PPDSparse: A Parallel Primal-Dual Sparse Method for Extreme Classification

Ian E.H. Yen† Xiangru Huang‡ Wei Dai†,* Pradeep Ravikumar† Inderjit Dhillon‡ Eric Xing†,*

†Carnegie Mellon University  ‡University of Texas at Austin  *Petuum Inc.

{eyan, pradeepr}@cs.cmu.edu, {xrhuang, iinderjit}@cs.utexas.edu, {wei.dai, eric.xing}@petuum.com

ABSTRACT

Extreme Classification comprises multi-class or multi-label prediction where there is a large number of classes, and is increasingly relevant to many real-world applications such as text and image tagging. In this setting, standard classification methods, with complexity linear in the number of classes, become intractable, while enforcing structural constraints among classes (such as low-rank or tree-structure) to reduce complexity often sacrifices accuracy for efficiency. The recent PD-Sparse method addresses this via an algorithm that is sub-linear in the number of variables, by exploiting primal-dual sparsity inherent in a specific loss function, namely the max-margin loss. In this work, we extend PD-Sparse to be efficiently parallelized in large-scale distributed settings. By introducing separable loss functions, we can scale out the training, with network communication and space efficiency comparable to those in one-versus-all approaches while maintaining an overall complexity sub-linear in the number of classes. On several large-scale benchmarks our proposed method achieves accuracy competitive to the state-of-the-art while reducing the training time from days to tens of minutes compared with existing parallel or sparse methods on a cluster of 100 cores.

KEYWORDS

Extreme Classification, Primal Dual Method, Sparse Optimization.

1 INTRODUCTION

Extreme Classification comprises Multiclass and Multilabel prediction with a large number of classes, and has become increasingly prevalent in real-world applications such as text and image tagging, where the number of tags can easily go beyond tens or even hundreds of thousands.

In such a setting, standard approaches such as one-versus-all and one-versus-one become intractable due to their high training and prediction complexity (that is at least linear) w.r.t. the number of classes. On a benchmark data set with hundreds of thousand of classes, the training time of the one-versus-all approach could take a few months [11, 34], with a model taking hundreds of gigabytes to store.

Structural Constraints. Many approaches have since been proposed to address this space and computational complexity by imposing structural constraints among the sub-models for each class, such as embedding or hierarchy based, among others. One of the most commonly used structural constraint is based on low-rank embedding [5, 17, 36], which projects parameters of different classes to a low-dimensional subspace, and thus reduces the effective number of classes. However, such a low-rank assumption often be violated in real-world data, for instance data with a power-law label distribution. Finding a globally optimal tree that maximizes performance measure is computationally hard; note that a greedy partitioning method could potentially have large cumulative error due to cascading effects. As a result, one often needs to train a large number of trees to get reasonable performance. Moreover, in practice, a good hierarchy might not exist in the first place, so that the tree-based approach often has to trade accuracy for efficiency.

Primal and Dual Sparsity. As [34] pointed out, when the number of classes become large, the collection of model parameters is under-determined, so that a simple structural constraint of sparsity might be practically useful. For the specific max-margin loss, [34] showed that there exists a naturally sparse solution to the Extreme Classification problem with sub-linear number of non-zeros in both primal and dual variables, which can be exploited to develop an estimation algorithm with sub-linear dependency on the number of classes. Since primal-dual sparsity is naturally satisfied in the Extreme Classification setting, the estimation algorithm often leads to higher accuracy on problems with larger number of classes. However, the max-margin loss employed in [34] has several disadvantages:
We consider the standard work for Multilabel/Multiclass classification, where we are given a collection of training samples \( \mathcal{D} \) with weight truncation (for model compression) can reduce the training time substantially via parallelization, albeit the approach has no theoretical guarantee on the resulting model quality. This leads to the question: can we develop a method that enjoys both the parallelizability, small memory footprint of the one-versus-all technique and the sub-linear complexity of primal-dual sparse method?

In this work, we propose a greedy algorithm that enjoys both the low runtime complexity inherent in the primal-dual sparse approach and one-versus-all’s simple parallelization training with small memory footprint. Our specific contributions are as follows:

- We show that the loss optimized by the common one-versus-all technique also enjoys a similar primal-dual sparse structure presented in [34] when a class-wise bias is added.
- We propose a greedy active-set algorithm that optimizes parameters of each class separately without communication and thus enjoys both parallelizability and primal-dual sparsity.
- We extend the analysis in [34] in two ways: (i) we bound the number of non-zero dual variables in terms of the number of positive samples instead of the number of confusing samples that could be growing linearly with the total number of samples for the separable loss considered; (ii) the bound holds not only for the optimal solution but also for any descent iterates during the optimization, which leads to a more realistic analysis for the sub-linear complexity of the algorithm.
- In our experiments on several benchmark data sets with hundreds of thousands of classes, our proposed method achieves accuracy competitive with state-of-the-art. On a cluster of 100 cores, our method is orders of magnitude faster than the existing parallel one-versus-all methods and the sequential PD-Sparse.

2 PROBLEM SETUP AND BACKGROUND

We consider the standard Empirical Risk Minimization (ERM) framework for Multilabel/Multiclass classification, where we are given a collection of training samples \( \{(x_i, y_i)\}_{i=1}^N \) with \( x_i \in \mathbb{R}^D \) denoting \( D \)-dimensional feature vectors that encode relevant information about the instances, and \( y_i \in \{-1, 1\}^K \) denoting label vectors such that \( y_{ik} = 1 \) if \( k \) is a correct label for the \( i \)-th sample and \( y_{ik} = -1 \) otherwise. We denote \( \mathcal{P}(y) = \{k | y_{ik} = 1\} \) as the set of positive labels and \( \mathcal{N}(y) = \{k | y_{ik} = -1\} \) as the set of negative ones for each sample.

Extreme Classification refers to cases when \( K \) is extremely large, sometimes as large as \( N \). On the other hand, the number of positive labels per sample \( |\mathcal{P}(y_i)| \) is usually pretty small. In Multiclass classification, we have \( |\mathcal{P}(y_i)| = 1 \), and in most Multilabel prediction problems, \( |\mathcal{P}(y_i)| \) is typically in the order of tens or less than 10.

In this work we consider classifiers \( h : \mathbb{R}^D \rightarrow \{-1, 1\}^K \) of the form

\[
h_W(x) = \text{sign}(W^T x + b)
\]

where \( W \) is a \( D \times K \) matrix. Each column of \( W \) is denoted as \( w_k \), which corresponds to the model parameters of \( k \)-th class. For such linear classifiers, given a loss function \( L : \mathbb{R}^N \times \{-1, 1\}^K \), we are interested in the ERM approach to classification that solves the following problem

\[
W^* \in \arg \min_{W \in \mathbb{R}^{D \times K}} \sum_{i=1}^N L(W^T x_i, y_i).
\]

2.1 Primal & Dual Sparsity

The estimation problem (1) is said to have primal sparsity if the solution \( W^* \) has a small number of non-zeros \( \text{nnz}(W^*) \). Interestingly, for specific loss functions, the optimization problem could also have dual sparsity.

