One-Pass Ranking Models for Low-Latency Product Recommendations

Antonino Freno*
Zalando
Berlin, Germany
antonino.freno@zalando.de

Martin Saveski*
MIT Media Lab
Cambridge, USA
msaveski@mit.edu

Rodolphe Jenatton
Amazon
Berlin, Germany
jenatton@amazon.com

Cédric Archambeau
Amazon
Berlin, Germany
cedrica@amazon.com

ABSTRACT

Purchase logs collected in e-commerce platforms provide rich information about customer preferences. These logs can be leveraged to improve the quality of product recommendations by feeding them to machine-learned ranking models. However, a variety of deployment constraints limit the naïve applicability of machine learning to this problem. First, the amount and the dimensionality of the data make in-memory learning simply not possible. Second, the drift of customers' preference over time require to retrain the ranking model regularly with freshly collected data. This limits the time that is available for training to prohibitively short intervals. Third, ranking in real-time is necessary whenever the query complexity prevents us from caching the predictions. This constraint requires to minimize prediction time (or equivalently maximize the data throughput), which in turn may prevent us from achieving the accuracy necessary in web-scale industrial applications. In this paper, we investigate how the practical challenges faced in this setting can be tackled via an online learning to rank approach. Sparse models will be the key to reduce prediction latency, whereas one-pass stochastic optimization will minimize the training time and restrict the memory footprint. Interestingly, and perhaps surprisingly, extensive experiments show that one-pass learning preserves most of the predictive performance. Additionally, we study a variety of online learning algorithms that enforce sparsity and provide insights to help the practitioner make an informed decision about which approach to pick. We report results on a massive purchase log dataset from the Amazon retail website, as well as on several benchmarks from the LETOR corpus.

*Antonino Freno and Martin Saveski contributed to this work while they were at Amazon Development Center Germany.

KDD'15, August 10-13, 2015, Sydney, NSW, Australia.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-3664-2/15/08 ...$15.00.
DOI: http://dx.doi.org/10.1145/2783258.2788579.

1. INTRODUCTION

Ranking algorithms for document retrieval and (content-based) product recommendation typically work with high-dimensional feature vector representations of the products to be ranked. The available features range from query-independent information (e.g., product category or document topic) to information measuring the match between the user or query and the retrieved product or document (e.g., similarity scores or other relational quantities). The dimensionality of the feature vectors and the complexity of the statistical relationships involved are such that accurate results cannot be achieved by designing the relevant ranking functions manually. Therefore, learning to rank from examples has become the dominant approach when designing and optimizing ranking systems [16, 21].

Learning to rank in web-scale, real-time applications pose at least three major challenges to algorithmic design. First, the learning algorithms must be able to process several gigabytes (when not terabytes) of training data in a limited amount of time. Here, the data set size stems both from the huge number of training examples (in the order of millions to hundreds of millions) and the large number of attributes describing the examples. When a ranking system is deployed to production, a common requirement is to train a new model at regular intervals (e.g., daily) on fresh query logs, which are collected over the past few days. This limits the amount of time available for training to at most a few hours, taking into account the overhead imposed by collateral deployment issues. Second, when the amount of training data is too large, we cannot store anything but a minimal fraction of the available data in memory. This restricts the range of viable learning options to methods characterized by extremely frugal memory requirements. In particular, any learning method whose memory footprint grows with the size of the training data set is a nonstarter in this setting. Third, the learned ranking function must meet strict requirements in terms of latency/throughput. For instance, when the customer of an e-commerce website is browsing the catalogue, we must be able to compute recommendations tied to that particular, dynamically generated context in quasi real time, i.e., at the same time as he or she is loading and scrolling the content of the web page. In practice, this means that, for lists containing hundreds of candidate recommendations, all the products in each list need to be scored/ranked in just a few milliseconds.

The general framework we adopt in order to address these challenges is listwise ranking loss minimization with sparsity-inducing penalties, where optimization is based on single pass of stochastic gradient descent (SGD) updates or variants thereof. In particular, we focus on a set of techniques within the LambdaRank family [3]. The main reasons for this choice are the following. First, LambdaRank provides a straightforward way to customize the loss function in order to accommodate for different ranking evaluation metrics. This allows us to abstract our analysis from a specific application domain, such as product recommendation or document retrieval. Second, SGD requires the smallest memory footprint we might achieve, i.e., one training example for each model update or (small) mini-batches. Third, doing one pass through the training examples minimizes the time needed to stream data from disk or through a network connection, which can be extremely expensive. Finally, we learn sparse models (i.e., models where most of the data attributes play no role at prediction time) to significantly reduce the amount of computation required to score/rank candidates (not only in terms of strict scoring time, but especially in terms of fetching/extracting the required features), which brings us closer to our low-latency requirements.

The impact of sparsity-inducing schemes on ranking quality is a relatively recent topic in the learning-to-rank community. The ℓ_1-based methods that we consider in the next sections are related to the truncated gradient approach [19], which has been applied to text document retrieval in [31]. The authors of [18] also consider a sparsity-inducing approach, but it is based on a different primal-dual optimization scheme. Emphasis on non-convexity issues in sparsity-enforcing ranking loss formulations has been discussed instead in [20]. While we share the same interest in sparsity as the aforementioned studies, at least one point has not been addressed by them (or, to the best of our knowledge, by any related work). Specifically, previous work does not explore the impact (in terms of learning and ranking quality) of working under such constraints as *minimal training time* (i.e., only one pass allowed over the training data) and *minimal prediction latency* (i.e., extremely aggressive sparsity requirements). Furthermore, they evaluate (sparse) ranking models on benchmarks of relatively modest size, both in terms of sample size and in terms of dimensionality. As a concrete illustration, the used benchmarks contained at most 64 features and 148,657 samples (see the summary in [18]). Such a setting is simply not commensurable with the scale of current challenges from the WWW. Therefore, we believe that we are filling a gap in the learning-to-rank research by benchmarking sparsity-inducing schemes in a more realistic scenario.

This paper aims to provide *empirically solid* answers to the following two questions:

1. Can we achieve state-of-the-art ranking quality by one-pass SGD (or variants thereof) over the training data?

2. Can we boost the efficiency of our ranking function to the extent necessary to cope with low-latency (real-time) WWW applications?

Our investigation is rooted in web-scale machine learning practice (as enforced, in particular, in the online retail industry), which we deem to be crucial to back theoretical results with their empirical counterpart.

