ONE-PASS RANKING MODELS FOR LOW-LATENCY PRODUCT RECOMMENDATIONS

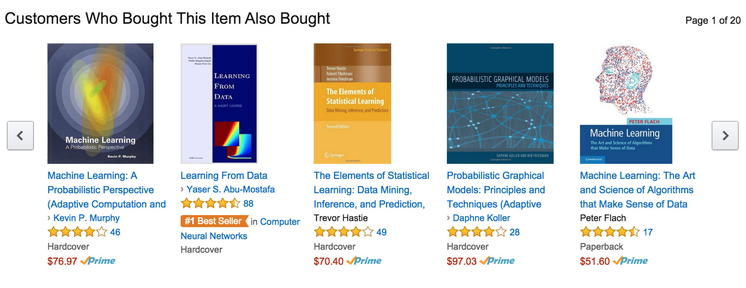
Antonino Freno, Martin Saveski, Rodolphe Jenatton, Cédric Archambeau – Amazon Machine Learning Team – Berlin, Germany

antonino.freno@zalando.de, msaveski@mit.edu, jenatton@amazon.com, cedrica@amazon.com

ABSTRACT. We present a web scale recommender system that has a *small memory footprint*, *fast training time*, and *low prediction latency*. We show that by using stochastic optimization with only a single pass over the data and sparsity constraints, we can efficiently learn ranking functions without sacrificing accuracy.

PRODUCT RECOMMENDATIONS

The picture below shows the "*Customers Who Bought This Item Also Bought*" widget shown on all product pages in AMAZON.com. The goal of this widget is to provide accurate recommendations and help users find relevant items. In this study, we are interested in using purchase logs to learn more accurate ranking models and improve the quality of product recommendations.



A variety of deployment constraints make the applicability of machine learning algorithms challenging:

1. The scale of the data: large number of examples, large number of features
2. Constant change in customer's preferences: requires constant retraining
3. Making predictions in real time: to provide better customer experience

We address these challenges by making the following design decisions:

1. Stochastic optimization
2. Single-pass learning
3. Exploiting sparsity

LOSS FUNCTION

- Ranking metrics (like NDCG) are hard to optimize explicitly, since they are discontinuous.
- *LambdaRank* [1] estimates the model parameters while *implicitly* accounting for the ranking metric.
- Parameter updates are weighted proportionally to changes in the ranking metric.

For a given ranking evaluation metric $\mathscr{M}(\boldsymbol{r}, \boldsymbol{l})$, we introduce delta function:

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

($\boldsymbol{r}_{i/j}$ is the ranking obtained by swapping the positions of product *i* and *j* in $\boldsymbol{r}$.) $\Delta_{\mathscr{M}(\mathrm{r,l})}(i,j)$ measures the importance of sorting *i* and *j* correctly.

Let $\mathcal{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ be the features associated to the products and $\varphi(\boldsymbol{x}_i)$ be a parametric function. We define the ranking loss as follows:

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

Possible choices of $\varphi(\boldsymbol{x}_i)$ include:

$$\mathscr{P}(x,y) = \max\{0, y - x + \epsilon\} \qquad \mathscr{P}(x,y) = \log(1 + \exp(x - y))$$

- If $\varphi(\boldsymbol{x}_i) > \varphi(\boldsymbol{x}_j)$ => little loss is incurred
- If $\varphi(\boldsymbol{x}_i) < \varphi(\boldsymbol{x}_j)$ => the pairwise loss is weighted by $\Delta_{\mathscr{M}(\mathrm{r,l})}(i,j)$

ELASTIC RANK

To handle large high-dimensional datasets it is crucial to exploit sparsity and perform only minimum number of operations per update.

We add:
- *L1*-regularization term to enforce sparsity,
- *L2*-regularization term to improve convergence and add strong convexity.

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

We consider three optimization algorithms for which an update for a single instance is *linear* in the number of non-zero features.

FOBOS: Forward-Backward Splitting [2], solves the regularized optimization problem by alternating between two phases:

1. Taking a simple gradient step:

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

2. Taking a proximal step that involves the elastic-net regularization:

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

where η_t is learning rate.