[34] specifically considered the max-margin loss [8, 9]:

\[
L(z, y) = \max_{k \in \mathcal{N}(y)} \max_{p \in \mathcal{P}(y)} (1 + z_k - z_p) + .
\]

For a given \((z, y)\), define the set of active labels as those labels \( k \) that attain the maximum of (2). In other words, a label is active if it is most confusing; note that a negative label is more confusing if it gets a higher score, while a positive label is more confusing if its score is lower. Letting \( k_a \) denote the average number of active labels ranging over the training dataset, the optimization problem in (1) with the max-margin loss (2) is said to be dual sparse if \( k_a \) is small relative to \( K \), which entails that only few of the dual variables are non-zeros at a minimizer of the ERM problem. [34] showed that this is indeed typically the case in extreme classification settings, and proposed a method they called PD-Sparse that exploits this fact.

[34] further showed that, under the loss (2), if one has \( k_a \) average number of active labels, there exists a minimizer of the ERM problem (1) with \( \text{nnz}(W^*) \leq N k_a \), which can moreover be the unique solution by augmenting the objective (1) with an arbitrarily small \( \ell_1 \)-penalty. In other words, the dual sparsity (small \( k_a \)) implies primal sparsity (small \( \text{nnz}(W^*) \)) if

\[
k_a := \frac{\text{nnz}(W^*)}{D} \leq \frac{N k_a}{D} \ll K,
\]

This results in a model of both primal and dual sparse structure as long as \( DK \gg Nk_a \). Note this result is significant since the sparse solution is not enforced but identified. In other words, it is not a trade-off between accuracy and sparsity. Instead, when \( DK \gg Nk_a \), there are many more parameters than constraints in the problem (1), and there exits a very sparse solution among the many possible solutions.

3 FORMULATION: PRIMAL-DUAL SPARSITY WITH SEPARABILITY

A significant disadvantage of the loss (2) is that it is not separable w.r.t. the class parameters \( w_1, \ldots, w_K \), and therefore it requires training all parameters \( W \) together. This incurs a much larger
memory consumption than the one-versus-all approach even in the presence of sparsity. This prohibits PD-Sparse from using a simple parallelization scheme that assigns the training of different classes to different cores—a scheme that could enjoy nearly linear speedup to even a thousand of cores in [1].

To achieve parallelizability and space efficiency, we consider the following class-separable hinge loss

$$L(z, y) := \sum_{k=1}^{K} f(z_k, y_k) = \sum_{k=1}^{K} \max(1 - y_k z_k, 0)$$  \hspace{1cm} (3)

One-versus-all method can be interpreted as minimizing (3) since

$$\min_{w \in \mathbb{R}^{D \times K}} \frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} \ell(w^T_k x_i, y_{ik}) = \frac{1}{N} \sum_{i=1}^{N} \left( \sum_{k=1}^{K} \ell(w^T_k x_i, y_{ik}) \right)$$  \hspace{1cm} (4)

The goal of this section is to show that (4) also permits a primal-dual sparse structure similar to [34] when a bias term is added to the parameters of each class. We first note that showing (3) has dual sparsity is more difficult, since unlike (2), it penalizes each class separately, so there could potentially be many more active labels for each sample as we discuss below. We thus take two different approaches to show the desired dual sparsity structure.

### 3.1 Dual Sparsity: Active Labels

We first employ an approach similar to [34], and try to establish dual sparsity in terms of the number of the active labels of each sample $i$. In our case, the set of active labels are characterized as

$$C_i := \{k \mid y_{ik} \langle w_k, x_i \rangle < 1\},$$

which are either wrong predictions or confidence scores $\leq 1$ for a particular sample $i$. This is related to the set of support vectors of each class $S_k := \{i \mid y_{ik} \langle w_k, x_i \rangle \leq 1\}$. Denote $k_a := \frac{1}{N} \sum_{i=1}^{N} |C_i|$ and $n_a := \frac{1}{K} \sum_{k=1}^{K} |S_k|$, as the average number of active labels and support vectors. We have

$$N k_a = K n_a.$$

Note when the number of active labels $k_a$ is constant, we have $n_a = N k_a / K$ decreases with $K$, which results in the dual sparsity when $K \gg k_a$.

The following theorem then shows that the dual sparsity also implies a primal-dual sparse solution when $n_a \ll D$.

**Theorem 3.1.** Let $\lambda > 0$ be an arbitrarily small constant and

$$w^* \in \arg \min_{w \in \mathbb{R}^D} \lambda \|w\|_1 + \sum_{i=1}^{N} \ell(w^T x_i, y_i).$$  \hspace{1cm} (5)

Then for $\{x_i\}_{i=1}^{N}$ drawn from a continuous distribution we have

$$d_k := \text{nnz}(w^*) \leq n_a.$$  \hspace{1cm} (6)

where $n_a := \{i \mid y_i (w^*, x_i) \leq 1\}$ is the number of support vectors.

**Proof.** Let $X$ be an $N \times D$ feature matrix with rows $\{x_i\}_{i=1}^{N}$. Any optimal solution of (5) satisfies

$$\lambda \rho^* + \sum_{i=1}^{N} \alpha_i^* x_i = \lambda \rho^* + X \alpha^* = 0$$  \hspace{1cm} (7)

for some $\rho^* \in \partial \|w^*\|_1$ and $\alpha_i^* \in \partial \ell(z_i, y_i)$ with $z_i := \langle w^*, x_i \rangle + b$. Then since hinge loss (and square hinge loss) has $\alpha_i^* \neq 0$ only when $y_i z_i^* \leq 1$, there are at most $n_a$ non-zeros $\alpha_i^*$ in the linear system (7). On the other hand, the subgradient $\rho^*$ of $\|w^*\|_1$ satisfies

$$\rho^* = \begin{cases} 1, & w_i^* > 0 \\ -1, & w_i^* < 0 \\ 0, & v \in [-1, 1], w_i^* = 0. \end{cases}$$

Let $B := \{j \mid w_j^* \neq 0\}$ be the indexes with non-zeros weights. Then consider equations given by the non-zeros $\alpha_i$ and $|B|$ rows in (7)

$$-\lambda \text{sign}(w^*)_B = X^T \alpha^*.$$  \hspace{1cm} (8)

It has $\text{nnz}(w^*)$ equations and $n_a$ variables, which, for a feature matrix at general position, can be satisfied only if

$$\text{nnz}(w^*) = |B| \leq n_a.$$  \hspace{1cm} (9)

A feature matrix $X$ is always at general position if its rows $\{x_i\}_{i=1}^{N}$ are drawn from a continuous distribution [38].

Note the Theorem 3.1 implies that when the number of active labels $k_a$ is constant, both the number of non-zeros in the primal variables $d_k$ and dual variables $n_a$ are proportional to $\frac{N k_a}{K}$. However, there are several drawbacks with Theorem 3.1. First, the loss (3) does not force $k_a$ to be small as the max-margin loss (2) does, so in our case $k_a$ could actually increase linearly with $K$. Second, Theorem 3.1 only analyzes the optimal solution $(\alpha^*, w^*)$, which cannot however guarantee the sparsity of $w$ during the intermediate iterates of the training algorithm. Finally, the result (6) holds only for a purely $\ell_1$-regularized objective, which is known to yield a non-smooth dual objective that can be hard to optimize in a coordinate-wise fashion. In next section, we thus employ a different approach to resolve these caveats.