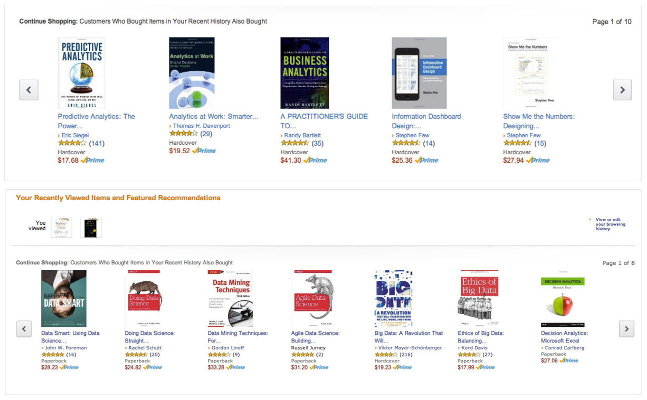

Continue Shopping: Customers Who Bought Items in Your Recent History Also Bought
Page 1 of 10
Predictive Analytics: The Power...
› Eric Siegel
(141)
Hardcover
$17.68 Prime
Analytics at Work: Smarter...
› Thomas H. Davenport
(29)
Hardcover
$19.52 Prime
A PRACTITIONER'S GUIDE TO...
› Randy Bartlett
(35)
Hardcover
$41.30 Prime
Information Dashboard Design:...
› Stephen Few
(14)
Hardcover
$25.36 Prime
Show Me the Numbers: Designing...
› Stephen Few
(15)
Hardcover
$27.94 Prime
Your Recently Viewed Items and Featured Recommendations
You viewed
View or edit your browsing history
Continue Shopping: Customers Who Bought Items in Your Recent History Also Bought
Page 1 of 8
Data Smart: Using Data Science...
› John W. Foreman
(16)
Paperback
$28.23 Prime
Doing Data Science: Straight...
› Rachel Schutt
(20)
Paperback
$24.82 Prime
Data Mining Techniques: For...
› Gordon Linoff
(9)
Paperback
$33.28 Prime
Agile Data Science: Building...
Russell Jurney
(2)
Paperback
$31.20 Prime
Big Data: A Revolution That Will...
› Viktor Mayer-Schönberger
(216)
Hardcover
$19.23 Prime
Ethics of Big Data: Balancing...
› Kord Davis
(27)
Paperback
$17.99 Prime
Decision Analytics: Microsoft Excel
› Conrad Carlberg
Paperback
$27.06 Prime

Figure 1: Two examples of Amazon's Recent History Footer (RHF) widget showing 5 (top) and 7 (bottom) products to the customer. The number of products shown depends on the browser's viewing settings.

The paper is organized as follows. We provide an overview of the use case in Section 2 and discuss related ranking evaluation metrics in Section 3. In Section 4, we discuss related work. The listwise ranking model and the on-line learning algorithms are introduced in Section 5. Section 6 describes the Bayesian optimization technique that we use in order to make hyperparameter tuning, both, accurate and efficient. In Section 7, we assess the predictive performance of our approach by looking at three benchmark datasets from the LETOR collection, which is a universally accessible reference for research on learning to rank. After demonstrating the quality of the adopted method on these benchmarks, we investigate the behavior of the proposed approach on a more challenging, web-scale application based on Amazon retail data and discuss the latency/predictive power tradeoff (Section 8). The lessons learned from our study are summarized in Section 9.

2. USE CASE: RECOMMENDATIONS BY AMAZON WIDGETS

Consider the two examples shown in Figure 1. Each one is a list of impressions from the recent history footer (RHF) widget. RHF appears at the bottom of many pages on the retail website. The widget displays a number of products which are deemed to be relevant w.r.t. the most recent browsing history of a website visitor (rather than the whole browsing/purchase history, or an aggregate, non-personalized relevance model). When the RHF widget is displayed to a customer, a smaller or larger number of recommendations are actually shown depending on viewport constraints, typically screen resolution, browser font size, etc.

As depicted in Figure 1 the product impressions are shown horizontally. The relative ordering of the products within a widget of size k is not crucial to capture the customer's attention; any product appearing within the top k will be considered relevant. However, once the size k of the widget is determined, it is important to move the products that are most likely to be considered by the customer within the top k positions. If a product which is appealing for the customers is ranked below position k, they do not have the possibility to go directly to the detail page.

Other widgets, with a similar structure, are displayed on the product detail pages from the Amazon retail portal. One such widget is the "Customers Who Bought This Item Also Bought" carousel. Instead of considering relevance to the customer's recent browsing history, this widget focuses on relevance to the currently visited product detail page, based on the criterion of a purchase-to-purchase similarity. Yet, another widget is based on view-to-purchase similarity ("What Other Items Do Customers Buy After Viewing This Item?"). Each widget provides a different type of recommendations by ranking different sets of products.

3. RANKING METRICS

The goal in product recommendation is to recommend products that maximize the click-through rate (CTR) or the purchase rate (PR), which are computed based on the impression logs of these products. CTR and PR express an ordering of the products in the catalog (or any subset of it), which we will call the target ranking. Several ranking evaluation metrics have been proposed in information retrieval [7] to assess the quality of rankings produced, for example, by machine-learned ranking models. These data-driven models try to minimize the discrepancy between the ranking they produce and the target ranking.

Consider a list of n products that is (partially) displayed to one customer. We denote by $\boldsymbol{l} = (l_1, \dots, l_n)$ the relevance labels associated to the products in this list. Label l_i is the feedback provided by the customer regarding product i. It could be an integer in $\{1, \dots, n\}$, but it need not be. For example, the feedback could be in some cases continuous, like the purchase price of the product, or ordinal, like a rating by the customer. We will consider the case where every display of a list of products will result in at most one non-zero feedback.

Let $\boldsymbol{r} = (r_1, \dots, r_n)$ be a ranking of the products in the list. The rank r_i of product i is an integer in $\{1, \dots, n\}$, where 1 is the first position and n the last one. A popular metric to evaluate the quality of $\boldsymbol{r}$ in light of the relevance labels $\boldsymbol{l}$ is the discounted cumulative gain (DCG):

$$DCG(\boldsymbol{r}, \boldsymbol{l}) = \sum_{i=1}^{n} \frac{2^{l_i} - 1}{\log_2(r_i) + 1}. \tag{1}$$

DCG is non-negative and larger values correspond to better rankings; the contribution of products with low relevance labels that are poorly ranked are downweighted as desired. A closely related evaluation metric is NDCG, which is obtained by normalizing the righthand side of (1) such that the perfect ranking corresponds to a value of NDCG equal to 1 and does not depend on the length of the list. NDCG, can be further parameterized by a threshold $k \in \{1, \dots, n\}$, such that the summation only runs over the top k products of the proposed ranking $\boldsymbol{r}$. We will call this measure NDCG@K. This variant is appealing in practice as one is usually only interested in an accurate estimation of the rank of the most relevant products.

When the relevance labels are restricted to be binary (i.e. $l_i \in \{0, 1\}$), for example, because we only care about relevant products being ranked at position k or above, we can use recall at k (R@K):

$$R@K(\boldsymbol{r}, \boldsymbol{l}) = \frac{\sum_{i=1}^{n} \mathbb{I}(r_i \leqslant k) l_i}{\sum_{i=1}^{n} l_i}, \tag{2}$$

where $\mathbb{I}(\cdot)$ is the indicator function. R@K is non-negative and larger values correspond to better rankings. This evaluation metric is more suitable than NDCG@K when the relative sorting of the top products does not matter.

