Indexed Learning for Large-Scale Linear Classification

Ian En-Hsu Yen  
Department of Computer Science  
National Taiwan University  
Taipei 106, Taiwan  
r00922017@csie.ntu.edu.tw

Abstract

Linear classification has achieved complexity linear to the data size. However, in many applications, large-scale data contains only a few samples that can improve the target objective. In this paper, we propose a sublinear-time algorithm that uses Nearest-Neighbor-based Coordinate Descent method to solve Linear SVM with truncated loss. In particular, we propose a sequential relaxation that solves problem with general truncated-loss by a series of sparse convex programs. Then we solve each program using indexed dual coordinate descent, which avoids I/O of unnecessary data by searching coordinates of large gradient. We show the proposed algorithm has linear convergence rate, and has sublinear complexity for each iteration. We also demonstrates its empirical success in sufficient and limited memory conditions. In both cases, the indexed learning algorithm learns times faster than state-of-the-art linear hinge-loss and ramp-loss solvers without sacrificing accuracy.

1 Introduction

Recent advance in linear classification has achieved complexity linear to data size. However, in practice, data are usually collected from unstructured and noisy environment, where could be many redundant or, even worse, noisy samples that significantly increase the data size, I/O time, and memory space. When memory is insufficient, solutions such as online or block-wise optimization can be exploited, but they often need multiple rounds of I/O to get convergence [1][2]. Random sampling, or single-pass online learning, might be an alternative. Practically, they often find rough solution in reasonable time. And theoretically, as [3] shows, online learner can converge to ground truth asymptotically at the same rate as optimal batch learner. This seems to indicate that, for large-scale learning, there is no need to perform optimization accurately. However, we found, in many applications, people did optimize as accurate as possible, since they care more about improvement on a small part of samples that are of special interest. One example is when people want to improve their model from 99% to 99.9%, where they focus on how to predict right on those difficult samples $D_{mis}(w)$ misclassified by current model $w$. Another example comes from applications who use Precision $P(D_k(w))$ and Recall $R(D_k(w))$ as their evaluation measures. The two measures depend only on the top-k samples ranked by $w$. In those cases, only a small subset of data $D_k(w)$ can effectively improve model on the defined measures. As online method draw samples from the whole data set, the effective sample size from $D_k(w)$ may grow too slowly to obtain the asymptotic result.

This motivates us to think if there is a way to directly sample from $D_k(w)$ that can effectively improve our model? This seems to be a very challenging problem since $D_k(w)$ is a function of $w$, which keeps changing during training process. However, we have realized that it is achievable, if
one can organize data following certain criteria before learning. In practice, this is beneficial since people often learn a lot of models from the same data. Scenarios such as parameter tuning, model selection, cross-validation, multi-class classification, feature selection, data incremental learning etc. can all benefit from a pre-built index which serves as a search engine for samples from \( D_e(w) \).

The goal of this paper is to design an indexed learning algorithm that learns from informative samples without browsing through the data. However, there are at least three challenges to achieve such goal: (i) the optimization algorithm should not require to use the whole data, (ii) the optimal solutions without browsing through the data is more efficient due to the save of search cost. In this paper, we demonstrate that even if the single variable subproblem can be solved very cheaply in Linear SVM, NNGCD is still very attractive considering the cost of I/O and sparsity of model.

The sparsity of underlying problem is critical to the efficiency of NNGCD approach. Though SVM is known to be dual sparse, the number of support vectors under standard L1/L2-loss is actually linear to the size of data [5]. In [6], Collobert et al. proposed using ramp-loss to obtain a sparser SVM, which is also known to be more robust than L1/L2-loss in noisy setting [7][8]. Although learning ramp-loss SVM is a non-convex problem, [6] points out the computational advantage of trading convexity for sparsity. They decompose the non-convex problem into a series of convex ones using Concave-Convex Procedure (CCCP) proposed by Yuille et al. [9], where one can learn a non-linear SVM much more efficiently due to significant decrease in number of support vectors.

While CCCP can also solve linear ramp-loss SVM by integrating with linear solvers such as LIBLINEAR [10], Pegasos [11] or SVMPerf [12], since their efficiency does not depend much on the number of support vectors, the ramp-loss solver can be even slower than the hinge-loss one. In this paper, we propose a indexed dual coordinate descent algorithm that applies NNGCD on linear ramp-loss SVM problem, which, once again, takes advantage of the sparsity, by selecting coordinates of large gradient in sublinear time.

The paper is organized as follows. In section 2, we propose a new relaxation that solves problem of truncated loss by a series of convex programs with superior sparsity. In section 3, we introduce indexed dual coordinate descent to solve each subproblem without browsing through the data. Section 4 gives implementation details and indexing method we use in the experiment. Section 5 conduct experiments on three large-scale datasets under sufficient and limited memory conditions. In both conditions, our algorithm learns times faster than state-of-the-art linear hinge-loss and ramp-loss solvers without sacrificing accuracy.