### 3.2 Dual Sparsity: Positive Examples

In this section, we take a more realistic approach which bounds the $\ell_1$-norm of primal and dual variables by the number of positive examples of each class, which decreases as a function of $K$ even under the separable loss (3). Denote $n_a$ as the average number of positive samples per class, and $k_p$ as the average number of positive labels per sample. We have

$$n_p := \left(\frac{k_p}{K}\right) N \ll N.$$  \hspace{1cm} (10)

when $K$ is large. Note unlike the number of active labels $k_a$, the number of positive labels $k_p$ is a constant that does not grow with $K$ in most of Extreme Classification problems. Therefore, the number of positive samples per class $n_p$ is decreasing with $K$ since $n_p = \frac{N k_p}{K}$.

Now consider a model that incorporates the bias term, where we have an additional feature $x_{i,0} = 1$ for all samples, and an additional weight $w_{ik,0}$ for all classes. For ease of optimization in the dual, we consider the following $\ell_1-\ell_2$ (Elastic-Net) regularized objective :

$$\min_{w_k \in \mathbb{R}^D} F(w_k) := \sum_{j=1}^{D} \left| w_j k + \frac{1}{2} \|w_k\|^2 + \sum_{i=1}^{N} \ell(w^T_k x_i, y_{ik}) \right. \hspace{1cm} (11)$$
which has a dual objective of the form
\[
\min_{\alpha_k \in \mathbb{R}^N} G(\alpha_k) := \frac{1}{2} \|\alpha(\alpha_k)\|^2 = -\sum_{i=1}^{N} \alpha_{ik}
\]
\[
s.t. \quad w(\alpha_k) = \text{prox}_{\lambda (\cdot)} (\tilde{X}^T \alpha_k),
\]
\[
0 \leq \alpha_{ik} \leq 1.
\]
where \( \tilde{X} \) is an \( N \times D \) matrix with rows \( \{y_{ik}x_i\}_{i=1}^{N} \), and \( \text{prox}_{\lambda (\cdot)} \) is the proximal operator of the function \( \lambda \sum_{j=1}^{D} |w_j| \). Note we do not penalize bias term in the \( \ell_1 \) regularization, so \( w_0 = [\tilde{X}^T \alpha_k]_0 \). Then the following two theorems bound the \( \ell_1 \)-norm of primal and dual parameters respectively.

**Theorem 3.2 (\( \ell_1 \)-norm of primal variables).** Let \( \hat{w} = (\hat{w}_0 = -1, \hat{w}_{-0} = 0) \) be a trivial solution. For any \( w_k \) with \( F(w_k) \leq F(\hat{w}) \), we have
\[
\|w_k\|_1 \leq \frac{2n_p^k}{\lambda}.
\]
where \( n_p^k \) is the number of positive samples of \( k \)-th class.

**Proof.** \( \hat{w} \) satisfies
\[
F(\hat{w}) = \frac{1}{2} + n_p^k \leq 2n_p^k
\]
since \( \ell(-1, y_{ik}) = 0 \) for \( y_{ik} = -1 \) and \( \ell(0, y_{ik}) = 1 \) for \( y_{ik} = 1 \). Then for any \( w_k \) of better objective than \( w \),
\[
\lambda \|w_k\|_1 \leq F(w_k) \leq F(\hat{w}) \leq 2n_p^k
\]
which yields the result. \( \square \)

**Theorem 3.3 (\( \ell_1 \)-norm of dual variables).** Any optimal solution \( \alpha_k \) of (10) satisfies
\[
\|\alpha^*_k\|_1 \leq 4n_p^k.
\]

**Proof.** Let \( w, \alpha \) be the primal and dual variables for a particular class. By strong duality, the optimal dual objective (in its maximization form) equals to the optimal primal objective
\[
\max_{\alpha} -G(\alpha) = \min_{w} F(w)
\]
and therefore, any optimal solution \( (\alpha^*, w^*) \) satisfies
\[
-\frac{1}{2} \|w^*\|^2 + \sum \alpha^*_i = \frac{1}{2} \|w^*\|^2 + \lambda \sum_{j=1}^{D} |w_j^*| + \sum_{i=1}^{N} \ell((w^*, x_i), y_{ik})
\]
which yields the bound
\[
\|\alpha^*_k\|_1 = \|w^*\|^2 + \lambda \sum_{j=1}^{D} w_{j,k}^* + \sum_{i=1}^{N} \ell((w^*, x_i), y_{ik})
\]
\[
\leq 2 \left( \frac{1}{2} \|w^*\|^2 + \lambda \sum_{j=1}^{D} |w_j^*| + \sum_{i=1}^{N} \ell((w^*, x_i), y_{ik}) \right) \leq 4n_p^k
\]
where the last inequality is due to the existence of \( \hat{w} = (\hat{w}_0 = -1, \hat{w}_{-0} = 0) \) with primal objective \( F(\hat{w}) \leq 2n_p^k \). \( \square \)

Theorem 3.2 and 3.3 successfully bound the \( \ell_1 \)-norm of primal, dual variables by \( n_p = O(N/K) \). Although a small \( \ell_1 \)-norm does not imply small number of non-zeros in general, we show in the next section that the complexity of our algorithm is determined by the \( \ell_1 \) norm of primal and dual variables. In particular, in Section 4.2, we propose a random sparsification procedure that finds a sparse approximation to \( w_k \) in order to perform efficient greedy search of active coordinates. The procedure is guaranteed to find a sparse solution with number of non-zeros proportional to \( \|w_k\|_0^2 \). Further, Section 4.3 gives an iteration complexity, and thus a bound on the active size, proportional to \( \|\alpha^*\|_1 \), where \( \alpha^* \) is an optimal solution of (10).

## 4 ALGORITHM

In this section, we propose a greedy algorithm that alternates between the search of potential support vectors and the optimization over a small set of active samples. Note we have an objective (9) for each class \( k \) independent of other classes, so in the following we simply use \( \alpha \) to denote \( \alpha_k \) and \( w \) to denote \( w_k \) for a particular class \( k \) being solved. The algorithm will be performed for each class independently without communication, and thus it is inherently easy to parallelize.

### 4.1 A Greedy Active-Set Method

Instead of searching for the confusing labels of each sample as in [34], we propose a greedy algorithm that optimizes (10) by looking for active samples of each class. Note the domain of our objective (10) has box constraints \( 0 \leq \alpha_i \leq 1 \) instead of simplex constraints as in [34], so a Frank-Wolfe-based algorithm used in [34] does not lead to sparse iterates. Here we propose a greedy active-set coordinate descent algorithm that alternates between the greedy search of novel active variables and the optimization over an active set.

