Large Scale Learning to Rank

D. Sculley
Google, Inc.
dsculley@google.com

Abstract

Pairwise learning to rank methods such as RankSVM give good performance, but suffer from the computational burden of optimizing an objective defined over $O(n^2)$ possible pairs for data sets with n examples. In this paper, we remove this super-linear dependence on training set size by sampling pairs from an implicit pairwise expansion and applying efficient stochastic gradient descent learners for approximate SVMs. Results show orders-of-magnitude reduction in training time with no observable loss in ranking performance. Source code is freely available at: http://code.google.com/p/sofia-ml

1 Introduction: The Problem with Pairs

In this paper, we are concerned with learning to rank methods that can learn on large scale data sets. One standard method for learning to rank involves optimizing over the set of pairwise preferences implicit in the training data. However, in the worst case there are $\binom{n}{2}$ candidate pairs, creating the potential for a quadratic dependency on the size of the data set. This problem can be reduced by sharding the data (*e.g.* by query) and restricting pairwise preferences to be valid only within a given shard, but this still results in super-linear dependencies on n. Even with efficient data structures for special cases this leads to $O(n \log n)$ computation cost for training [8]. Applying such methods to internet-scale data is problematic.

Bottou *et al*. have observed that in large-scale settings the use of stochastic gradient descent methods provide extremely efficient training with strong generalization performance – despite the fact that stochastic gradient descent is a relatively poor optimization strategy [2, 1]. Thus, the main approach of this paper is to adapt the pairwise learning to rank problem into the stochastic gradient descent framework, resulting in a scalable methodology that we refer to as Stochastic Pairwise Descent (SPD). Our results show that best performing SPD methods provide state of the art results using only a fraction of a second of CPU time for training.

2 Problem Statement: Pairwise Learning to Rank

The data sets D studied in this paper are composed of labeled examples $(\mathbf{x}, y, q)$ with each $\mathbf{x} \in \mathbb{R}^n$ showing the location of this example in n-dimensional space, each $y \in \mathbb{N}$ denoting its *rank*, and each $q \in \mathbb{Z}$ identifying the particular query/shard to which this example belongs. We define the set P of *candidate pairs* implied by D as the set of all tuple-pairs $((\mathbf{a}, y_a, q_a), (\mathbf{b}, y_b, q_b))$ with $y_a \neq y_b$ and $q_a = q_b$. When $y_a > y_b$, we say that $\mathbf{a}$ is *preferred over* $\mathbf{b}$ or *ranked better than* $\mathbf{b}$. Note that in the worse case $|P|$ grows quadratically with $|D|$.

For this paper, we restrict ourselves to solving the classic RankSVM optimization problem, first posed by Joachims [7].[1] In our notation, we seek to minimize:

$$\min \frac{\lambda}{2}||\mathbf{w}||^2 + \frac{1}{|P|}\left(\sum_{((\mathbf{a},y_a,q_a),(\mathbf{b},y_b,q_b))\in P} \text{HingeLoss}((\mathbf{a}-\mathbf{b}), \text{sign}(y_a - y_b), \mathbf{w})\right)$$

That is, we seek a weight vector $\mathbf{w} \in \mathbb{R}^n$ that gives low hinge-loss over the set candidate pairs P and also has low complexity. Here, λ is a regularization parameter and the function sign(p) returns +1 for $p > 0$, -1 for $p < 0$. Hinge loss is computed as $\max(0, 1 - \langle \mathbf{w}, \text{sign}(y_a - y_b)(\mathbf{a} - \mathbf{b})\rangle)$.

Given $\mathbf{w}$, prediction scores are made for unseen $\mathbf{x}$ by $f(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x}\rangle$, and predicted rankings are inferred by sorted score.

Observe that in the special case of binary rank values of `relevant` and `not-relevant` in a single query shard, this optimization problem is equivalent to the problem of optimizing area under the ROC curve for binary-class data.

3 A Stochastic Pairwise Strategy

Bottou *et al.* point out that the generalization ability of stochastic gradient descent relies only on the number of stochastic steps taken, not the size of the data set [1]. Thus, our main idea is to sample candidate pairs from P for stochastic steps, without constructing P explicitly. This avoids dependence on $|P|$.