## 2 Relaxation for Truncated Loss

We start from introducing the ramp-loss (or truncated loss) as:

\[
R_s(z) = \min(L(z), 1 + s)
\]

where \( L(z) \) can be \( L_1 \) loss \( \max(1 - z, 0) \), \( L_2 \) loss \( \max(1 - z, 0)^2 \), or other convex loss function. Under (1), a sample with error more than threshold \( s \) is taken as outlier, and has loss at most \( 1 + s \).

This truncated loss function is more like 0/1-loss, and thus more robust in noisy setting. More importantly, all outliers will not become support vectors under (1). Thus the solution can be sublinearly sparse. However, the ramp-loss learning problem

\[
\min_w \frac{1}{2} \|w\|^2 + C \sum_{i \in D} R_s(y_i w^T x_i)
\]

is non-convex. [6] divides (2) into a convex part \( J_{\text{vex}}(w) \) and a concave part \( J_{\text{cav}}(w) \):

\[
J(w) = J_{\text{vex}}(w) + J_{\text{cav}}(w)
\]

\[
= \left\{ \frac{1}{2} \|w\|^2 + C \sum_{i \in D} \max(1 - y_i w^T x_i, 0) \right\} + \left\{ -C \sum_{i \in D} \max(-s - y_i w^T x_i, 0) \right\}
\]

2
The sequence (5) is a series of convex programs. [9] shows that this algorithm strictly decrease the original objective $J(w)$, and more recently, [13] shows this algorithm globally converges to a stationary point of $J(w)$. However, they also point out the convergence rate of (4) is still an open problem for non-differentiable problem like (2).

Another concern for CCCP is, the formulation (3) is only for hinge-loss. For other loss functions, it is not obvious how to derive the convex part and concave part. And since (4) is different from the original problem for $L(z)$, solver of the original problem cannot be utilized. Here we propose a new convex relaxation for general truncated-loss (1). Firstly, define set of outliers and inliers as $\text{OUT}(w) = \{y_l w^T x_l < -s\}$ and $\text{IN}(w) = \{y_l w^T x_l \geq -s\}$. In each iteration, the new relaxation solves

$$w^{t+1} = \arg\min_w \frac{1}{2} \|w\|^2 + C \sum_{l \in \text{IN}(w^t)} L(y_l w^T x_l) + C \sum_{\text{OUT}(w^t)} 1 + s$$

which is very intuitive: solve convex problem of the original loss $L(z)$ for only inliers, while fix loss of outliers at $1+s$. This formulation is motivated by the observation that truncated loss $R_s(z)$ is upper-bounded both by $L(z)$ and $1+s$, so

$$\frac{1}{2} \|w^t\|^2 + C \sum_{l \in \mathcal{D}} R(y_l w^T x_l)$$

$$= \frac{1}{2} \|w^t\|^2 + C \sum_{\text{IN}(w^t)} L(y_l w^T x_l) + C \sum_{\text{OUT}(w^t)} 1 + s$$

$$\geq \frac{1}{2} \|w^{t+1}\|^2 + C \sum_{\text{IN}(w^t)} L(y_l w^{t+1}T x_l) + C \sum_{\text{OUT}(w^t)} 1 + s$$

$$\geq \frac{1}{2} \|w^{t+1}\|^2 + C \sum_{l \in \mathcal{D}} R(y_l w^{t+1}T x_l)$$

In other words, (5) minimizes an upper bound of (2), which strictly decrease (2) when $\text{IN}(w^{t+1}) = \text{IN}(w^t)$ (or, equivalently, $\text{OUT}(w^{t+1}) = \text{OUT}(w^t)$). When $\text{IN}(w^{t+1}) = \text{IN}(w^t)$, (2) equals to (5), and thus we get a minimum of (2).

The advantage of (5) is obvious. Since it just ignores outliers (they are not related to $w$) and solves the original problem of $L(z)$, any solvers for the original loss can be utilized to solve (5). Furthermore, the relaxation has no assumption on $L(z)$. Thus it is a general approach for solving any truncated version of loss function. Even in other problems like regression or clustering, one may take this approach to obtain a much sparser, outlier-free model for indexed learning.

We further show that the sequence $\{w^t\}_{t=0}^\infty$ produced by (5) has linear convergence rate by following theorem.

**Theorem 2.1.** The sequence $\{w^t\}_{t=0}^\infty$ produced by (5) converges to a stationary point of (2) with at least linear convergence rate.