The objective (10) has gradient of the form
\[
\nabla G(\alpha) = \tilde{X}^T w(\alpha) - 1,
\]
which can be evaluated in time \( O(\text{nnz}(w)\bar{n}) \) if the vector \( w(\alpha) \) is maintained whenever any dual coordinate \( \alpha_i \) is changed, where \( \bar{n} \) is an upper bound on the number of non-zero in each column of \( \tilde{X} \). On the other hand, when \( \alpha \) is changed by \( \Delta \alpha \), the maintenance of
\[
w(\alpha + \Delta \alpha) = \text{prox}_{\lambda} \|\cdot\|_1 (\tilde{X}^T \alpha + \tilde{X}^T \Delta \alpha)
\]
requires a cost of \( O(\text{nnz}(\Delta \alpha)\bar{d}) \) where \( \bar{d} \) is an upper bound on the number of non-zeros for each row of \( \tilde{X} \). Note one can exploit the sparsity of both \( \tilde{X} \) and \( w(\alpha) \) simultaneously when computing the gradients of all coordinates together as
\[
\nabla G(\alpha) = \sum_{j=0}^{D} w_j \tilde{X}_{-, j},
\]
Therefore, an efficient algorithm exploits (14) to compute gradients of all coordinate simultaneously while updates only a small number of them to ensure a small \( \text{nnz}(\Delta \alpha) \). This suggests a greedy strategy that optimizes only coordinates \( \alpha_i \) leading to the most progress. The resulting algorithm is summarized in Algorithm 1, where we perform an inner minimization over active set via Randomized Dual Coordinate Descent (Algorithm 2) [13]. The cost of one epoch of Algorithm 2 over the active set is \( O(|\mathcal{A}| \bar{d}) \). Therefore, each iteration of Algorithm 1 has an overall cost of \( O(\text{nnz}(w)\bar{n} + |\mathcal{A}| \bar{d}) \). Note this is a cost sublinear to the size of data matrix \( \text{nnz}(X) \) if \( \text{nnz}(w) \ll D \) and \( \text{nnz}(\alpha) \ll N \). In section 4.2 and 4.3, we give bounds on \( \text{nnz}(w) \) and \( |\mathcal{A}| \) that decreases with number of classes \( K \).
Algorithm 1 Greedy Active-Set Algorithm

0. $\alpha = 0$, $b = 1$, $\mathcal{A} = \{i | y_{ik} = 1\}$ and $H_i = ||x_i||^2, i \in [N]$. 

for $t = 1,...,T$ do 
1. Compute $\nabla G(\alpha)$ via random sparsification (Algorithm 3).
2. $\mathcal{A} \leftarrow$ Pick $k$ variables $\not\in \mathcal{A}$ of largest $-\nabla G(\alpha)$.
3. Minimize (10) w.r.t. coordinates in $\mathcal{A}$ via Algorithm 2.
4. Eliminate $\{i | \alpha_i = 0 \text{ or } y_{ik} \neq 1\}$ from $\mathcal{A}$.
end for 

Algorithm 2 Coordinate Descent for Active Subproblem 

for $s = 1,...,S$ do 
1. Draw $i$ uniformly at random.
2. Compute $\nabla_i G(\alpha) = y_i(\hat{w}, x_i) - 1$.
3. $\Delta \alpha_i \leftarrow \min(\max(\alpha_i - \nabla_i G(\alpha)/H_i, 0), U) - \alpha_i$
4. Maintain (13) with update $\Delta \alpha_i$.
end for 

4.2 Sublinear-Time Search via Random Sparsification

Given an $\ell_1$ norm bounded by (11), we show that the maximum negative gradient found in Step 2 of Algorithm 1 can be approximated with $\delta$ precision when replacing $w(\alpha)$ with its random sparsified version $\hat{w}$ obtained from Algorithm 3, where the number of nonzeros in $\hat{w}$ is bounded by the square of $\ell_1$-norm as stated in the following theorem.

Theorem 4.1. Running the Random Sparsification procedure 3 for $R = \left\lfloor \frac{2||w||_1^2}{\delta^2} \right\rfloor$ iterations gives a $\hat{w}$ satisfying

$$\text{nnz}(\hat{w}) \leq \left( \frac{4n^2_p}{\alpha^2} \right) \frac{1}{\delta^2},$$

with

$$\mathbb{E}[\min_i y_i(\hat{w}, x_i)] - \min_i y_i(\hat{w}, x_i) \leq \delta.$$

Proof. Since the function $f(z) = \min_i z_i$ is 1-Lipschitz-continuous, we have

$$\min_i y_i(\hat{w}, x_i) - \min_i y_i(\hat{w}, x_i) \leq |(\hat{w}, x_i) - (w, x_i)|.$$  

Taking expectation over $\hat{w}$ on both sides, we have

$$\mathbb{E}[\min_i y_i(\hat{w}, x_i)] - \min_i y_i(\hat{w}, x_i) \leq \mathbb{E}[(\hat{w}, x_i) - (w, x_i)]^2 \leq \frac{\mathbb{E}[(\hat{w}, x_i) - (w, x_i)]^2}{2}$$

from Jensen’s inequality. Since $\mathbb{E}[(\hat{w}, x_i)] = (w, x_i)$ by the construction of $\hat{w}$ in Algorithm 3, the RHS of (17) corresponds to the square root of variance

$$\mathbb{E}[\min_i (\hat{w}, x_i)] \leq \sqrt{\text{Var}[\hat{w}, x_i]} = \frac{||w||_1}{\sqrt{R}}.$$  

The conclusion follows by noticing that $\text{nnz}(\hat{w}) \leq R$ and $||w||_1$ satisfies (11).

Note that (16) implies the greedy coordinate $i$ found by approximate search and the coordinate found by exact search $i^* = arg \min_i y_i(\hat{w}, x_i) - 1$ satisfy

$$\frac{1}{\alpha^*} \leq \frac{1}{\alpha^*}$$

4.3 Convergence Analysis

In this section, we give an iteration complexity of Algorithm 1 that depends on the $\ell_1$ norm of the optimal solution $\alpha^*$. For simplicity of the analysis, we assume that a normalized feature matrix with $||x_i||_2 \leq 1$.

Theorem 4.2. Let $\alpha^*$ be an optimal solution of (10). The iterates $\{\alpha^t\}_{t=1}^\infty$ given by Algorithm 1 with Random Sparsification tolerance $\delta \leq \frac{\epsilon}{4\|\alpha^*\|}$ has $||G(\alpha^t) - G(\alpha^*)|| \leq \epsilon$ for any iterate

$$t \geq \frac{4\|\alpha^*\|^2_1}{\epsilon} + G(0) - G^*.$$  

Proof. Our dual objective (10) is of the form

$$g_\alpha + h_\alpha$$

where $h_\alpha := \sum_{i=1}^N h_i(\alpha_i)$ and

$$h_i(\alpha) = \begin{cases} 0, & 0 \leq \alpha_i \leq 1 \\ \infty, & \text{otherwise} \end{cases}$$

and $g(\alpha) = \frac{1}{2} ||w(\alpha)||_2^2 - \sum_{i=1}^N \alpha_i$ is smooth with Lipschitz-continuous gradient $\nabla g(\alpha)$. To see this, let $B_\alpha = \{j | y_{ij}(\alpha_i) \neq 0\}$. The generalized Hessian of $g(\alpha)$ is

$$\nabla^2 g(\alpha) = X_iB_iX_i^T$$

which has diagonal elements $||x_iB_i||_2^2$ bounded by $||x_i||_2^2 \leq 1$ and thus a spectral norm bounded by $N$. Then for any coordinate $i$, we have the following descent amount since the second derivative of the smooth part of objective is bounded by $||x_i||_2^2$.