Other popular metrics include precision at k, mean average precision, mean reciprocal rank, Kendall's tau and the area under the ROC curve. We refer the interested reader to [5, 22] for in-depth discussions of these metrics, but we will not further discuss them as the most appropriate metric depends on the target application. In Section 7, we report results in terms of NDCG@K as this is the evaluation metric typically used to compare ranking models on the benchmark data sets we consider. In Section 8, we adopt R@K as it matches our use case more closely.

4. BACKGROUND AND RELATED WORK

Learning-to-rank models can be classified into three broad families: pointwise, pairwise, and listwise methods [21]. *Pointwise* approaches postulate a scoring function and attempt to estimate the relevance score of every item. The relevance score is typically the rank of the item in the list or a transformed version of it. At prediction time, the items returned for a query are sorted according to their estimated scores. Linear or logistic regression are examples of scoring functions used in pointwise approaches. *Pairwise* methods score ordered pairs of items instead of individual items. The goal is now to learn the order of such pairs correctly. In other words, the task is to score the more relevant items higher than less relevant ones. In general, this approach is preferable to the pointwise approach, because it does not require to learn absolute relevance scores. RankSVM [12, 16] is one of the most popular pairwise approaches. It formalizes ranking as a binary classification problem of item pairs and uses support vector machines as the underlying binary classifier. Alternatively, one could use Rank Logistic Regression (RankLR), which was shown to work slightly better [29]. RankBoost [11] is another pairwise ranking model, where boosting is used to learn the ranking. The idea is to construct a sequence of 'weak rankers' over iteratively reweighted training data, and then to make rank predictions using a linear combination of the weak learners. While the predictive power of RankBoost is greater in theory, it only marginally improves the quality of the ranking in practice (see Section 7). Finally, *listwise* approaches assume that the training examples are lists of ranked items. They attempt to minimize a loss function defined over the whole list instead of ordered pairs extracted from the list. ListNet [6] is a listwise ranking algorithm, which performs gradient descent over a loss function defined in terms of the cross-entropy. AdaRank [34] is another listwise approach, based instead on boosting. While more costly to learn, in practice these methods perform similarly to pairwise approaches (see Section 7). Recently, gradient-boosted trees (GBTs) have become pretty popular in learning to rank [4, 25]. Although GBTs have been shown to outperform simpler approaches (e.g., plain linear models), training them in the large-scale setting can be very expensive. Moreover, scoring latency is a serious issue for GBTs whenever we are not able to precompute and cache predictions for the task at hand, for the following reasons. In typical real-world applications, several hundreds of base learners (i.e. regression trees) are necessary to reach the desired accuracy, where each tree has about 4 to 8 leaves. This means going through a few thousands of decision rules for each candidate to be scored, which is problematic in our setting. In particular, parallelizing decisions would not be a solution in this case, because single-host multithreading has further latency overhead in terms of spawning the threads, whereas distributing to multiple hosts is even less effective because of inter-host communication. On top of these considerations about scoring latency, one more difficulty is that, without enforcing global feature selection strategies, most or all of the involved features will be used throughout our set of regression trees. This means having to go through the whole feature extraction process, which is often the most expensive side of online candidate scoring. On the other hand, sparse modelling would not suffer from such an overhead by simply dropping features from the scoring function. An alternative model, which has been shown to be very effective for the large-scale setting, is referred to as the WSABIE algorithm [32]. While WSABIE also uses SGD learning, it has not been targeted at the one-pass setting. Moreover, low-dimensional embeddings and data sub-sampling are the main learning tools used by that model, whereas our focus is on sparsity-inducing regularization schemes.

A recent trend in learning to rank is to attempt optimizing the ranking evaluation metric *directly*. This class of approaches typically falls into the listwise approaches, as the ranking metrics are functions of ranked lists (rather than of individual pairs or individual items). In principle, such listwise approaches should be able to outperform pairwise or pointwise methods w.r.t. the ranking metric, as the learning process is more firmly tied to the ranking objective. However, direct optimization of the ranking metric is problematic in practice. Evaluation metrics such NDCG@K and R@K are defined in terms of the resulting ranking, not the scoring functions that induce these rankings. Hence, the ranking metrics are discontinuous (and thus non-differentiable) w.r.t. the parameters of the scoring functions, which in turn means that we cannot apply continuous optimization techniques such as gradient-based methods. One solution to this problem is to construct a continuous approximation of the ranking metric [28], whose optimization will indirectly improve the actual objective. Another solution is to appeal to the idea of an *implicit* ranking metric, called *LambdaRank* [3]. The strategy consists in estimating the parameters of a (differentiable) scoring function while implicitely accounting for the ranking metric. This is achieved by weighting parameter updates proportionally to changes in the ranking metric. We pursue the approach of LambdaRank in the present paper, as discussed at greater length in the next section.

5. RANKING LOSS MINIMIZATION

In this section, we present our regularized formulation of the problem and discuss the different optimization strategies we use. We divide our discussions in two parts, first exposing a convex approach suitable for stochastic optimization strategies and then describing non-convex alternatives. The regularizer we choose leads to sparse solutions to ensure low-latency scoring functions with minimal computation time and memory footprint.

For a given ranking evaluation metric $\mathscr{M}(\boldsymbol{r}, \boldsymbol{l})$, we introduce the delta function $\Delta_{\mathscr{M}(\mathrm{r},\mathrm{l})}$, which is given by:

$$\Delta_{\mathscr{M}(\mathrm{r},\mathrm{l})}(i,j) = \mathscr{M}(\boldsymbol{r}, \boldsymbol{l}) - \mathscr{M}(\boldsymbol{r}_{i/j}, \boldsymbol{l}), \tag{3}$$

where $\boldsymbol{r}_{i/j}$ is the ranking we would obtain by swapping the positions of product i and j in $\boldsymbol{r}$. Intuitively, the delta function $\Delta_{\mathscr{M}(\mathrm{r},\mathrm{l})}$ can be thought of as measuring the importance of sorting i and j correctly in order for $\boldsymbol{r}$ to maximize the value of $\mathscr{M}(\boldsymbol{r}, \boldsymbol{l})$.

Further, let $\mathcal{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ be the features associated to the products in the (unsorted) list and suppose that product i is scored by some parametric function $\varphi(\boldsymbol{x}_i)$. We define the ranking loss as follows:

$$\ell_{\mathscr{M}(\boldsymbol{r},\boldsymbol{l})}(\mathcal{X}; \varphi) = \sum_{r_i \leqslant r_j} \Delta_{\mathscr{M}(\mathrm{r},\mathrm{l})}(i,j) \cdot \mathscr{P}(\varphi(\boldsymbol{x}_i), \varphi(\boldsymbol{x}_j)), \tag{4}$$

where $\mathscr{P}(x, y)$ is a pairwise loss term. Possible choices for the pairwise loss include the hinge loss, $\mathscr{P}(x, y) = \max\{0, y - x + \epsilon\}$ (for some slack parameter $\epsilon \geqslant 0$), or the logistic loss, $\mathscr{P}(x, y) = \log(1 + \exp(x - y))$.