Algorithm 1 gives the generic framework, reducing learning to rank to learning a binary classifier via stochastic gradient descent. This reduction preserves the convergence properties of stochastic gradient descent.

Care is needed to ensure that we are sampling from P and not some other pairwise expansion of the data. We propose two implementations of `GetRandomPair`, below, including a method that samples efficiently from P using and index of D. Methods for `StochasticGradientStep` appear in Section 3.2.

Algorithm 1 Stochastic Pairwise Descent (SPD). This generic framework reduces the pairwise learning to rank problem to the learning a binary classifier via stochastic gradient descent. The CreateIndex and GetRandomPair functions, instantiated below, enable efficient sampling from P.

1: $D_{index} \leftarrow$ CreateIndex(D)
2: $\mathbf{w_0} \leftarrow \oslash$
3: **for** $i = 1$ to t **do**
4: $\quad((\mathbf{a}, y_a, q), (\mathbf{b}, y_b, q)) \leftarrow$ GetRandomPair(D_{index})
5: $\quad\mathbf{x} \leftarrow (\mathbf{a} - \mathbf{b})$
6: $\quad y \leftarrow$ sign$(y_a - y_b)$
7: $\quad\mathbf{w}_i \leftarrow$ StochasticGradientStep($\mathbf{w}_{i-1}, \mathbf{x}, y, i$)
8: **end for**
9: **return** $\mathbf{w}_t$

3.1 Sampling Methods

Here, we instantiate the `GetRandomPair` method from the SPD algorithm, above.

Samping Without an Index It is important to be able to sample from P without explicitly indexing D when D does not fit in main memory or is unbounded, as with streaming data. This may be done by repeatedly selecting two examples $(\mathbf{a}, y_a, q_a)$ and $(\mathbf{b}, y_b, q_b)$ from the data stream until a pair is found such that $(y_a \neq y_b)$ and $(q_a = q_b)$. The expected number of rejected pairs per call is $O(\frac{|D|^2}{|P|})$.

[1] Using approximate SVM solvers such as ROMMA and the Passive-Aggressive Perceptron results in optimizing slight variants of this problem.

Indexed Sampling In our preliminary tests, we found that the rejection sampling method above was surprisingly efficient. However, we found that indexing gave faster sampling when D fit in memory. This index is constructed as follows. Let Q be the set of unique values q appearing in D, and $Y[q]$ map from values of $q \in Q$ to the set of unique y values appearing in examples $(\mathbf{x}, y, q') \in D$ with $q = q'$. Finally, let $P[q'][y']$ map from values $q \in Q$ and $y \in Y[q]$ to the set of examples $(\mathbf{x}, y', q') \in D$ for which $q = q'$ and $y = y'$. This index can be constructed in linear time using nested hash tables.

To sample from P in constant time using this index:

select q uniformly at random from Q
select y_a uniformly at random from $Y[q]$
select y_b uniformly at random from $Y[q] - y_a$.
select $(\mathbf{a}, y_a, q)$ uniformly at random from $P[q][y_a]$
select $(\mathbf{b}, y_b, q)$ uniformly at random from $P[q][y_b]$
return $((\mathbf{a}, y_a, q), (\mathbf{b}, y_b, q))$

Note that this method samples uniformly from P only when the query shards are of equal size, and the number of examples per unique y value are constant per shard. When these conditions are not met, sampling from P requires $O(\log |Q| + \log |Y[q]_{max}|)$ computation in the general case. This more general algorithm is omitted for lack of space, and because the constant time algorithm gives excellent results in practice. The quality of these results are due in part to the fact that the benchmark data sets have query shards that tend to be consistent in size. Furthermore, Cao *et al*. argue that this form of query-shard normalization is actually beneficial [4], because it ensures that very large query-shards do not dominate the optimization problem.

3.2 Stochastic Gradient Descent Methods

Because we are interested in the RankSVM optimization problem in this paper, the stochastic gradient descent methods we examine are all based on hinge-loss (also sometimes called SVM-loss) and are more properly referred to as stochastic sub-gradient descent methods. We review these variants briefly by noting how they perform updates of the weight vector $\mathbf{w}$ on each step.