$$\min_{\eta, \frac{\eta}{\eta}} G(\alpha + \eta e_i) - G(\alpha) \leq \min_{\eta} \nabla_i G + \frac{1}{2} \eta^2 + h_i(\alpha) + \eta$$

where $e_i$ is an indicator vector. Note for $i \in \mathcal{A}$, the minimizer of RHS of (19) is 0 since the previous iteration already minimizes our
objective w.r.t. the active set \( \mathcal{A} \). And for \( i \notin \mathcal{A} \), the minimizer of the RHS of (19) is \(-\nabla_i G(\alpha)\|\alpha_i\|_2^2/2\), which corresponds to the selection criteria at Step 2 of Algorithm 1. Therefore, at each iteration, the coordinate \( i \) and \( i' \) found by the approximate and exact greedy search respectively satisfy

\[

\begin{align*}
\mathbb{E}[G(\alpha + \Delta \alpha)] - G(\alpha) &\leq \mathbb{E}[\min_\eta G(\alpha + \eta e_i)] - G(\alpha) \\
&\leq \min_\eta \nabla_i G * \eta + \frac{1}{2} \eta^2 + h(\alpha + \eta) + 2\delta \sum_i \eta_i \\
&\leq \min_\eta (\nabla G, \eta) + \frac{1}{2} \|\eta\|_2^2 + h(\alpha + \eta) + 2\delta \sum_i \eta_i
\end{align*}

\]

where the first inequality is because the update \( \Delta \alpha \) is obtained by minimizing objective w.r.t. a working set \( \mathcal{A} \) containing \( i \), and the second, third inequalities follow from (18) and the fact that a linear objective subject to \( \ell_1 \) ball has minimizer at the corner respectively. Then we use convexity to obtain a global estimate of descent amount relative to the suboptimality \( G(\alpha) - G(\alpha^*) \):

\[

\begin{align*}
\mathbb{E}[G(\alpha + \Delta \alpha)] - G(\alpha) &\leq \min_\eta (\nabla G, \eta) + \frac{1}{2} \|\eta\|_2^2 + h(\alpha + \eta) + 2\delta \sum_i \eta_i \\
&\leq \min_\eta q (\nabla G, \alpha^* - \alpha) + \frac{q^2}{2} \|\alpha^*\|_2^2 + 2q\delta \|\alpha^*\|_1 \\
&\leq \min_\eta q (\nabla G, \alpha^* - \alpha) + \frac{q^2}{2} \|\alpha^*\|_2^2 + 2q\delta \|\alpha^*\|_1
\end{align*}

\]

where the last inequality is from convexity, and the second inequality is from a restriction of optimization space to \( \eta = q(\alpha^* - \alpha) \) and the fact that for \( i \notin \mathcal{A} \) the minimizer of (21) has \( \eta_i = 0 \). Then choosing \( \delta \leq \frac{G(\alpha) - G(\alpha^*)}{4\|\alpha^*\|_1} \) and minimizing the RHS w.r.t. \( q \) leads to

\[
\mathbb{E}[G(\alpha + \Delta \alpha)] - G(\alpha) \leq \frac{(G(\alpha) - G(\alpha^*))^2}{4\|\alpha^*\|_1^2}
\]

for iterates with \( G(\alpha) - G(\alpha^*) \leq 2\|\alpha^*\|_1^2 \) and has

\[
\mathbb{E}[G(\alpha + \Delta \alpha)] - G(\alpha) \leq -\frac{1}{2}\|\alpha^*\|_1^2/2
\]

for iterates with \( G(\alpha) - G(\alpha^*) > 2\|\alpha^*\|_1^2 \). Note the constant descent amount (25) happens only in the beginning iterates when \( G(\alpha) - G(\alpha^*) \) and can happen at most \( 2(G(0) - G(\alpha^*))/\|\alpha^*\|_1^2 \) times. Considering those iterates of case (24), we have recursive relation

\[
\Delta G^{t+1} - \Delta G^t \leq -\frac{(\Delta G^t)^2}{\|\alpha^{t+1}\|_2^2}
\]

where \( \Delta G^t := \mathbb{E}[G(\alpha^t) | \alpha^{t-1}] - G(\alpha^*). \) The recursion leads to the conclusion by, for example, Theorem 1 of [26].

Theorem 4.2 is significant when combined with Theorem 3.3, which gives us an iteration complexity of

\[
t = \frac{4\|\alpha^*\|_1^2}{\epsilon} \leq \frac{64n^2_p}{\epsilon},
\]

and also a bound on the active size \( |\mathcal{A}| \leq \kappa t \leq \kappa \frac{64n^2_p}{\epsilon} \) that depends only on the number of positive examples. Considering the average case where \( n_p = Nk_p/K \), for achieving \( \frac{1}{K}(G(\alpha) - G^*) \leq \epsilon \in (0, 1) \), we have

\[
|\mathcal{A}| = O\left(\frac{Nk^2_p}{K^2\epsilon}\right).
\]

Then the complexity for running Algorithm 1 on all classes is:

\[
K \times O(nnz(\hat{\nu})n + |\mathcal{A}|\hat{d}) = O\left(\frac{k^2_p n}{K^2\epsilon^2} + nnz(X)\frac{k^2_p}{K}\right),
\]

times the number of iterations, which is a cost proportional to the factor \( k^2_p/K \).

5 PRACTICAL ISSUES

5.1 Parallelization Architecture

The dual objective (10) comprises independent sub-objectives for different classes, and which moreover only differ by the set of labels \( y_{ik} \). Under a shared-memory architecture each thread can share the same copy of data matrix \( X \). In our implementation, we employ a two-layer parallelization with 10 computational thread per node. Dynamic load balancing is performed by one of the thread which assigns new labels to a thread once it finishes its current job. Usually one can enjoy a nearly linear speedup as long as the number of parallel jobs is an order of magnitude less than \( K \). More sophisticated communication and load balancing can also be achieved through distributed shared memory [10, 29] and scheduler [19].

5.2 Loss and Regularization Variants

In practice, we found replacing the hinge loss in (9) with a square hinge loss

\[
\ell(z, y) := \frac{1}{2} \max(1 - yz, 0)^2
\]

yields a better performance in our empirical evaluation. One can also introduce an additional tuning parameter \( \tau \) for the \( \ell_2 \) regularization in (9), although in our experiments we found \( \tau = 1 \) to be the optimal choice out of \( \{0.1, 1, 10\} \) for most of the data sets.

5.3 Two-Level Sparsification

The sparsification of \( w \) described in section 4.2 reduces the cost of computing (14) from \( O(nnz(X)) \) to \( O(nnz(\hat{\nu})n) \). For some data sets, each column of \( X \) is dense (\( n \approx N \)), but most of entries have small magnitude which contribute little to \( \nabla G(\alpha) \). A trick to further speed up the greedy search is to consider only entries in \( X \) with magnitude larger than a threshold. Then by sorting each column of \( X \) in descending order of magnitude, one can stop traversing the sorted list once processed all entries of magnitude larger than the threshold. This trick improves the efficiency of the proposed method significantly for a number of text data sets in our experiments.

6 EXPERIMENTS

In this section, we compare our proposed algorithm with state-of-the-art approaches on multiclass and multilabel problems chosen based on the experimental results shown in the Extreme Classification Repository \(^1\) and \([1, 34]\). The compared methods are:

\(^1\)https://manikvarma.github.io/downloads/XC/XMLRepository.html
Table 1: Results and statistics for large-scale Multilabel data sets, \(N_{\text{train}}\) number of training samples, \(N_{\text{test}}\) number of testing samples, \(T_{\text{train}}\) = training time, \(T_{\text{test}}\) = testing time, \(K\) = number of classes, \(D\) = number of features. P@k = top-k accuracy. DiSMEC and PPDsparse are parallelized with 100 cores. We highlight the best result for each metric, except that for \(T_{\text{train}}\) we highlight best results among single-core solvers (left four) and parallel solvers. For all experiments, we set a memory limit to be 100G. Experiments that exceeded limits are marked Memory Limit Exceeded (MLE).