The loss defined in (4) can be understood as follows. If the score of i is higher (or marginally lower) than the score of j, then little loss (or no loss) is incurred. However, if the score of j is higher than the score of i, then the pairwise loss is weighted by $\Delta_{\mathscr{M}(\mathrm{r},\mathrm{l})}(i,j)$, which is the cost of not ranking i and j correctly.

Given a training sample, that is, a list of products to be ranked together with relevance labels, we can learn the parameters in φ by minimizing the ranking loss $\ell_{\mathscr{M}}$ over that sample. This task can be accomplished via standard (sub) gradient descent methods. Any choice for the parametric form of φ, ranging from a simple linear regression to a multilayer perceptron, will be compatible with the ranking metric we are adopting as long as we are able to compute the corresponding (sub)gradients. To satisfy the requirements of our web-scale application, we will restrict ourselves to linear scoring functions. The score of a feature vector $\boldsymbol{x}_i$ is thus given by $\varphi(\boldsymbol{x}_i) = \boldsymbol{w}^\intercal \boldsymbol{x}_i$. This choice preserves the convexity of the ranking loss defined in (4), which is a convenient property for the optimization stage we discuss next.

5.1 Convex approach

In order to induce sparsity in the learned scoring functions, i.e., to zero out some components in the weight vector $\boldsymbol{w}$, we add a ℓ_1-regularization term to the loss defined in (4). This regularized objective is convex (since ℓ_1 is a norm) with, both, theoretical guarantees and efficient algorithmic schemes (e.g., see [26] and references therein). As advocated in large-scale settings (e.g., [24] in the context of online advertising), we additionally consider a squared ℓ_2-term, thus leading to the following regularized ranking loss:

$$\ell^*_{\mathscr{M}(\boldsymbol{r},\boldsymbol{l})}(\mathcal{X}; \varphi_{\boldsymbol{w}}) = \ell_{\mathscr{M}(\boldsymbol{r},\boldsymbol{l})}(\mathcal{X}; \varphi_{\boldsymbol{w}}) + \lambda_1 \|\boldsymbol{w}\|_1 + \frac{1}{2}\lambda_2 \|\boldsymbol{w}\|_2^2, \tag{5}$$

where λ_1 and λ_2 are the nonnegative hyperparameters determining, respectively, the weight of the ℓ_1 and ℓ_2 penalties within the overall loss. Note that the resulting regularization is usually referred to as elastic-net [35]. In the remainder of the paper, we will refer to our listwise ranking model as *ElasticRank*.

Our training set is composed of several lists of products $\{\boldsymbol{l}^{(j)}, \boldsymbol{r}^{(j)}, \mathcal{X}^{(j)}\}$, where $j = 1 \ldots m$, so that instead of minimising (5), we seek to optimise the *averaged* regularized problem, which is given by:

$$\min_{\boldsymbol{w}} \left\{ \frac{1}{m} \sum_{j=1}^{m} \ell_{\mathscr{M}(\boldsymbol{r}^{(j)}, \boldsymbol{l}^{(j)})}(\mathcal{X}^{(j)}; \varphi_{\boldsymbol{w}}) + \lambda_1 \|\boldsymbol{w}\|_1 + \frac{1}{2}\lambda_2 \|\boldsymbol{w}\|_2^2 \right\}. \tag{6}$$

In our web-scale application, m is typically in the order of 10^6. We therefore resort to stochastic optimization tools [2] to solve (6), where the update rule to get the next iterate is based solely on a *single* randomly drawn ranked list. The literature dedicated to the stochastic optimization of functions of the form of (6) is vast (e.g., [9,10,14,23,33]), and we shall focus next on two representative options. Note that we have not considered the recently proposed incremental-gradient schemes (e.g., see [8] and references therein) well-suited to the optimization of (6) because of the their prohibitive memory footprint.

Here, we consider two algorithms that can be implemented efficiently: the update for a single instance is *linear in the number of non-zero features* and independent of the total number of features. In many settings the data are described by a large number of features, but in each individual example only a small subset of these are non-zero. For instance, in text-based applications the number of features corresponds to size of the vocabulary (i.e., the number of unique tokens in the corpus), while each document contains only a small fraction of all the words in the vocabulary. Similarly, one-hot representation of categorical variables that can take many values results in a large number of binary features (as described in Section 8), but typically in each example only one feature is active. Thus, to handle large high-dimensional data sets it is crucial to exploit the sparsity of the examples and perform only the minimum number of operations in each iteration.

Forward-Backward Splitting (FOBOS) [10], on the one hand, solves the regularized optimization problem by alternating between two phases: taking a simple gradient step followed by a proximal step that involves the elastic-net regularization. More specifically, we first perform an unconstrained gradient step:

$$\widetilde{\boldsymbol{w}}_t = \boldsymbol{w}_t - \eta_t \nabla_{\boldsymbol{w}} \ell_{\mathscr{M}(\boldsymbol{r},\boldsymbol{l})}(\mathcal{X}; \varphi_{\boldsymbol{w}}), \tag{7}$$

where η_t is the learning rate at iteration t. Subsequently, we take a proximal step based on the elastic-net regularization (e.g., see Section 3.3 of [1]):

$$w_{t+1,i} = \begin{cases} 0 & \text{if } |\widetilde{w}_{t,i}| \leq \eta_t \lambda_1, \\ \frac{1}{1+\eta_t \lambda_2}(\widetilde{w}_{t,i} - \operatorname{sgn}(\widetilde{w}_{t,i}) \eta_t \lambda_1) & \text{otherwise.} \end{cases}$$

Note that the regularization hyperparameter λ_1 is scaled by the learning rate η_t, thus inducing more regularization at earlier iterations.

Regularized Dual Averaging (RDA) [33], on the other hand, adjusts the learning variables by solving a proximal step that involves the running average of *all past gradients* of the loss functions instead of taking a gradient step at each iteration. More formally, the exponentially weighted average is first updated as:

$$\bar{\mathbf{g}}_t = \frac{t-1}{t} \bar{\mathbf{g}}_{t-1} + \frac{1}{t} \nabla_{\boldsymbol{w}} \ell_{\mathscr{M}(\boldsymbol{r},\boldsymbol{l})}(\mathcal{X}; \varphi_{\boldsymbol{w}}).$$

Next, the weights are adjusted as follows:

$$w_{t+1,i} = \begin{cases} 0, & \text{if } |\bar{g}_{t,i}| \leq \lambda_1, \\ -\frac{1}{\lambda_2 + \eta_t}(\bar{g}_{t,i} - \operatorname{sgn}(\bar{g}_{t,i})\lambda_1) & \text{otherwise.} \end{cases}$$

Parameter t is the iteration number and parameter η_t the learning rate. We refer the interested readers to [23] for a in-depth discussion about the connections between FOBOS and RDA.