**Proof Sketch.** Since the objective in (5) is an upper bound for (2), a sample changing from $\text{IN}(w^t)$ to $\text{OUT}(w^{t+1})$ or $\text{OUT}(w^t)$ to $\text{IN}(w^{t+1})$ can be seen as a process of descent. Therefore, we can interpret (5) as an alternating minimization (block coordinate descent) between $d \in \mathbb{R}^m$ and $(w, \xi) \in (\mathbb{R}^n, \mathbb{R}^m)$ on the problem

$$\min_{w, \xi, d} \frac{1}{2} \|w\|^2 + C \sum_{l \in \mathcal{D}} d_l \xi_l + (1-d_l)(1+s)$$

$$s.t. \quad y_l w^T x_l \leq 1 - \xi_l$$

$$\xi_l \geq 0$$

$$0 \leq d_l \leq 1, \quad l = 1..m$$

The decrease is strict since $R(y_l w^T x_l) < 1+s$ for $l \notin \text{OUT}(w)$, and $R(y_l w^T x_l) < L(y_l w^T x_l)$ for $l \notin \text{IN}(w)$. 

```
which satisfies the form of nonsmooth, separable problem studied in [14]. Thus we can find an equivalent Coordinate Gradient Descent procedure introduced in their paper, which produces the same sequence as that produced by block coordinate descent on (6). By theorem 1, 2 and 4 of [14], the sequence converges to a stationary point of (6) with at least linear convergence rate. A detailed version of this proof is in the appendix.

Since problem (2) is nonconvex, the initialization \( w^0 \) of (5) may significantly affect result. A safe choice is solving the original hinge-loss problem to get \( w^H \) and set \( w^0 = w^H \). Then since (5) guarantees to decrease (2), we are likely to get a solution \( w^* \) with higher, at least not lower, accuracy. However, the initialization cannot gain any efficiency since it needs to solve the original problem. To gain efficiency, a practical choice is solving hinge-loss problem on a subset of random samples to give initial \( w^0 \), which yields great performance in our experiment.

3 Indexed Dual Coordinate Descent

Without outliers, each subproblem in (5) has significantly less support vectors and can be solved sparsely, which becomes the foundation of this section. In this section, we introduce an indexed optimization algorithm that solves (5) using only informative samples and avoids I/O of unnecessary data.

3.1 Informative Sample as Nearest-Neighbor

First, we consider how informative samples can be obtained via ANN search. In case of ramp-loss SVM, the informative samples are inliers with confidence less than the margin: \( \{ l | s < y_l w^T x_l < 1 \} \). When solving (5) in dual form, those samples correspond to dual variables with large projected gradient. In [15], a Locality Sensitive Hashing (LSH) method EH-Hash was proposed to handle nearest-to-hyperplane query occurs in Active Learning\(^2\). Here, we apply a similar technique that can be used by indexing search methods other than LSH, which however, costs cheaper when combined with Tree-based index, as described in section 4.1.

Given a query vector \( q \), the nearest neighbor is defined as the vector \( x_l \) most similar to \( q \), that is,

\[
\arg \max_i q^T x_l
\]

However, sample closest to a hyperplane of normal vector \( w \) is defined as

\[
\arg \min_i |w^T x_l|
\]

We connect these two problems using kernel trick for degree-2 polynomial feature expansion \( V(x) = [x_1^2, \sqrt{2}x_1x_2, ..., \sqrt{2}x_1x_d, x_2^2, \sqrt{2}x_2x_3, ..., x_d^2] \) such that the nearest-to-hyperplane problem in original space can be reduced to ANN problem in embedded space. To see this:

\[
\arg \min_i |w^T x_l| = \arg \min_i (w^T x_l)^2 = \arg \min_i V(w)^T V(x_l) = \arg \max_i (-V(w))^T V(x_l)
\]

Where \(-V(w)\) is the transformed nearest-neighbor query, and \((-V(w))^T V(x_l)\) can be computed efficiently using \((-w^T x_l)^2\), which is not like the case in [15], where quadratic cost was required to generate Gaussian-distributed hashing function in the embedded space. The above transformation guarantees \((-w^T x_l)^2\) to be a valid inner product in some Hilbert space, and thus, can use indexing structure designed for nearest-neighbor problem.

\(^2\)In [15], they focus on minimizing search time required to obtain a new informative sample in active learning, which differs from our goal: minimizing total learning time required by solving corresponding optimization problem.

\(^3\)Here \( q, x_l \) are assumed to be unit vectors for simplicity.
Algorithm 1 Indexed Dual Coordinate Descent

Input: \( w^{(t,0)} = w^t, S^{(t,0)} = S^t \setminus \text{OUT}(w^t) \)
Output: \( w^{t+1} = w^{(t,k)}, S^{t+1} = S^{(t,k)} \)
repeat
\[
(N, n_e) \leftarrow \text{queryIndex}(w^{(t,k)}; w^t, s, n)
\]
\[
B \leftarrow [N, S^{(t,k)}[r : r + n_e - |N|]]
\]
Solve block minimization problem (10) over \( B \).
\[
k \leftarrow k + 1; r \leftarrow (r + n_e + 1) \mod |S^{(t,k)}| \]
until max projected gradient in \( N \cup S < \epsilon_D \)

3.2 Indexed Dual Coordinate Descent