<table>
<thead>
<tr>
<th>Data</th>
<th>Metrics</th>
<th>FastXML</th>
<th>PfastreXML</th>
<th>SLEEC</th>
<th>PDSparse</th>
<th>DiSMEC</th>
<th>PPDsparse</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon-670K</td>
<td>(T_{\text{train}})</td>
<td>5624s</td>
<td>6559s</td>
<td>20904s</td>
<td>174135s</td>
<td>921.9s</td>
<td></td>
</tr>
<tr>
<td>(N_{\text{train}})=490449</td>
<td>P@1 (%)</td>
<td>33.12</td>
<td>32.87</td>
<td>35.62</td>
<td>43.00</td>
<td>43.04</td>
<td></td>
</tr>
<tr>
<td>(N_{\text{test}})=153025</td>
<td>P@3 (%)</td>
<td>28.98</td>
<td>29.52</td>
<td>31.65</td>
<td>38.23</td>
<td>38.24</td>
<td></td>
</tr>
<tr>
<td>D=135909</td>
<td>P@5 (%)</td>
<td>26.11</td>
<td>26.82</td>
<td>28.85</td>
<td>34.93</td>
<td>34.94</td>
<td></td>
</tr>
<tr>
<td>K=670091</td>
<td>model size</td>
<td>4.0G</td>
<td>6.3G</td>
<td>6.6G</td>
<td>8.1G</td>
<td>5.3G</td>
<td></td>
</tr>
<tr>
<td>(T_{\text{test}}/N_{\text{test}})</td>
<td>1.41ms</td>
<td>1.98ms</td>
<td>6.94ms</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WikiLSHTC-325K</td>
<td>(T_{\text{train}})</td>
<td>19160s</td>
<td>20070s</td>
<td>39000s</td>
<td>94343s</td>
<td>271407s</td>
<td>353s</td>
</tr>
<tr>
<td>(N_{\text{train}})=1778351</td>
<td>P@1 (%)</td>
<td>50.01</td>
<td>57.17</td>
<td>58.34</td>
<td>60.70</td>
<td>64.00</td>
<td>64.13</td>
</tr>
<tr>
<td>(N_{\text{test}})=587084</td>
<td>P@3 (%)</td>
<td>32.83</td>
<td>37.03</td>
<td>36.7</td>
<td>39.62</td>
<td>42.31</td>
<td>42.10</td>
</tr>
<tr>
<td>D=161789</td>
<td>P@5 (%)</td>
<td>24.13</td>
<td>27.19</td>
<td>26.45</td>
<td>29.20</td>
<td>31.40</td>
<td>31.14</td>
</tr>
<tr>
<td>K=325056</td>
<td>model size</td>
<td>14G</td>
<td>16G</td>
<td>650M</td>
<td>547M</td>
<td>8.1G</td>
<td>4.9G</td>
</tr>
<tr>
<td>(T_{\text{test}}/N_{\text{test}})</td>
<td>1.02ms</td>
<td>1.47ms</td>
<td>4.85ms</td>
<td>3.89ms</td>
<td>65ms</td>
<td>290ms</td>
<td></td>
</tr>
<tr>
<td>Delicious-200K</td>
<td>(T_{\text{train}})</td>
<td>8832.46s</td>
<td>8807.51s</td>
<td>4838.7s</td>
<td>5137.4s</td>
<td>3881.4s</td>
<td>2689.1s</td>
</tr>
<tr>
<td>(N_{\text{train}})=196606</td>
<td>P@1 (%)</td>
<td>48.85</td>
<td>26.66</td>
<td>47.78</td>
<td>37.69</td>
<td>44.71</td>
<td>45.05</td>
</tr>
<tr>
<td>(N_{\text{test}})=100095</td>
<td>P@3 (%)</td>
<td>42.84</td>
<td>23.56</td>
<td>42.05</td>
<td>30.16</td>
<td>38.08</td>
<td>38.34</td>
</tr>
<tr>
<td>D=782585</td>
<td>P@5 (%)</td>
<td>39.83</td>
<td>23.21</td>
<td>39.29</td>
<td>27.01</td>
<td>34.7</td>
<td>34.90</td>
</tr>
<tr>
<td>K=205443</td>
<td>model size</td>
<td>1.3G</td>
<td>20G</td>
<td>2.1G</td>
<td>3.8M</td>
<td>18G</td>
<td>9.4G</td>
</tr>
<tr>
<td>(T_{\text{test}}/N_{\text{test}})</td>
<td>1.28ms</td>
<td>7.40ms</td>
<td>2.685ms</td>
<td>0.432ms</td>
<td>311.4ms</td>
<td>275ms</td>
<td></td>
</tr>
<tr>
<td>AmazonCat-13K</td>
<td>(T_{\text{train}})</td>
<td>11535s</td>
<td>13985s</td>
<td>119840s</td>
<td>2789s</td>
<td>11828s</td>
<td>122.8s</td>
</tr>
<tr>
<td>(N_{\text{train}})=1186239</td>
<td>P@1 (%)</td>
<td>94.02</td>
<td>86.06</td>
<td>90.56</td>
<td>87.43</td>
<td>92.72</td>
<td>92.72</td>
</tr>
<tr>
<td>(N_{\text{test}})=306782</td>
<td>P@3 (%)</td>
<td>79.93</td>
<td>76.24</td>
<td>76.96</td>
<td>70.48</td>
<td>78.11</td>
<td>78.14</td>
</tr>
<tr>
<td>D=203882</td>
<td>P@5 (%)</td>
<td>64.90</td>
<td>63.65</td>
<td>62.63</td>
<td>56.70</td>
<td>63.40</td>
<td>63.41</td>
</tr>
<tr>
<td>K=13330</td>
<td>model size</td>
<td>9.7G</td>
<td>11G</td>
<td>12G</td>
<td>15M</td>
<td>2.1G</td>
<td>355M</td>
</tr>
<tr>
<td>(T_{\text{test}}/N_{\text{test}})</td>
<td>1.21ms</td>
<td>1.34ms</td>
<td>13.36ms</td>
<td>0.87ms</td>
<td>0.20ms</td>
<td>1.82ms</td>
<td></td>
</tr>
</tbody>
</table>

- **FastXML** [24]: An efficient and scalable tree-based algorithm. We adopted parameter setting suggested by the solver.
- **PfastreXML** [14]: An efficient and scalable tree ensemble based method improving upon FastXML by minimizing propensity-scored loss at each tree node, which leads to better performance on tail labels. Since all other methods are not adjusted based on the propensity, in our experiment we still measure performance via traditional top-k accuracy.
- **SLEEC** [2]: A non-linear solver that 1) partitions training sample into clusters and 2) compute local embeddings that preserves nearest neighbor structure within each cluster. Because of this composition of components, its performance highly relies on its parameter setting. We adopted settings suggested by the authors for each data set.
- **PDSparse** [34]: A Primal-Dual sparse method that minimizes a max-margin loss with \(\ell_1-\ell_2\) regularization and enjoys sublinear complexity w.r.t. the number of classes.
- **DiSMEC** [1]: A distributed and parallelized method that learns one-versus-all classifiers with heuristic model compression (weight truncation). We run this method with 100 cores using the same parallelization framework to our solver.
- **PPDsparse**: The proposed method with 100 cores (10 machines with 10 cores on each machine).