5.2 Non-convex approach

There exists a variety of approaches similar to (6) where, either a non-convex penalty, or some heuristics are used in lieu of the convex ℓ_1 term to promote sparsity. We discuss below two possible options.

As proposed in [19], we can simply enforce sparsity by adding a pruning operation to the l_2-regularized stochastic gradient descent (SGD) update rule. The pruning, scheduled every k gradient steps, simply consists of setting to 0 all the weights w_i such that, for a chosen threshold θ, $|w_i| < \theta$. We refer to this (somehow naïve) strategy as *pruned SGD* (PSGD). Clearly, the higher the value we choose for θ, the sparser the model will get as a result. Despite its simplicity, our experiments show that PSGD applied to (6) with $\lambda_1 = 0$ compares remarkably well to the more sophisticated convex approaches discussed previously.

A more elaborate variation built upon this idea is provided by Truncated Gradient Descent (TGD) [19], which truncates the solutions obtained by standard SGD after every k iterations based on the ℓ_1 norm. It is worth noting that if we restrict FOBOS to only use the ℓ_1 penalty, we end up with an algorithm that is a special case of TGD (as detailed in [19]). In particular, the technique used in [31] is exactly this special variant of TGD, and hence, is equivalent to FOBOS without the ℓ_2 regularization.

6. BAYESIAN OPTIMIZATION OF THE HYPERPARAMETERS

The family of ranking models we are focusing on is parameterised by several hyperparameters. Our formulation depends on the careful choice of the weighting scheme $\Delta_{\mathscr{M}(\mathrm{r},\mathrm{l})}$, the pairwise loss function $\mathscr{P}$, the regularization parameters for both the ℓ_1 and ℓ_2 penalties, and the choice of the online optimization algorithm itself.

While some of these hyperparameters could be tuned *separately* based on tailored heuristics (e.g., the ℓ_1-regularization parameter via homotopy techniques [27]), it becomes much more challenging to optimize them all jointly in a scalable fashion. To this end, we resort to recent progress made in the field of Bayesian optimization applied to the automatic tuning of machine-learning hyperparameters [15, 30]. Bayesian optimization is well-suited to carry out the optimization of unknown black-box functions whose evaluation is expensive, which, in our context, corresponds to the entire ranking loss minimization pipeline.

In a nutshell, this class of methods works as follows: while collecting (possibly noisy) observations over the course of the optimization, a probabilistic model of the unknown black-box function is being constructed and maintained. The next point to evaluate is determined based on the posterior distribution of the probabilistic model combined with some surrogate acquisition function, such as the expected improvement [17]. This process is then continued until some predefined budget gets consumed, e.g., expressed as a maximum number of black-box function evaluations. In our settings, we typically run the optimization for 30 evaluations. Note that this hyperparameter-tuning step is made possible thanks to the sufficiently low training- and evaluation-time requirements of our ranking model.

As advocated in [30], for our probabilistic engine we consider a Gaussian process with a 5/2-Matérn covariance function and automatic relevance determination. Moreover, regarding the choice of the acquisition function to drive the optimization, we found that Thompson sampling worked best at trading off exploration and exploitation (e.g., as also recently reported in [13]). Given that our space of hyper-parameters is relatively small (two numerical and three categorical hyper-parameters), we observed that a simple empirical Bayes approach performed well to handle the covariance parameters of the Gaussian process.

It is lastly worth mentioning that in our industrial environment, the different development and deployment phases of our ranking package imply that customer teams with various skills, in particular, without machine-learning expertise, should have access to a simple turn-key interface. Moreover, this interface should at the same time be easy to extend to additional instantiations of the loss family (4). We have found that Bayesian optmization is a good choice in these respects.

7. DOCUMENT RETRIEVAL BENCHMARK

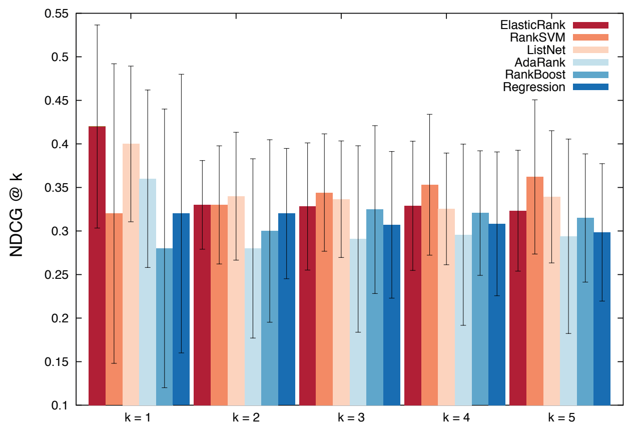
In this section, we conduct experiments on benchmark data sets for text document retrieval. In the next section, we will return to our use case and report results on large-scale experiments with Amazon data.

The goal of the first set of experiments is to assess how the proposed approach compares to a selection of state-of-the-art learning-to-rank models in terms of predictive performance. We focus on the LETOR 3.0 benchmark collection, which provides easy access to a number of authoritative baselines in terms of different evaluation metrics.[1] In particular, we provide results for TD2003, TD2004, and OHSUMED data sets. TD2003 contains 49,058 query-document pairs. Each pair is described by a 64-dimensional, pre-extracted feature vector. The total number of unique queries is 50, and for each query, the corresponding list of results is labeled by a binary relevance judgment for the document. TD2004 has the same number of features and labeling structure as TD2003, except that there are 75 queries overall, i.e., a total of 74,146 query-document pairs. Finally, OHSUMED contains 16,140 query-document examples, described by 45 attributes, for a total number of 106 queries. The relevance judgments are organized on a 3-graded integer scale.

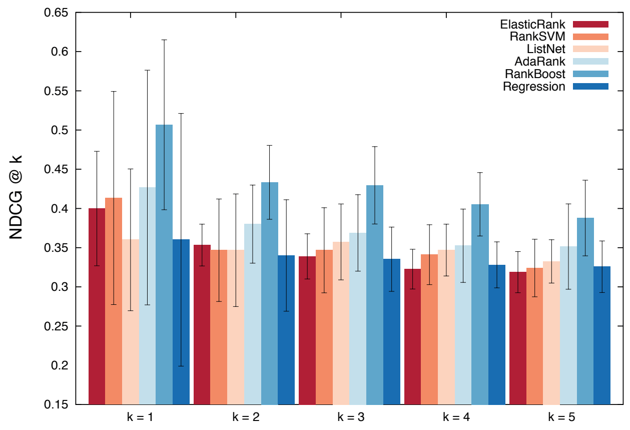
For ease of exposition, our approach is referred to as *ElasticRank*. We compare ElasticRank based on RDA to a selection of alternative methods in the pointwise, pairwise and listwise families, including both linear and non-linear estimators (see Section 4): RankSVM [12, 16], RankBoost [11], ListNet [6], AdaRank-NDCG [34], and simple linear regression. The results are reported in Figure 2. They were obtained based on 5 random splits into train and test.