SGD SVM The simple stochastic sub-gradient descent SVM solver updates only when the hinge-loss for an example $(\mathbf{x}, y)$ is non-zero [14], using the rule: $\mathbf{w} \leftarrow \mathbf{w} + \eta y \mathbf{x} - \eta \lambda \mathbf{w}$. We set the learning rate η for using the Pegasos schedule $\eta_i \leftarrow \frac{1}{\lambda i}$ on step i [13]. (We also experimented with a more traditional update rule $\eta \leftarrow \frac{c}{c+i}$ finding no improvements.)

Pegasos (SVM) Pegasos is an SVM solver that adds a hard constraint: $||\mathbf{w}|| \leq \frac{1}{\sqrt{\lambda}}$. Pegasos takes the same sub-gradient step as SGD SVM, but then projects $\mathbf{w}$ back into the L2-ball of radius $\frac{1}{\sqrt{\lambda}}$, which gives theoretical guarnatees for fast convergence [13].

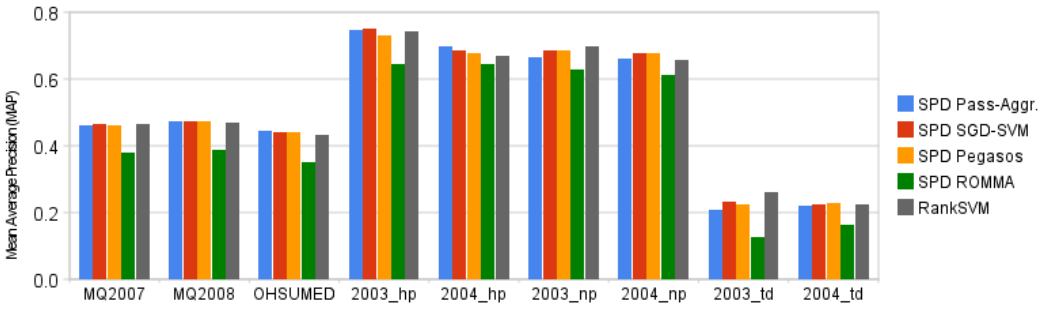
Passive-Aggressive Perceptron The PA-I algorithm [5] takes variable-size steps to minimize hinge loss and uses a regularization parameter C to control complexity, similar to SVM in several respects. The update rule is $\mathbf{w} \leftarrow \mathbf{w} + \tau y \mathbf{x}$ where $\tau \leftarrow \min(C, \frac{\text{HingeLoss}(\mathbf{x}, y, \mathbf{q})}{||\mathbf{x}||^2})$.

ROMMA The ROMMA algorithm attempts to maintain a maximum-margin hypothesis; we apply the aggressive variant that updates on any example with positive hinge loss [11]. The update rule is $\mathbf{w} \leftarrow c\mathbf{w} + d\mathbf{x}$, where $c \leftarrow \frac{||\mathbf{x}||^2||\mathbf{w}||^2 - y(\langle \mathbf{x}, \mathbf{w} \rangle)}{||\mathbf{x}||^2||\mathbf{w}||^2 - \langle \mathbf{x}, \mathbf{w} \rangle^2}$ and $d \leftarrow \frac{||\mathbf{w}||^2(y - \langle \mathbf{x}, \mathbf{w} \rangle)}{||\mathbf{x}||^2||\mathbf{w}||^2 - \langle \mathbf{x}, \mathbf{w} \rangle^2}$.

3.3 Related Methods

Joachims gave the first RankSVM optimization problem, solved with SVM-light [7], and later gave a faster solver using a plane-cutting algorithm [9] referred to here as SVM-struct-rank. Burgess *et al.* used gradient descent for ranking for non-linear neural networks and applied a probabilistic pairwise cost function [3]. Elsas *et al.* adapted voted Perceptron for ranking, but their method maintained a quadratic dependence on $|D|$ [6].

Figure 1: **Mean Average Precision (MAP) for LETOR Benchmarks.** SPD Pegasos, SPD Passive-Aggressive and SPD SGD-SVM all give results statistically equivalent to RankSVM with far less cost.