The indexed algorithm 1 solves dual form of (5)

\[
\min_{\alpha \in \mathbb{R}^N} f(\alpha) = \frac{1}{2} \alpha^T Q \alpha - e^T \alpha
\text{ s.t. } 0 \leq \alpha_l \leq C, \quad l \in I(N(w^t))
\alpha_l = 0, \quad l \in \text{OUT}(w^t)
\]

where \( Q \) is a \( N \times N \) matrix with \( Q_{ij} = y_i y_j x_i^T x_j \), and \( e \) is \( N \times 1 \) vector \([1, 1, ..., 1]^T\).

Using Dual Coordinate Descent method proposed by [16], we minimize (7) over single variable \( \alpha_l \) while fixing all the others. It can be solved in closed form

\[
\alpha_l^* = \min \left( \max \left( \alpha_l - \frac{\nabla_l f(\alpha)}{Q_{ll}}, 0 \right), C \right)
\]

Computing \( \nabla_l f(\alpha) \) can be very expensive in nonlinear case. However, in linear case, if we maintain a weight vector

\[
w = \sum_{l=1}^{n} y_l \alpha_l x_l
\]

we can compute gradient cheaply for every single \( \alpha_l \) as

\[
\nabla_l f(\alpha) = (Q\alpha)_l - 1 = y_l w^T x_l - 1
\]

and maintain relation (8) by

\[
w \leftarrow w + (\alpha_l^* - \alpha_l)y_l x_l
\]

In Algorithm 1, instead of loading all samples from \( I(N(w^t)) \), we request potential support vectors from \( \text{queryIndex}(w^{(t,k)}; w^t, s, n) \) with query defined in 3.1, which searches for \( n \) samples \( \notin S^{(t,k)} \) satisfying

\[
y_l w^T x_l \geq -s \quad \text{AND} \quad y_l w^{(t,k)} x_l < 1 \quad (9)
\]

where \( S^{(t,k)} \) is a set containing samples returned by \( \text{queryIndex}(\cdot) \) of all iterations. (9) guarantees samples to be inliers with non-zero projected gradient (PG)

\[
\nabla_l^P f(\alpha) = \begin{cases} 
\nabla_l f(\alpha) & \text{if } 0 < \alpha_l < C \\
\min(0, \nabla_l f(\alpha)) & \text{if } \alpha_l = 0 \\
\max(0, \nabla_l f(\alpha)) & \text{if } \alpha_l = C 
\end{cases}
\]

When sample \( l \) satisfies (9), it has non-zero PGs since \( \alpha_l = 0 \) (they are never updated) and \( \nabla_l f(\alpha) = y_l w^T x_l - 1 < 0 \). Furthermore, \( \text{queryIndex}(\cdot) \) performs ANN search for sample close to boundary, that is, \( w^T x_l \) close 0, so the query result tends to have large gradients no matter label \( y_l \) is 1 or -1.

In Algorithm 1, we denote fresh samples returned by \( \text{queryIndex}(\cdot) \) as \( N \), where \( n_e \) is the number of traversed instances for finding \( |N| = n \) samples with non-zero PG. Though samples from \( N \) have larger PGs, they come from cost of search. To balance cost between search and optimize, we maintain a set \( S^{(t,k)} \subseteq IN(w^t) \) that contains samples from \( N \) of previous iterations to speed up
convergence. Each iteration, we do block minimization on block $B$ composed of samples from both $N$ and $S^{(t,k)}$

$$\begin{align*}
\min_{d} & \quad f(\alpha + d) \\
\text{s.t.} & \quad 0 \leq \alpha_l + d_l \leq C, \quad l \in B \\
& \quad d_l = 0, \quad l \notin B
\end{align*}$$

where we design our block size $|B| = n_e$ with $n_e - |N|$ elements from cyclically traversing $S^{(t,k)}$. Then both search and block minimization will not become the bottleneck since both of their cost are proportional to $n_e$. In this paper, we use Coordinate Descent to solve (10).

The following theorem shows the complexity of Algorithm (1) depends on the efficiency of indexed ANN search, which is sublinear to data size $N$.

**Theorem 3.1.** The sequence $\{\alpha^t\}_{t=1}^{\infty}$ produced by Algorithm 1 globally converges to an optimal solution $\alpha^*$ of (7) with at least linear rate. Thus the time complexity for obtaining an $\epsilon_D$ solution is

$$O\left( \frac{nd}{\text{prec}[n]} \left( 1 + \log(1/\epsilon_B) \right) \log(1/\epsilon_D) \right)$$

where $d$ is number of non-zero features per instance, $\text{prec}[n]$ is the lowest Precision at Recall = $n$ of indexed ANN search over all iterations, and $\epsilon_B$ is the tolerance for block minimization (10).

**Proof Sketch.** The algorithm can be seen as Block Coordinate Descent (BCD) on the dual problem (7), which satisfies the form of convex, smooth problem studied in [17], as shown in [16]. Since in every iteration, block $B$ contains coordinates with non-zero projected gradient from $N$, the algorithm satisfies Gauss-Southwell update rule for BCD, thus with linear convergence rate by the extended version of theorem 2.1 in [17]. It means the BCD needs $O(\log(1/\epsilon_D))$ iterations to obtain an $\epsilon_D$ solution. In each iteration, the cost for indexed search is $O(dn_e)$, and cost for block minimization is $O(dn_e \log(1/\epsilon_B))$, where $n_e = n/\text{prec}[n]$. Therefore, the total cost of algorithm (1) is $O\left( \frac{nd}{\text{prec}[n]} \left( 1 + \log(1/\epsilon_B) \right) \log(1/\epsilon_D) \right)$. 

Since the sequential relaxation for truncated-loss also converges with linear rate, we can express the complexity of whole algorithm as