Overall, ElasticRank is competitive w.r.t. state-of-the-art learning-to-rank models, while only performing a single pass on the data. The other models are allowed to run as many iterations as needed in order to converge to optimal performance. On OHSUMED (Figure 2c), ElasticRank is achieving the highest performance (for $NDCG$@1–5) and on TD2003

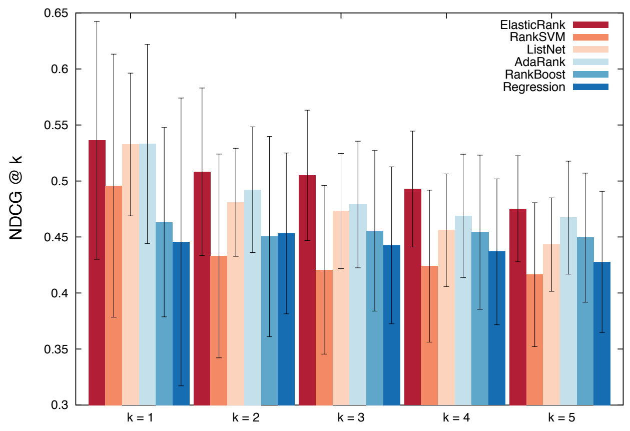
[1]Documentation available at `http://research.microsoft.com/en-us/um/beijing/projects/letor/default.aspx`.



(a) TD2003



(b) TD2004



(c) OHSUMED

Figure 2: Predictive performance in terms of NDCG@K measured on three different data sets from the LETOR 3.0 collection: (a) TD2003, (b) TD2004 and (c) OHSUMED. Results are averaged based on 5-fold cross-validation. ElasticRank is competitive w.r.t. the other models, while only performing a single pass on the training data.

Metric	**R@1**	**R@2**	**R@3**	**R@4**	**R@5**	**NDCG**
LR	92.30%	57.27%	35.68%	23.17%	12.83%	18.67%
RankSVM	124.54%	75.36%	47.71%	32.04%	19.80%	25.99%
PSGD	128.73%	**77.21%**	**48.62%**	**32.74%**	**20.00%**	**26.64%**
FOBOS	128.27%	75.73%	48.39%	32.04%	19.16%	26.16%
RDA	**129.03%**	76.26%	48.37%	32.08%	19.13%	26.33%

Table 1: Highest R@K achieved by each one of the considered algorithms (over the test set) after one training pass. R@K is reported in terms of relative improvement (in percentage) over the values achieved by ranking the recommendations uniformly at random. ElasticRank (listwise) is more accurate than RankSVM (pairwise), which in turn outperforms LR (pointwise).

(Figure 2a) it performs best in $NDCG$@1. The performance obtained on TD2004 is less satisfying (Figure 2b), although for $NDCG$@1–2 ElasticRank is still outperforming ListNet and RankSVM. The best results for TD2004 are achieved by RankBoost, which suggests that non-linearities are important here. Unfortunately, nonlinear methods usually do not scale and, in practice, only linear methods are feasible. More data also leads to larger feature sets, which in turn, provide a way to compensate for the absence of nonlinearities.

8. PRODUCT RECOMMENDATION ON AMAZON.COM

In this section, we evaluate ElasticRank on a data set collected on the Amazon retail website. The task is to learn ranking models based on purchase logs. The results we report are based on data collected for a widget similar to the one described in Section 2. We cannot name the actual widget for confidentiality reasons.

We focus on the following questions, which are crucial to provide practical guidance when considering massive, low-latency ranking problems:

i. Which regularization technique leads to the highest predictive performance?

ii. In the one-pass setting, which sparsity-inducing technique is best at trading-off performance for sparsity?

iii. How large is the impact of the number of training passes w.r.t. the quality of the learned models?

In order to run our experiments, we sampled a set of impression log data from a contiguous time interval, and used $\frac{9}{11}$ of the data for training and $\frac{1}{11}$ for validation and testing respectively, leaving the temporal order intact. The sample contains millions of impressions, where an impression is simply one of the items displayed within the widget as a recommendation tied to a particular context. Note that this size is larger than the average size of the LETOR benchmarks by several orders of magnitude. Each item is described by a high-dimensional feature vector, including boolean, numerical, and categorical features. In order to protect sensitive information, we cannot disclose the exact number of features available from our logs, their actual semantics or the way they are engineered. For the purpose of the present experiment we consider up to a few thousand features. We convert categorical features into numerical ones using a sparse, one-hot representation, which may generate hundreds of thousands or even millions of dimensions. We scale the feature values in a listwise fashion, so that the minimum and maximum values of each feature are, respectively, 0 and 1 for each of the processed impression lists. Relevance labels are derived from purchase decisions made by the customers.

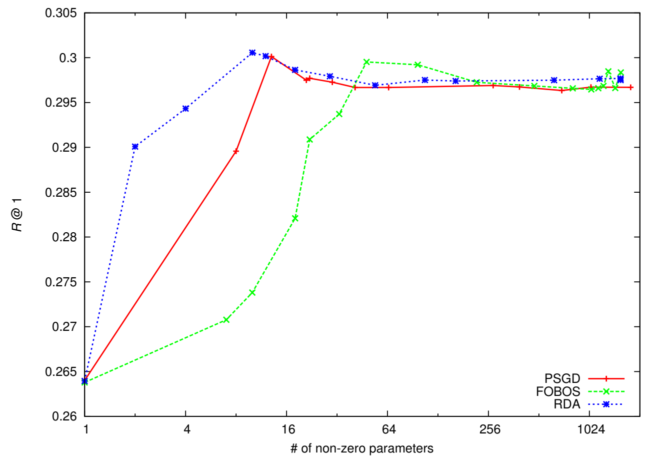
Table 1 collects the highest values obtained by ElasticRank in the one-pass setting, both for R@K and NDCG@K. The algorithms described in Section 5, namely RDA, FOBOS and PSGD, are compared to one another. We further compare to a number of alternative ranking models, respectively logistic regression (LR), which falls into the pointwise family, and RankSVM, which falls into the pairwise family.

From the results listed in the table, we can draw at least two conclusions. First, ElasticRank (abstracting from the choice of the specific regularization technique) is more accurate than both LR and RankSVM. The fact that RankSVM performs slightly worse should not be a surprise as it can be seen as optimizing a special case of the objective stated in (4). Indeed, RankSVM simply sets the delta function to 1 and considers a hinge loss. Second, the three variants of ElasticRank are all pretty close to one another in terms of achieved performance. This answers question (i) above.