All compared solvers are available from Extreme Classification Repository\(^1\). Other solvers not compared in this paper are i) one-vs-all logistic regression, one-vs-all SVM, multiclass SVM, one-vs-all \(\ell_1\)-regularized logistic regression, implemented in LibLinear [12], ii) Vowpal-Wabbit [7], iii) LEML [36], iv) RobustXML [33], v) PLT [15], vi) LPSR-NB [30]. All of these have been shown less competitive in a number of previous papers [1, 34].

Experiments are conducted on four large-scale multilabel data sets, four medium-scale multilabel data sets and three large-scale multiclass data sets. We adopt data sets used by [1, 14, 24, 34] and also that from the Extreme Classification Repository\(^1\). Large-scale multilabel data sets are WikiLSHTC-325K, Delicious-200K, Amazon-670K and AmazonCat-13K. Medium-scale Multilabel data sets are Mediamill, Biktext, RCV1-2K and EURLex-4K. They can be found at Extreme Classification Repository\(^1\). Multiclass data sets LSHTC1, Dmoz and aloi.bin are available\(^2\) from authors of [34].

The data statistics and results are shown in Table 1, 2 and 3. For all experiments, we select our hyperparameter \(\lambda\) from \([0.01, 0.1, 1]\) and \(\tau\) from \([0.1, 1, 10]\) to maximize the holdout performance. However, we observed that for most of data sets, \(\tau = 1, \lambda = 0.01\) consistently gives the best performance. For large-scale multilabel datasets

\(^1\)http://www.cs.utexas.edu/~xhuang/PDSparse/

\(^2\)http://www.cs.utexas.edu/~xrhuang/PDSparse/
Table 2: Results and statistics for small Multilabel data sets, \( N_{\text{train}} \) = number of training samples, \( N_{\text{test}} \) = number of testing samples, \( T_{\text{train}} \) = training time, \( T_{\text{test}} \) = testing time, \( K \) = number of classes, \( D \) = number of features. \( P@k \) = top-k accuracy. DiSMEC and PPDSparse are parallelized with 100 cores. We highlight the best result for each metric, except that for \( T_{\text{train}} \) we highlight best results among single-core solvers (left four) and parallel solvers.

<table>
<thead>
<tr>
<th>Data</th>
<th>Metrics</th>
<th>FastXML</th>
<th>PfastreXML</th>
<th>SLEEC</th>
<th>PDSparse</th>
<th>DiSMEC</th>
<th>PPDSparse</th>
</tr>
</thead>
<tbody>
<tr>
<td>MediaMill</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( N_{\text{train}} = 30993 )</td>
<td>( T_{\text{train}} )</td>
<td>276.4s</td>
<td>293.2s</td>
<td>950.4s</td>
<td>23.8s</td>
<td>12.15s</td>
<td>34.1s</td>
</tr>
<tr>
<td>( N_{\text{test}} = 12914 )</td>
<td>( P@1 % )</td>
<td>84.26</td>
<td>84.08</td>
<td>87.37</td>
<td>83.64</td>
<td>84.83</td>
<td>84.42</td>
</tr>
<tr>
<td>( D = 120 )</td>
<td>( P@3 % )</td>
<td>67.34</td>
<td>67.45</td>
<td>72.60</td>
<td>66.13</td>
<td>67.17</td>
<td>67.26</td>
</tr>
<tr>
<td>( K = 101 )</td>
<td>model size</td>
<td>53.06</td>
<td>53.23</td>
<td>58.39</td>
<td>50.90</td>
<td>52.80</td>
<td>52.78</td>
</tr>
<tr>
<td></td>
<td>( T_{\text{test}}/\text{N}_{\text{test}} )</td>
<td>0.27ms</td>
<td>0.37ms</td>
<td>4.95ms</td>
<td>0.004ms</td>
<td>0.142ms</td>
<td>0.078ms</td>
</tr>
<tr>
<td>Bibtex</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( N_{\text{train}} = 4880 )</td>
<td>( T_{\text{train}} )</td>
<td>21.68s</td>
<td>21.47s</td>
<td>296.86s</td>
<td>7.71s</td>
<td>0.203s</td>
<td>0.232s</td>
</tr>
<tr>
<td>( N_{\text{test}} = 2515 )</td>
<td>( P@1 % )</td>
<td>63.66</td>
<td>63.18</td>
<td>64.77</td>
<td>62.36</td>
<td>63.69</td>
<td>63.69</td>
</tr>
<tr>
<td>( D = 1836 )</td>
<td>( P@3 % )</td>
<td>39.42</td>
<td>39.67</td>
<td>38.97</td>
<td>35.60</td>
<td>38.80</td>
<td>39.43</td>
</tr>
<tr>
<td>( K = 159 )</td>
<td>model size</td>
<td>28.60</td>
<td>29.47</td>
<td>28.50</td>
<td>26.50</td>
<td>28.30</td>
<td>28.67</td>
</tr>
<tr>
<td></td>
<td>( T_{\text{test}}/\text{N}_{\text{test}} )</td>
<td>0.64ms</td>
<td>0.73ms</td>
<td>0.70ms</td>
<td>0.007ms</td>
<td>0.028ms</td>
<td>0.094ms</td>
</tr>
<tr>
<td>RCV1-2K</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( N_{\text{train}} = 623847 )</td>
<td>( T_{\text{train}} )</td>
<td>4874.4s</td>
<td>4947.2s</td>
<td>8521.2s</td>
<td>709.5s</td>
<td>641.1s</td>
<td>35.0s</td>
</tr>
<tr>
<td>( N_{\text{test}} = 155962 )</td>
<td>( P@1 % )</td>
<td>91.14</td>
<td>89.79</td>
<td>91.36</td>
<td>90.02</td>
<td>90.52</td>
<td>91.08</td>
</tr>
<tr>
<td>( D = 47236 )</td>
<td>( P@3 % )</td>
<td>73.35</td>
<td>72.65</td>
<td>73.38</td>
<td>71.92</td>
<td>72.31</td>
<td>72.93</td>
</tr>
<tr>
<td>( K = 2456 )</td>
<td>model size</td>
<td>52.69</td>
<td>52.23</td>
<td>52.50</td>
<td>51.23</td>
<td>51.25</td>
<td>52.10</td>
</tr>
<tr>
<td></td>
<td>( T_{\text{test}}/\text{N}_{\text{test}} )</td>
<td>0.87ms</td>
<td>1.08ms</td>
<td>53.95ms</td>
<td>0.066ms</td>
<td>1.72ms</td>
<td>0.338ms</td>
</tr>
<tr>
<td>EURLex-4K</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( N_{\text{train}} = 15339 )</td>
<td>( T_{\text{train}} )</td>
<td>315.9s</td>
<td>324.4s</td>
<td>4543.4s</td>
<td>773.2s</td>
<td>76.07s</td>
<td>9.95s</td>
</tr>
<tr>
<td>( N_{\text{test}} = 3809 )</td>
<td>( P@1 % )</td>
<td>70.86</td>
<td>70.33</td>
<td>79.15</td>
<td>75.90</td>
<td>70.61</td>
<td>74.61</td>
</tr>
<tr>
<td>( D = 5000 )</td>
<td>( P@3 % )</td>
<td>59.06</td>
<td>58.61</td>
<td>64.09</td>
<td>61.16</td>
<td>57.56</td>
<td>59.56</td>
</tr>
<tr>
<td>( K = 3993 )</td>
<td>model size</td>
<td>49.58</td>
<td>49.69</td>
<td>52.09</td>
<td>50.83</td>
<td>47.33</td>
<td>48.43</td>
</tr>
<tr>
<td></td>
<td>( T_{\text{test}}/\text{N}_{\text{test}} )</td>
<td>3.65ms</td>
<td>4.53ms</td>
<td>3.67ms</td>
<td>0.73ms</td>
<td>2.26ms</td>
<td>1.5ms</td>
</tr>
</tbody>
</table>