$$O\left( \frac{nd}{\text{prec}[n]} \left( 1 + \log(1/\epsilon_B) \right) \log(1/\epsilon_D) \log(1/\epsilon_R) \right)$$

where $\epsilon_R$ is the tolerance for ramp-loss problem (2). There is a trade-off between the efficiency of outer and inner loops. When inner loop solves with higher precision, we can get faster convergence at outer, however, with high cost at inner. Practically, we can set a loose inner precision, and increase it with outer iterations.

### 4 Implementation Issues

In this section, we address some details of implementing an indexed optimization algorithm. First, we discuss challenges of building index for learning problem, as opposed to other problem like retrieval. Then we discuss the Caching, Shrinking strategy for learning with less memory and time.

#### 4.1 Indexing for Learning

There are two practical challenges of building index for selecting informative training samples.

**Indexing Bias** Index for ANN search is approximated and biased towards the reference points (or hash functions) selected. While such bias is acceptable for applications like multimedia retrieval, training on biased samples will seriously degrade the quality of solution.
Index on Disk The index should be built on disk so it can be reused for different models. However, not all indexing methods are cost-effective to be built on disk. For example, Locality-sensitive hashing (LSH), one of the most popular methods recent years, requires a sample to be stored in different hash tables, which can significantly increase the storage cost [18].

Here we propose Metric Forest that addresses these issues. Unlike LSH, a tree-structured index has size comparable to the original data. In [19], a tree-based indexing, called Metric Tree, is modified to solve ANN problem, and is shown to be more efficient than LSH. A Metric Tree organizes points as a binary tree, in which each subtree $N(v)$ rooted on a node $v$ partitioned into points closer to $v.lpv$ and those closer to $v.rpv$, where $v.lpv$ and $v.rpv$ are pivot points selected from data, and the metric is defined by angle distance in the embedded space: $d(x_1, x_2) = \cos^{-1}(V(x_1)^T V(x_2)) = \cos^{-1}((x_1^T x_2)^2)$. Here we propose Metric Forest, which shares the basic idea of Metric tree, but is different in three ways:

Bias Reduction Since reference points closer to the root of a Metric Tree are used more frequently, query results are often biased toward those points. [19] propose a variant called Spill-Tree to allow overlap between different partitions, thus alleviate the bias problem. However, the overlap between partitions significantly increases storage, and under limited memory, even building a tree is time-consuming. In our design, data are split into $R$ random blocks which can be put into memory, and a Metric Tree is built for each block. When performing search, all trees are traversed in an order determined randomly. Such design warrants the query results not biased to a small number of reference points.

Partitioning While binary tree works well in memory, it is inefficient on disk because the seeking time on disk is more expensive. Here we use, instead, K-way partitioning such that the tree depth is smaller and each tree node contains more points. For each subtree $N(v)$ rooted on node $v$, points are partitioned into $N(v.pv_1)...N(v.pv_K)$ s.t. $v.pv_k$ is nearest to the points in $N(v.pv_k)$ among $v.pv_1...v.pv_K$.

Searching In [19], Defeatist Search is used to obtain efficient approximate search, in which only the nearest partition $N(v.pv_k)$ to a query point is visited. However, in our algorithm, we need to increase, or decrease, search range according to the current margin $1/\|w\|$. Therefore, we exploit Best-Bin-First search strategy, as proposed in [20], in which each partition $N(v.pv_k)$ is put in a priority queue, and ranked by their distance to the query, where the distance is measured by their nearest boundary to the query point, as shown in Figure 1. Let $v.pv_q$ be the closest pivot point to the query, the nearest boundary of partition $N(v.pv_k)$ is defined by the hyperplane passing $(v.pv_k + v.pv_q)/2$ with normal vector $v.pv_k - v.pv_q$.

4.2 Caching Coordinates

Indexed optimization shows a new way to handle large-scale learning under limited memory. In our implementation, we use Least-Recently-Used (LRU) cache for frequently-used coordinates to minimize I/O between memory and disk. This is effective since, in queryIndex(.), we only traverse branches of Metric Tree near SVM decision boundary. As model changes over time, samples away from the boundary will not be traversed again, and thus, are removed from cache when memory is full. Since we keep samples with larger projected gradient in memory, I/O of unnecessary data is minimized. In contrast, other approaches like online or block minimization, as in [1] and [2], need to traverse data repeatedly to update all samples, even if they are without projected gradients.