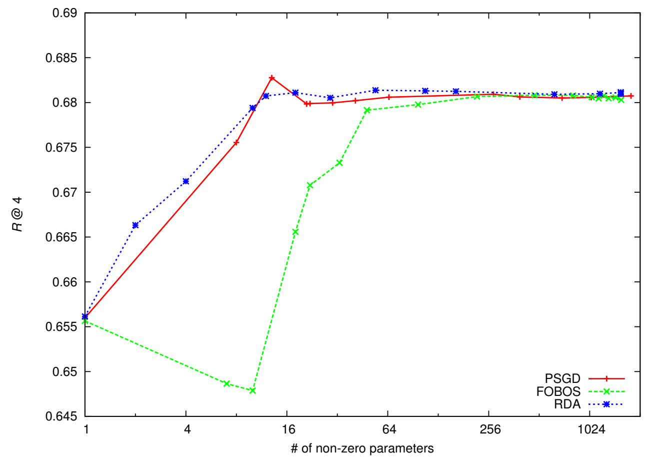
Figure 3 shows the predictive performance of the models learned by different sparsity-inducing formulations, measured by R@1 and R@4, as a function of the number of non-zero parameters. The plots are obtained by varying the regularization parameter (λ_1 or θ depending on the algorithm). We report results for $K = 1$ and $K = 4$ to show that the observed trends are not an artifact of the specific values of K. We only report result for R@K because this evaluation metric matches our use case better and it is therefore preferred over NDCG@K. Note, however, that we observed similar trends with the NDCG@K-based delta function, but the results were slightly worse than for R@K-based one (which was not the case in the LETOR experiments).

Next, let us focus on question (ii). RDA allows us to smoothly increase the sparsity induced in the model while keeping the model performance under control. In fact, when regularization becomes too aggressive, the ranking quality decreases with a trend which is seemingly monotonic in the growth of λ_1. This relatively smooth decay in ranking quality is what ensures the model sparsity can be safely adapted to the latency requirements of the application without the risk of missing the optimal sparsity/performance trade-off. Interestingly, such smoothness is also observed for PSGD, which is a simple way of reducing the model latency. On the contrary, the curves for FOBOS are pretty bumpy, which means we could get stuck in suboptimal solutions whenever we cannot afford to explore a wide range of λ_1 values.

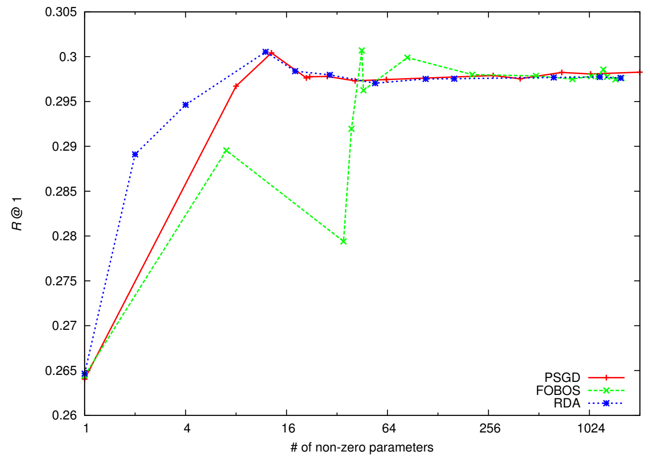
As a sanity check whether elastic net helps, we also tested the version of TGD adopted in [31], which corresponds to



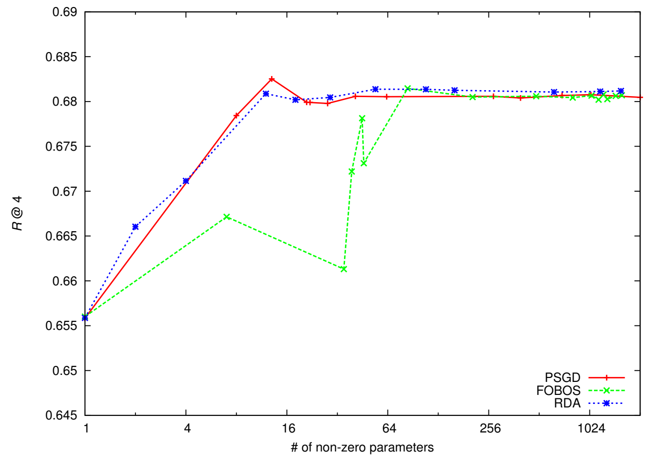
(a) Recall@1, one pass



(b) Recall@4, one pass



(c) Recall@1, two passes



(d) Recall@4, two passes

Figure 3: Predictive performance of the three algorithms: PSGD, FOBOS, and RDA in terms of Recall@1 (left) and Recall@4 (right), over one (top) and two (bottom) passes. RDA and PSGD are more reliable than FOBOS at trading-off model sparsity for predictive performance.

FOBOS without the ℓ_2 penalty. The resulting curves were less stable and less accurate than those reported for FOBOS.

Independently of these stability considerations, Elastic-Rank based on RDA or PSGD converges faster to the high values regime as a function of model size (i.e., the number of non-zero parameters). In other words, RDA and PSGD enable us to pick more informative features than the ones selected by FOBOS. This is important, because when latency is a mission-critical requirement, we might have to restrict the maximal number of features a priori.

Next, we turn our attention to question (*iii*). Letting the algorithm run for more than one pass over the training data is mostly useful for the algorithm that is slowest in converging to its highest value (i.e., FOBOS). We only show results for up to 2 training passes as additional passes over the data did not improve the quality of the learned models.

We end this section by analyzing the prediction latency, i.e., the time it takes for our model to compute the score of candidate recommendations as a function of the model sparsity. We benchmarked the scoring functions learned over the runs plotted in Figure 3 by averaging the time to score approximately 5 million examples. The results are reported in Table 2.

# of weights	Latency	TPS
4	0.0062 ms	≈ 161,290
29	0.0087 ms	≈ 114,942
1804	0.0109 ms	≈ 91,743

Table 2: Average latency per recommendation and total throughput per second (TPS, i.e., number of scored impressions) for models of decreasing sparsity.

The latency/throughput requirements in web-scale applications can be extremely strict. It is crucial to enforce the desired level of sparsity when learning a ranking model as too many non-zeros can rapidly lead to exceeding these requirements due to the amount of traffic. For example, to meet the requirements of a latency smaller than 0.009 ms, one would not be able to afford more than 29 parameters.

The latency due to the computation of the scoring function might seem very small and it can be noted that it does not grow linearly with the size of the model. However, features are rarely readily available in real-time applications. Even if the features are pre-computed, they might have to be read from disk or transmitted over the network. The time required to access the features is much longer than the time required to process them with the scoring function. Hence, and even though this claim is not fully illustrated here in the time needed to compute the scores alone, model sparsity may be even more impactful when it comes to web-scale deployment of the ranking system.