Table 3: Results and statistics for Multiclass data sets, \( N_{\text{train}} \) = number of training samples, \( N_{\text{test}} \) = number of testing samples, \( T_{\text{train}} \) = training time, \( T_{\text{test}} \) = testing time, \( K \) = number of classes, \( D \) = number of features. DiSMEC and PPDSparse are parallelized with 100 cores. We highlight the best result for each metric, except that for \( T_{\text{train}} \) we highlight best results among single-core solvers (left four) and parallel solvers.

<table>
<thead>
<tr>
<th>Data</th>
<th>Metrics</th>
<th>FastXML</th>
<th>PfastreXML</th>
<th>SLEEC</th>
<th>PDSparse</th>
<th>DiSMEC</th>
<th>PPDSparse</th>
</tr>
</thead>
<tbody>
<tr>
<td>aloi.bin</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( N_{\text{train}} = 100000 )</td>
<td>( T_{\text{train}} )</td>
<td>1900.9s</td>
<td>1901.6s</td>
<td>16193s</td>
<td>139.8s</td>
<td>92.0s</td>
<td>7.05s</td>
</tr>
<tr>
<td>( N_{\text{test}} = 8000 )</td>
<td>accuracy (%)</td>
<td>95.71</td>
<td>93.43</td>
<td>93.74</td>
<td>96.2</td>
<td>96.28</td>
<td>96.38</td>
</tr>
<tr>
<td>( D = 636911 )</td>
<td>model size</td>
<td>1.3G</td>
<td>1.3G</td>
<td>3.7G</td>
<td>19M</td>
<td>16M</td>
<td>14M</td>
</tr>
<tr>
<td>( K = 1000 )</td>
<td>( T_{\text{test}}/\text{N}_{\text{test}} )</td>
<td>5.05ms</td>
<td>5.10ms</td>
<td>28.00ms</td>
<td>0.064ms</td>
<td>0.02ms</td>
<td>0.0178ms</td>
</tr>
<tr>
<td>LSHTC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( N_{\text{train}} = 88806 )</td>
<td>( T_{\text{train}} )</td>
<td>1398.2s</td>
<td>1422.4s</td>
<td>5919.3s</td>
<td>196.8s</td>
<td>298.8s</td>
<td>45.8s</td>
</tr>
<tr>
<td>( N_{\text{test}} = 5000 )</td>
<td>accuracy (%)</td>
<td>22.04</td>
<td>23.32</td>
<td>12.2</td>
<td>22.46</td>
<td>22.74</td>
<td>22.70</td>
</tr>
<tr>
<td>( D = 347255 )</td>
<td>model size</td>
<td>937M</td>
<td>1.1G</td>
<td>631M</td>
<td>88M</td>
<td>142M</td>
<td>381M</td>
</tr>
<tr>
<td>( K = 12294 )</td>
<td>( T_{\text{test}}/\text{N}_{\text{test}} )</td>
<td>5.73ms</td>
<td>8.81ms</td>
<td>14.66ms</td>
<td>0.40ms</td>
<td>3.7ms</td>
<td>6.94ms</td>
</tr>
<tr>
<td>Dmoz</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( N_{\text{train}} = 345068 )</td>
<td>( T_{\text{train}} )</td>
<td>6475.1s</td>
<td>6619.7s</td>
<td>47490s</td>
<td>2518.9s</td>
<td>1972.0s</td>
<td>170.60s</td>
</tr>
<tr>
<td>( N_{\text{test}} = 385340 )</td>
<td>accuracy (%)</td>
<td>40.76</td>
<td>39.78</td>
<td>33.03</td>
<td>39.91</td>
<td>39.38</td>
<td>39.32</td>
</tr>
<tr>
<td>( D = 833484 )</td>
<td>model size</td>
<td>3.5G</td>
<td>3.8G</td>
<td>1.5G</td>
<td>680M</td>
<td>369M</td>
<td>790M</td>
</tr>
<tr>
<td>( K = 11947 )</td>
<td>( T_{\text{test}}/\text{N}_{\text{test}} )</td>
<td>3.29ms</td>
<td>3.20ms</td>
<td>40.43ms</td>
<td>1.87ms</td>
<td>4.38ms</td>
<td>6.58ms</td>
</tr>
</tbody>
</table>

(Table 1), we use tf-idf features with sample-wise normalization as suggested by the author of [1].

Our experimental results confirmed several comments from previous work [1, 34]: 1) Among single-core solvers (PDSparse, FastXML, PfastreXML, SLEEC), PDSparse can achieve orders of
magnitude speed up in terms of training time without significantly downgrading performance compared to the direct one-vs-all approach (except on Delicious-200k, a data set of missing labels).

2) Given enough computing resources, DiSMEC is able to achieve significantly higher accuracy than other approaches on some data sets. However, their training on the largest data sets typically take few days even with 100 cores.

From Table 1-3, we illustrate our proposed PPD-Sparse method combines the strength from both PDS-Sparse and DiSMEC. By adopting one-versus-all loss, PPD-Sparse can achieve accuracy as good as DiSMEC on most of data sets and resolve drawbacks of the max-margin loss used by PDS-Sparse in three ways: (i) PPD-Sparse reduces the memory requirement of PDS-Sparse by orders of magnitude due to the separation of training of each class, which clears the MLE issue of PDS-Sparse on Amazon-670K, (ii) By embarrassingly parallelized to 100 cores, PPD-Sparse is orders of magnitude faster than both PDS-Sparse and parallel 1-vs-all (DiSMEC), (iii) the performance of PPD-Sparse is less sensitive to data set of mislabeling. On Delicious-200k, a data set of missing positive labels, PPD-Sparse improves accuracy of PDS-Sparse significantly.

The training of PPD-Sparse is consistently faster than tree-based and local embedding methods by orders of magnitude while maintaining a competitive accuracy on most of data sets. On Amazon-670K and WikiLSHTC-325K, PPD-Sparse (and DiSMEC) enjoy a significant increase in accuracy compared to tree-based approaches. On the other hand, the prediction speed of Primal Dual sparse approaches (PDS-Sparse, PPD-Sparse) are slower than tree-based methods (FastXML, PFastreXML) on problems of more than 104 classes, while being comparable on medium-sized data sets of 103 – 104 classes (Table 3), presumably because tree-based methods enjoy logarithmic-time prediction w.r.t. the number of classes.

7 ACKNOWLEDGEMENT