4.3 Shrinking on Index

In search of queryIndex(.,), samples not satisfying (9) for \( w^{(t,k)} \) are likely not satisfying (9) for \( w^{(t,k+1)}, w^{(t,k+2)}, \ldots \) either. This is like the case in most SVM solver, where many bounded variables, that is, \( \alpha_l = 0 \) or \( \alpha_l = C \), tends to be unchanged if their gradients are large to the bounded direction. Here we apply a similar technique to shrink variables that are not likely to satisfy (9) in later iterations, which includes variables with (i) \( y_l w^{(t,k)} x_l > 1 + T \), (ii) \( y_l w^{(t)} x_l < -s - T \), (iii) \( l \in S^{(t,k)} \), where \( T \) is a threshold. We determine the threshold \( T \) using the maximum magnitude of projected gradient in recently used variables. The shrunk variables would not be traversed in a Metric Tree, and a tree node is not traversed if all of its descendants are shrunk. When the shrunk problem converges, we recover all of shrunk variables and solve the non-shrunk problem to check convergence. This is repeated until the original non-shrunk problem is solved.

5 Experiment

In this section, we compare our algorithm with state-of-the-art linear SVM solvers\(^4\) LIBLINEAR, Pegasos, and ramp-loss solver that uses Dual Coordinate Descent (as implemented in LIBLINEAR) as inner procedure for CCCP. We compare batch version of above solvers in sufficient memory condition (28G under Linux Ubuntu 11.04). Under limited memory condition (2G under Linux Ubuntu 12.04), we compare with online version of Pegasos and LIBLINEAR-CDBLOCK, a limited-memory version of LIBLINEAR proposed in [1], refined in [2]. The initialization for both CCCP and our algorithm uses 10,000 random samples solved by LIBLINEAR. Initialization and I/O are all included into training time.

Table 1: Statistics of Data.

<table>
<thead>
<tr>
<th>DATASET</th>
<th>#SAMPLES</th>
<th>#FEATURES</th>
<th>STORAGE (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>COVTYPE</td>
<td>581,012</td>
<td>54</td>
<td>69,516</td>
</tr>
<tr>
<td>KDDCUP1999</td>
<td>4,898,431</td>
<td>126</td>
<td>725,180</td>
</tr>
<tr>
<td>MNIST8M</td>
<td>8,100,000</td>
<td>784</td>
<td>19,042,640</td>
</tr>
</tbody>
</table>

Table 2: Statistics of Index.

<table>
<thead>
<tr>
<th>DATASET</th>
<th>STORAGE (KB)</th>
<th>TREE SIZE</th>
<th>TREE WIDTH</th>
<th>BUILD TIME (S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>COVTYPE</td>
<td>446,444</td>
<td>2,000</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>KDDCUP1999</td>
<td>1,476,580</td>
<td>100,000</td>
<td>100</td>
<td>163</td>
</tr>
<tr>
<td>MNIST8M</td>
<td>20,704,784</td>
<td>2,000</td>
<td>10</td>
<td>1,539</td>
</tr>
</tbody>
</table>

Our experiments conducted on 3 large-scale public datasets of increasing size: Covtype, Kddcup1999, and Mnist8m. Their statistics are summarized in Table 1.

Table 2 shows the statistics of building index. The construction time and storage size are linear to the data size. In our experiment, we found that the parameter tree size (e.g. #samples / tree) and tree width (e.g. #pivot-points) do not affect result significantly. However, tree size should be large enough s.t. each tree contains useful samples, and small enough s.t. each tree can be built in memory. Tree width should be large enough s.t. the average depth=log_width(Size) is reasonably small (3 to 5 in our experiment). Since KDDCUP1999 has many irrelevant samples, its tree size and tree width are set to be larger than that of the others.

We scale all features to between 0 and 1. For Covtype and Kddcup1999, we randomly selected 1/3 of samples for testing and the remaining 2/3 samples for training. For Mnist8m, to test on limited memory, we only used 1/10 for testing and left 9/10 for training. All of these data have multi-classes, and thus the index built can be reused for different models. Due to space limitation, our figures come from one of those 1-against-all models for each dataset. For Covtype, the given result is on the model

\(^4\)There are still other state-of-the-art solvers. However, as our contribution is a indexed learning framework, a comparison between methods with/without our technique is more essential than that between different solvers.
Figure 2: Testing error of indexed optimization (blue), LIBLINER (pink), Pegasos (black), and CCCP (red), where the lower 2 plots use log-scale in time, and the 4th plot runs in limited memory (2G) condition, where we compare with Online Pegasos, and CDBLOCK-LIBLINEAR using 20 blocks and 1G cache of informative samples.