9. CONCLUSIONS

The core question motivating this work was whether learning-to-rank methods are able to cope with the scale of a global e-commerce platform. In particular, we asked whether the ranking quality, which is typically demonstrated on public learning-to-rank benchmarks, is still achievable when the challenges in terms of training time and memory requirements, as well as prediction latency and throughput, are pushed to their limits. Our results suggest that web-scale efficiency requirements can be met without sacrificing predictive performance. Surprisingly enough, such goal is exemplarily achieved by just sticking to the simplest options available. First, we obtain state-of-the-art results by using a *linear* scoring function as the basic modeling tool. Second, making only *one-pass* over the training data is sufficient for SGD-type algorithms to converge to fairly accurate solutions. Third, *naively pruning* the estimated model weights at regular training intervals leads to models which are nearly as sparse and accurate as those delivered by the best-behaved ℓ_1-based regularization techniques, such as elastic net.

10. ACKNOWLEDGMENTS

We are grateful to Mitchell Goodman, Vijai Mohan, and JJ Tavernier for helping us geting a grip on the log data and model deployment infrastructure at Amazon. We are also indebted to Ralf Herbrich for providing input and feedback on model design.

11. REFERENCES

[1] F. Bach, R. Jenatton, J. Mairal, and G. Obozinski. Optimization with sparsity-inducing penalties. *Foundations and Trends in Machine Learning*, 4(1):1–106, 2011.

[2] L. Bottou and Y. LeCun. Large scale online learning. In *Advances in Neural Information Processing Systems*, volume 16, pages 217–224, 2004.

[3] C. J. C. Burges, R. Ragno, and Q. V. Le. Learning to rank with nonsmooth cost functions. In *Advances in Neural Information Processing Systems (NIPS)*, pages 193–200, 2006.

[4] C. J. C. Burges, K. M. Svore, P. N. Bennett, A. Pastusiak, and Q. Wu. Learning to Rank Using an Ensemble of Lambda-Gradient Models. In *Proceedings of the Yahoo! Learning to Rank Challenge, held at ICML 2010, Haifa, Israel, June 25, 2010*, pages 25–35, 2011.

[5] S. Büttcher, C. L. Clarke, and G. V. Cormack. *Information Retrieval: Implementing and Evaluating Search Engines*. MIT Press, 2010.

[6] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th International Conference on Machine learning (ICML 2007)*, pages 129–136, New York, NY, USA, 2007. ACM.

[7] B. Croft, D. Metzler, and T. Strohman. *Search Engines: Information Retrieval in Practice*. Addison-Wesley, Boston (MA), 2009.

[8] A. Defazio, F. Bach, and S. Lacoste-Julien. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. Technical report, preprint arXiv:1407.0202, 2014.

[9] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159, 2011.

[10] J. C. Duchi and Y. Singer. Efficient online and batch learning using forward backward splitting. *Journal of Machine Learning Research*, 10:2899–2934, 2009.

[11] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer. An Efficient Boosting Algorithm for Combining Preferences. *Journal of Maching Learning Research*, 4:933–969, 2003.

[12] R. Herbrich, T. Graepel, and K. Obermayer. Large Margin Rank Boundaries for Ordinal Regression. In Smola, Bartlett, Schölkopf, and Schuurmans, editors, *Advances in Large Margin Classifiers*, chapter 7, pages 115–132. MIT Press, 2000.

[13] M. Hoffman, B. Shahriari, and N. de Freitas. On correlation and budget constraints in model-based bandit optimization with application to automatic machine learning. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 365–374, 2014.

[14] C. Hu, J. T. Kwok, and W. Pan. Accelerated gradient methods for stochastic optimization and online learning. In *Advances in Neural Information Processing Systems*, 2009.

[15] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proceedings of LION-5*, page 507-523, 2011.

[16] T. Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 133–142, New York, NY, USA, 2002. ACM.

[17] D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

[18] H. Lai, Y. Pan, C. Liu, L. Lin, and J. Wu. Sparse Learning-to-Rank via an Efficient Primal-Dual Algorithm. *IEEE Transactions on Computers*, 62:1221–1233, 2013.

[19] J. Langford, L. Li, and T. Zhang. Sparse Online Learning via Truncated Gradient. *Journal of Machine Learning Research*, 10:777–801, 2009.

[20] L. Laporte, R. Flamary, S. Canu, S. Déjean, and J. Mothe. Nonconvex Regularizations for Feature

Selection in Ranking With Sparse SVM. *IEEE Trans. Neural Netw. Learning Syst.*, 25(6):1118–1130, 2014.

[21] T.-Y. Liu. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3(3):225–331, 2009.

[22] C. D. Manning, P. Raghavan, and H. Schutze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[23] B. H. McMahan. Follow-the-Regularized-Leader and Mirror Descent: Equivalence Theorems and L1 Regularization. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011*, pages 525–533, 2011.

[24] H. B. McMahan, G. Holt, D. Sculley, M. Young, D. Ebner, J. Grady, L. Nie, T. Phillips, E. Davydov, D. Golovin, et al. Ad click prediction: a view from the trenches. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1222–1230. ACM, 2013.

[25] A. Mohan, Z. Chen, and K. Q. Weinberger. Web-Search Ranking with Initialized Gradient Boosted Regression Trees. In *Yahoo! Learning to Rank Challenge*, pages 77–89, 2011.

[26] S. Negahban, P. Ravikumar, M. J. Wainwright, and B. Yu. A unified framework for high-dimensional analysis of M-estimators with decomposable regularizers. In *Advances in Neural Information Processing Systems*, 2009.

[27] M. Y. Park and T. Hastie. L1-regularization path algorithm for generalized linear models. *Journal of the Royal Statistical Society. Series B*, 69(4):659–677, 2007.

[28] T. Qin, T.-Y. Liu, and H. Li. A general approximation framework for direct optimization of information retrieval measures. *Information Retrieval*, 13(4):375–397, 2010.

[29] D. Sculley. Large scale learning to rank. In *NIPS Workshop on Advances in Ranking*. 2009.

[30] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, pages 2960–2968, 2012.

[31] Z. Sun, T. Qin, Q. Tao, and J. Wang. Robust sparse rank learning for non-smooth ranking measures. In *Proceedings of the 32nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2009, Boston, MA, USA, July 19-23, 2009*, pages 259–266. ACM, 2009.

[32] J. Weston, S. Bengio, and N. Usunier. WSABIE: Scaling Up to Large Vocabulary Image Annotation. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pages 2764–2770, 2011.

[33] L. Xiao. Dual Averaging Methods for Regularized Stochastic Learning and Online Optimization. *Journal of Machine Learning Research*, 11:2543–2596, 2010.

[34] J. Xu and H. Li. Adarank: A boosting algorithm for information retrieval. In *SIGIR '07: Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 391–398, New York, NY, USA, 2007. ACM.

[35] H. Zou and T. Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society. Series B*, 67(2):301–320, 2005.