research*, *10*, 2233–2271.

Su, A.-J., Hu, Y. C., Kuzmanovic, A., & Koh, C.-K. (2010). How to improve your Google ranking: myths and reality. In *IEEE/WIC/ACM international conference on web intelligence and intelligent agent technology (WI-IAT)* (Vol. 1, pp. 50–57).

Tsochantaridis, I., Joachims, T., Hofmann, T., Altun, Y., & Singer, Y. (2005). Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, *6*, 1453–1484.

Xu, J., Liu, T. Y., Lu, M., Li, H., & Ma, W. Y. (2008). Directly optimizing evaluation measures in learning to rank. In *Proceedings of the 31st annual international ACM SIGIR conference on research and development in information retrieval*. New York: ACM Press.