Figure 3: Testing error vs. #SV for a spectrum of s.

of class 2 against the other 6 classes, following the choice of [21]. For KDDCUP1999, we show result of the class normal against all the other 22 abnormal classes. And for mnist8m, we show result of class digit-1 against all the other 9 digits. We set parameter s=1 for both CCCP and our method, and set c=1 for all SVM models, if not mentioned otherwise.

The first three plots in Figure 2 show the testing error under sufficient memory, where our algorithm obtains reasonable accuracy even before other algorithms finish data loading, and converge times faster to the same accuracy of CCCP, which is more accurate than LIBLINEAR and Pegasos. In the forth plot, we compare indexed learning algorithm with Online Pegasos and LIBLINEAR-CDBLOCK, where data is 10 times larger than memory space, under which the batch version of LIBLINEAR, CCCP suffer from severe swaps and can hardly progress. In the plot, all algorithms slow down under limited memory due to repeated I/O, where Online Pegasos converges slowly since it cannot reuse samples already in memory. LIBLINEAR-CDBLOCK can hardly obtain the same accuracy as in the case of sufficient-memory, since it tends to optimize towards each noisy block iteratively. The indexed learning algorithm, by keeping only informative samples in memory, avoids
lots of I/O and converge more smoothly.\footnote{Note our algorithm converges more than once since it solves many convex sub-problems.} Note that although building index takes time linear to the data size, they can be reused for models trained on the same data, as mentioned in the introduction. Note in the case of limited-memory, our approach is faster even including the time for building index, since building index only costs one-pass of I/O, while online, or block minimization costs several passes. In Figure 5, we also test our algorithm for a spectrum of parameter $c$.

In Figure 3, we show the number of support vectors, and testing error of our algorithm for a spectrum of $s$, which determines how much error it takes to become an outlier. When $s = 1$, it means outlier should have error more than current margin. This choice yields great performance for most data. However, in mnist8m dataset, $s = 2$ is a better choice.

Figure 4 shows how the number of support vectors and training time increase with data size. The number of support vectors under ramp-loss is clear to be sublinear, while the training time depends on the efficiency of indexed ANN search, which become advantageous when data increase to a larger scale.

6 Conclusion

In many applications, only a small subset of samples $D_e(w)$ in large-scale data can effectively improve models. While random sampling, or single-pass online learning can overlook those rare
but crucial samples, batch learners will cost too much memory and I/O time. In this paper, we propose indexed optimization algorithm that tries to directly sample from $D_n(w)$, and solves sparse truncated-loss problem in sublinear time. Though building index takes time linear to the data size, in practice, this is beneficial since in general it requires multiple times of training to obtain a good model, and one often learn several models from the same data. Scenarios such as parameter tuning, model selection, cross-validation, multi-class classification, feature selection, and data incremental learning all require multiple passes of training. In the case of limited memory, our approach can be more efficient even considering the costs of building index, due to the save of repeated I/O time. This approach can potentially apply to a general class of large-scale learning problem, through defining new truncated-loss function for convex-loss problems in classification, regression or clustering.

References

## 1 Proof of Theorem 2.1

**Lemma 1.1.** Consider the problem
\[
\min_{x, y} \quad F(x, y) = f(x, y) + cP(x, y) \quad (1)
\]
where \(f(x, y)\) is smooth and \(P(x, y)\) is nonsmooth, convex, lower semi-continuous, and separable for \(x\) and \(y\). The sequence \(\{x^i, y^i\}_{i=0}^\infty\) produced by alternating minimization
\[
x^{i+1} = \arg\min_x F(x, y^i) \quad \text{(2)}
\]
\[
y^{i+1} = \arg\min_y F(x^{i+1}, y) \quad \text{(3)}
\]

(a) converges to a stationary point of (1), if in (2) and (3), or their equivalent problems\(^1\), the objective functions have smooth parts \(f(x), f(y)\), that are strictly convex quadratic. (b) with linear convergence rate, if \(f(x, y)\) is quadratic (maybe nonconvex), \(P(x, y)\) is polyhedral, in addition to assumption in (a).