3.4 Things That Did Not Help

Neither Perceptron nor Perceptron with Margins were competitive methods. Expanding feature vectors $\mathbf{x} \in \mathbb{R}^n$ into cross-product feature-vectors $\mathbf{x}' \in \mathbb{R}^{n \times n}$ gave no improvement in preliminary trials. Our efforts to include pairs with ties (where $y_a = y_b$) yielded no additional benefit, agreeing with a similar finding in [3].

4 LETOR Experiments

Experimental Setup We first wished to compare the ranking performance of the SPD methods with RankSVM. To do so, we applied each SPD method on the LETOR 3.0 and LETOR 4.0 benchmark data sets for supervised learning to rank [12], using the standard LETOR procedures for tuning, training, and testing. We used 100,000 iterations for each SPD method; this number of iterations was picked during initial tests and held constant for all methods. The RankSVM comparison results are previously published benchmark results on these tasks [12].

Ranking Performance The Mean Average Precision (MAP) results for each LETOR task are given in Figure 1. To our surprise, we found not only that SPD Pegasos was statistically indistinguishable from RankSVM, but also that SPD SGD-SVM and SPD Passive-Aggressive were equivalent to RankSVM as well. These results are extremely encouraging; a range of simple SPD methods perform as well as RankSVM on these standard benchmarks. SPD ROMMA was not competitive.

Training Speed All experiments were run on a dual-core 2.8GHz laptop with 2G RAM with negligible additional load. The SPD methods each averaged between 0.2 and 0.3 CPU seconds per task for training time, which includes the construction of the sampling index and 100,000 stochastic steps. This was up to 100 times faster than the SVM-light implementation [7] of RankSVM, and up to 5 times faster than the SVM-struct-rank implementation [9]. However, this small scale setting of at most 70,000 examples was not large enough to fully highlight the scalability of the SPD methods.

Table 1: **Large-Scale Experimental Results**. Results for ROC area and CPUs for training time are given for the E311 task in the RCV1 data set.

LEARNER	ROC AREA	CPUS TRAINING
SVM-struct-rank	0.9992	30,716.33s
SVM-perf-roc	0.9992	31.92s
SPD Pass-Aggr	0.9990	.22s
SPD SGD-SVM	0.9992	.20s
SPD Pegasos	0.9992	.20s

5 Large Scale Experiments

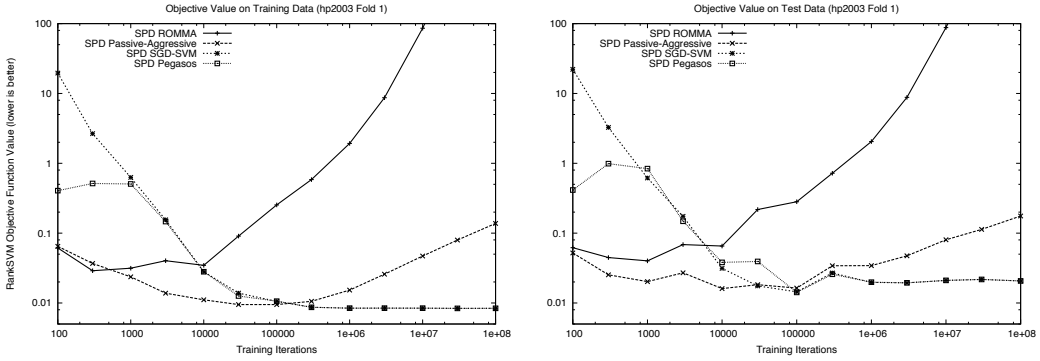
Experimental Setup To assess the ability of the SPD to perform at scale, we needed a larger ranking task than the LETOR benchmark data sets provided. Thus, we ran experiments on the much larger RCV1 data set for text categorization [10], treating the goal of optimizing ROC area (ROCA) as a large-scale ranking problem. (Recall that optimizing ROCA is equivalent to solving a ranking problem with binary ranks and a single query shard.)

The training set we used had 781,265 examples, with 23,149 examples in the test set. For space, we report results only the E311 topic, a minority class topic with a 0.2% positive class distribution. (Results on 20 other RCV1 minority-class topics from this data set are similar; omitted for space.) Parameters were tuned by cross-validation on the training set alone.