RDA: Regularized Dual Averaging [3], solves a proximal step involving the running average of all past gradients of the loss functions

- The exponentially weighted average is first updated as:

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

- The weights are adjusted as follows:

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

pSGD: Pruned Stochastic Gradient Descent, enforces sparsity by simply adding a pruning operation to the L2-regularized SGD update rule.

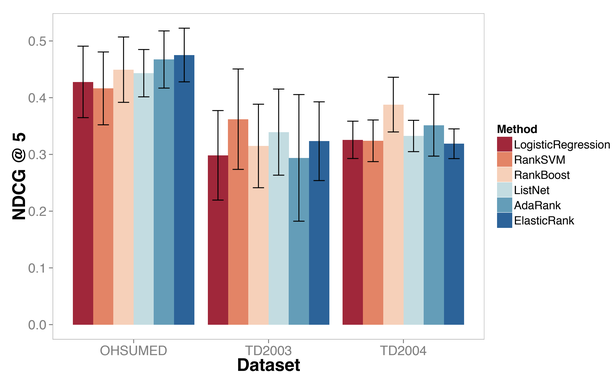
- Every *k* steps we set:

$$|w_i| < \theta \Rightarrow w_i = 0$$

EXPERIMENTS

Goal: Assess how ElasticRank compares to state-of-the-art ranking models

Dataset: LETOR 3.0 benchmark collection

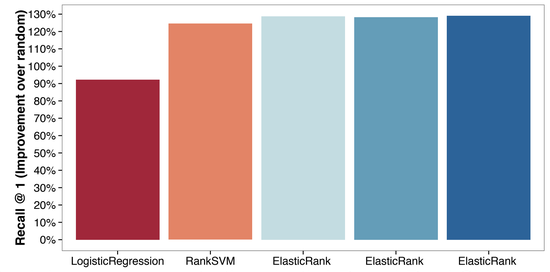


RECOMMENDATIONS ON AMAZON.COM

Next, we evaluate ElasticRank on a dataset collected on the Amazon retail website. To run the experiments, we sampled a set of impression log data from a contiguous time interval, and used 9/11 of the data for training and 1/11 for validation and testing, leaving the temporal order intact.

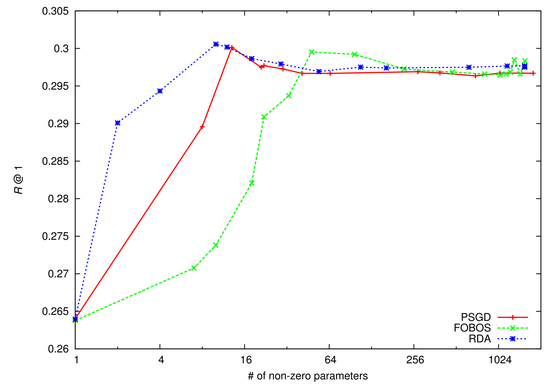
- Number of examples ~ Millions
- Number of features ~ Thousands (leading to millions of dimensions)

Perfomance Comparison



- ElasticRank is more accurate than both LR and RankSVM.
- The three variants of ElasticRank achieve similar performance.

Sparsity vs Performance Trade-off



- RDA and pSGD allow us to smoothly increase the sparsity induced, while keeping the model performance under control.
- FOBOS achives less optimal trade-off and is less stable.

Latency

We benchmarked the scoring functions learned by averaging the time to score approximately 5 million examples:

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

This experiment doesn't capture the time required to precompute/access the features, which may be much longer than the time needed to compute the scoring function. Hence, aside from 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.

References

1. C. J. C. Burges, R. Ragno, and Q. V. Le. "Learning to rank with nonsmooth cost functions". Advances in Neural Information Processing Systems (NIPS), 2006.
2. J. C. Duchi and Y. Singer. "Efficient online and batch learning using forward backward splitting". Journal of Machine Learning Research, 2009.
3. L. Xiao. "Dual Averaging Methods for Regularized Stochastic Learning and Online Optimization". Journal of Machine Learning Research, 2010.