**Proof.** Let the smooth parts of objective function in (2) and (3), or their equivalent problems, be \(f(x)\) and \(f(y)\). We can define a special case of Coordinate Gradient Descent (CGD) proposed in [1] which yields the same sequence \(\{x^i, y^i\}_{i=0}^\infty\) as that produced by the alternating minimization, where for each iteration \(k\) of the CGD algorithm, we use hessian matrix \(H_{xx}^k = \nabla^2 f(x) \succ 0\), for block \(J = x\) and \(H_{yy}^k = \nabla^2 f(y) \succ 0\), for block \(J = y\), with exact line search (which satisfies Armijo Rule)\(^2\). By Theorem 1 of [1], the sequence \(\{x^i, y^i\}_{i=0}^\infty\) converges to a stationary point of (1). If we further assume that \(f(x, y)\) is quadratic and \(P(x, y)\) is polyhedral, by applying Theorem 2 and 4 in [1], the convergence rate of \(\{x^i, y^i\}_{i=0}^\infty\) is at least linear.

**Lemma 1.2.** There exist \(\epsilon_0 > 0\) and \(d^*\) such that the problem
\[
\arg\min_{d \in \mathbb{R}^m} \quad -c^T d + \frac{\epsilon}{2} \|d\|^2 \\
s.t. \quad 0 \leq d_i \leq 1, \quad i = 1..m \quad (4)
\]
has the same optimal solution \(d^*\) for \(\forall \epsilon < \epsilon_0\).

**Proof.** Let set \(S = \{i | c_i > 0\}\). As \(\epsilon = 0\), we can obtain an optimal solution \(d^*\) by setting \(d^*_i = 1\) and \(d^*_i = 0\). Let \(\alpha_i, \beta_i\), be Lagrange multipliers of constraints \(d_i \geq 0\) and \(d_i \leq 1\) in (4) respectively. By KKT condition, the solution \(d^*, \alpha^*, \beta^*\) must have \(-c_i - \alpha_i^* + \beta_i^* = 0\), with \(\alpha_i^* = 0\), \(\beta_{i \in S} = c_i\), and \(\beta_{i \notin S} = 0\). When \(\epsilon > 0\), the KKT condition for (4) only differs by the equation \(\epsilon d^*_i - c_i - \tilde{\alpha}_i - \tilde{\beta}_i = 0\), which can be satisfied by setting \(\tilde{\beta}_{i \in S} = \tilde{\beta}_{i \notin S} = -\epsilon\), and others the same as before. If \(\epsilon < \beta_i^* = c_i\) for \(\forall i \in S\), then \(\tilde{\beta}_{i \in S}\) can still be positive, and \(d^*, \tilde{\alpha}, \tilde{\beta}\) still satisfy the new KKT condition. In other words, \(d^*\) is still optimal if \(\epsilon < \epsilon_0\), where \(\epsilon_0 = \min_{i \in S} c_i > 0\).

---

\(^1\)By “equivalent”, we means the two problems have the same optimal solution.

\(^2\)Note, in [1], the Hassian matrix \(H^k\) affect CGD algorithm only through \(H^k_{JJ}\), so we can always have a positive-definite \(H^k\) when \(H^k_{JJ}\) is positive definite by assigning, other than \(H^k_{JJ}\), 1 to diagonal elements, 0 to non-diagonal elements.

\(^3\)By “equivalent”, we means the two problems have the same optimal solution.
Theorem 1.3. The sequence \( \{w^t\}_{t=0}^\infty \) produced by the sequential relaxation in section 2-(5) converges to a stationary point of truncated-loss problem (section 2-(2)) with at least linear convergence rate.

Proof. First, as in the proof sketch, the sequential relaxation in section 2-(5) can be interpreted as an alternating minimization (block coordinate descent) between \( d \in \mathbb{R}^m \) and \( (w, \xi) \in (\mathbb{R}^n, \mathbb{R}^m) \) on the problem

\[
\min_{w, \xi, d} \frac{1}{2} \|w\|^2 + C \sum_{l \in D} d_l \xi_l + (1 - d_l)(1 + s)
\]

s.t. \( y_l w^T x_l \leq 1 - \xi_l \)
\( \xi_l \geq 0 \)
\( 0 \leq d_l \leq 1, \quad l = 1..m \) (5)

We can define the smooth part in (1) of lemma 1.1 as \( f((w, \xi), d) = \frac{1}{2} \|w\|^2 + C \sum_{l \in D} d_l \xi_l + (1 - d_l)(1 + s) \). The constraints \( y_l w^T x_l \leq 1 - \xi_l, \xi_l \geq 0 \) form a polyhedral domain, so we can transform them into a polyhedral, lower semi-continuous function as \( P_{\text{dom}}(w, \xi) = 0 \) when \( y_l w^T x_l \leq 1 - \xi_l \) and \( \xi_l \geq 0 \); \( P_{\text{dom}}(w, \xi) = \infty \), otherwise. By the same way, the domain constraints on \( d \) can be also transformed into polyhedral, lower semi-continuous function \( P_{\text{dom}}(d) \). Then (5) can be expressed as \( \min_{(w, \xi), d} F((w, \xi), d) = f((w, \xi), d) + P((w, \xi), d) \), where \( f((w, \xi), d) \) is quadratic and \( P((w, \xi), d) = P_{\text{dom}}(w, \xi) + P_{\text{dom}}(d) \) is polyhedral, lower-semi-continuous, separable for \((w, \xi)\) and \(d\). When alternating minimizing \( F((w, \xi), d) \), the smooth part of subproblem \( \min_{w, \xi} F((w, \xi), d^t) \) is strictly convex with smooth part \( \frac{1}{2} \|w\|^2 \) being quadratic:

\[
\min_{w, \xi} \frac{1}{2} \|w\|^2 + C \sum_{\{d_l = 1\}} L(y_l w^T x_l)
\] (6)

The subproblem \( \min_{d} F((w, \xi)^{t+1}, d) \) is a linear program in the form of (4) with \( \epsilon = 0 \). By Lemma 1.2, it also has equivalent strictly convex quadratic problem. Hence, the alternating minimization of \( F((w, \xi), d) \), that is, the sequential relaxation in section 2-(5), converges at least linearly to a stationary point of section 2-(2) by Lemma 1.1.

References