Scalability Results The results in Table 1 show that SPD methods were not affected by this increase in scale, using no more CPU time on this task than on the smaller LETOR tasks. SPD methods were more than 150 times faster than the SVM-perf-roc implementation of RankSVM tuned to ROCA optimization [8], and were many orders of magnitude faster than SVM-struct-rank (which not is optimized for large query shards). The ROCA performance of all ranking methods exceeded that of a binary-class SVM with tuned class-specific regularization parameters.

These results highlight the ability of simple SPD methods to give good ranking performance on large scale data sets with low computation cost for learning.

Figure 2: **Convergence Results.** These graphs show the value of the RankSVM objective function for each SPD method on the training data (left) and on the test data (right) for the `hp_2003` task from LETOR.



6 Convergence Trials

After these results were digested, we went back to examine how quickly each SPD learner converged to a stable value of the RankSVM objective function. Because each of the SPD methods was intended to solve or approximate the RankSVM objective function given in Section 2, we used the value of this objective function as our test for convergence for all learners. (However, we keep in mind that SPD ROMMA and SPD Passive-Aggressive are not intended to solve this exact RankSVM problem.) Objective function values were observed at increasing intervals from 10^2 iterations (a small fraction of a second in CPUs for training) to 10^8 iterations (up to 500 CPUs for training). We performed these tests on the `hp_2003` task of the LETOR data set, using the train/test split given in Fold 1 of the cross-validation folds, using the value for λ in the objective function determined as optimal for RankSVM using the validation set for this fold.

The results, given in Figure 2, show several interesting trends. Both SPD SGD-SVM and SPD Pegasos converge to stable, apparently optimal values on training data within 10^6 iterations. Continued training does not materially degrade performance on the test objective values. In contrast, both SPD Passive-Aggressive and SPD ROMMA give much better objective values than the other methods at small numbers of iterations (less than 10^5). However, their performance does degrade with additional training, with SPD ROMMA degrading especially rapidly. We attribute this to the fact that

unlike SPD Pegasos and SPD SGD-SVM these methods do not optimize the RankSVM objective function directly, and to the fact that these methods can take large update steps on noisy examples regardless of the number of past iterations.

These results suggest that it may be fruitful to combine the best elements of both approaches, perhaps by using SPD Passive-Aggressive in early iterations and SPD Pegasos or SPD SGD-SVM in later iterations. We leave this investigation to future work.

References

[1] L. Bottou and O. Bousquet. The tradeoffs of large scale learning. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems*, volume 20, pages 161–168. NIPS Foundation (http://books.nips.cc), 2008.

[2] L. Bottou and Y. LeCun. On-line learning for very large datasets. *Applied Stochastic Models in Business and Industry*, 21(2):137–151, 2005.

[3] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*, 2005.

[4] Y. Cao, J. Xu, T.-Y. Liu, H. Li, Y. Huang, and H.-W. Hon. Adapting ranking SVM to document retrieval. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, 2006.

[5] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer. Online passive-aggressive algorithms. *J. Mach. Learn. Res.*, 7, 2006.

[6] J. L. Elsas, V. R. Carvalho, and J. G. Carbonell. Fast learning of document ranking functions with the committee perceptron. In *WSDM '08: Proceedings of the international conference on Web search and web data mining*, 2008.

[7] T. Joachims. Optimizing search engines using clickthrough data. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2002.

[8] T. Joachims. A support vector method for multivariate performance measures. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*, 2005.

[9] T. Joachims. Training linear svms in linear time. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2006.

[10] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. Rcv1: A new benchmark collection for text categorization research. *J. Mach. Learn. Res.*, 5:361–397, 2004.

[11] Y. Li and P. M. Long. The relaxed online maximum margin algorithm. *Mach. Learn.*, 46(1-3), 2002.

[12] T.-Y. Liu, T. Qin, J. Xu, W. Xiong, and H. Li. LETOR: Benchmark dataset for research on learning to rank for information retrieval. In *LR4IR 2007: Workshop on Learning to Rank for Information Retrieval, in conjunction with SIGIR 2007*, 2007.

[13] S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: Primal estimated sub-gradient solver for svm. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, 2007.

[14] T. Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *ICML '04: Proceedings of the twenty-first international conference on Machine learning*, 2004.