Downloaded 11/23/20 to 27.7.29.164. Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/page/terms

Fast Training for Large-Scale One-versus-All Linear Classifiers using Tree-Structured Initialization

Huang Fang* Minhao Cheng† Cho-Jui Hsieh‡ Michael Friedlander §

Abstract

We consider the problem of training one-versus-all (OVA) linear classifiers for multiclass or multilabel classification when the number of labels is large. A naive extension of OVA to this problem, even with hundreds of cores, usually requires hours for training on large real world datasets. We propose a novel algorithm called OVA-Primal++ that speeds up the training of OVA by using a tree-structured training order, where each classifier is trained using its parent's classifier as initialization. OVA-Primal++ is both theoretically and empirically faster than the naive OVA algorithm, and yet still enjoys the same highly parallelizability and small memory footprint. Extensive experiments on multiclass and multilabel classification datasets validate the effectiveness of our method.

Keywords: One-versus-All Classification

1 Introduction

Classification with multiple classes or labels is a fundamental problem in machine learning. With the emergence of big data, recent applications such as image recognition or text tagging commonly involve milticlass and multilabel classification with huge amount of features, labels, and data points. These are known as extreme classification (XMC) problems.

One-versus-all (OVA) is a simple and effective approach for multiclass and multilabel classification. It divides the whole problem into K binary classification subproblems, where K is the number of labels, and tries to learn K corresponding classifiers one by one. By its nature, OVA enjoys two important features that make it popular in practice: small memory usage and high parallelizability.

OVA with linear classifiers plays an important role in both academia and industry. Some important features include its ability to produce sparse and effective solutions for high-dimensional text classification by pruning out small weights after training [1], and its ability to achieve state-of-the-art accuracy for image classification by fine tuning the last layer of a pretrained convolutional network [2, 3]. However, these applications usually involve hundreds of thousands of labels and a naive implementation of OVA, even with hundreds of computational cores, would require hours for training [4].

Recent approaches [5, 6] have been proposed to resolve the expensive training issue. However, these proposals involve changes to the objective function of the linear classification that can degrade prediction accuracy. Moreover, both approaches are designed to handle high-dimensional sparse data, and as we demonstrate empirically, they usually works poorly with low-dimensional dense data; see §4.

Our proposed algorithm, called OVA-Primal++, works for large-scale multiclass and multilabel classification, and is effective for both high-dimensional sparse data and low-dimensional dense data. It also avoids modifcations to the classification objective function, and thus avoids the disadvantages of previous approaches. As with naive OVA, OVA-Primal++ is highly parallelizable and has a small memory footprint. Under some mild assumptions, we formally prove that the proposed algorithm can reduce the upper bound of runtime compared with the naive OVA approach. When a subproblem solver with a sublinear convergence rate is used, our algorithm can reduce the runtime upper-bound by a factor of K, where K is number of labels. When the subproblem solver has a linear-convergence rate, we can reduce the upper bound by a factor of $\Theta(\log N)$, where N is the number of samples. Experiments on large-scale real-world datasets validate the effectiveness of our approach.

2 Related Work

For classification that involves a large number of labels, existing approaches can roughly be divided into three categories: tree-based, embedding-based, and OVA-based approaches.

2.1 Tree-based approaches FastXML [7] was the first tree-based model for XMC problems. It partitions the feature space in a tree structure and can achieve logarithmic prediction time and high test accuracy. PfastreXML [8] is an improved version of FastXML and

*University of British Columbia, hgfang@cs.ubc.ca
†University of California, Los Angeles, mhcheng@ucla.edu
‡University of California, Los Angeles, chohsieh@cs.ucla.edu
§University of British Columbia, mpf@cs.ubc.ca

Copyright © 2019 by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 11/23/20 to 27.7.29.164. Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/page/terms

can yield better prediction for rare labels. A more recent approach called Prabel [6] tries to partition the label space into a balanced binary tree, and it empirically demonstrates fast training times for high-dimensional sparse data. Note that all of the above tree-structure based approaches require more memory usage than the training of OVA linear classifiers.

2.2 Embedding-based approaches To reduce the model size and training time, some approaches assume that the optimal classifiers have a *low-rank* structure [9, 10, 11]. Although this assumption can reduce the training and prediction time, the prediction accuracy is usually inferior to tree-based or OVA-based approaches. To yield better prediction, the local embedding approach SLEEC [12] has higher prediction accuracy than previous embedding-based approaches, but it is expensive to training and its performance can be unstable on some data sets [5].

2.3 OVA-based approaches A naive extension of OVA strategy to large-scale multiclass or multilabel classificaiton is expensive to train and could produce a very large model for high dimensional data. PDSparse [4] is a modified OVA approach that could learn very sparse model with *separation ranking loss* and L1 regularization. Although PDSparse can learn a small model, its training is not memory efficient and could require hundreds of gigabytes memory on some large datasets [5]. Moreover, its test accuracy is usually inferior to the naive OVA method. DiSMEC [1] was the first attempt to scale classical OVA to XMC by using hundreds of cores for training. After training, DiSMEC proposed to prune out small model weights by hard thresholding. This post-processing leads to small model size, and experiments on real world datasets demonstrated that traditional OVA (with pruning) can actually achieve state-of-the-art accuracy and competitive prediction time [1, 5]. A recent work PPDSparse [5] studied the sparse inducing property of OVA and proposed a primal-dual active set approach to speed up the training of traditional OVA. Although PPDSparse is empirically faster than DiSMEC, it is less flexible, allowing only *elastic-net* regularization and works well only with high dimensional sparse data.

3 Proposed Method

In this section we propose a novel approach to speedup the training of OVA.

3.1 Problem Setup We consider the multilabel classification problem with N training samples $\{\boldsymbol{x}_i\}_{i\in[N]}$, each sample $\boldsymbol{x}_i \in \mathbb{R}^D$ has D features. We use K to denote the number of labels and $\boldsymbol{y}_k \in \{-1,+1\}^N$ as the binary coding for label k, where $\boldsymbol{y}_{k,i} = +1$ if the k-th label belongs to $\boldsymbol{x}_i$ and $\boldsymbol{y}_{k,i} = -1$ otherwise. In this way, the labels for sample $\boldsymbol{x}_i$ can be expressed as $\mathcal{L}_i = \{k \mid \boldsymbol{y}_{k,i} = +1\}$. The average number of labels per sample is defined as $\overline{L} = \frac{\sum|\mathcal{L}_i|}{N}$. For extreme classification, both N and K are large, but $\overline{L}$ is comparatively small and does not scale with K or N.

For one-versus-all (OVA) linear classification, the original multilabel problem is divided into K binary classification subproblems where the k-th subproblem learns a classifier for the k-th label. The optimization problem for the k-th label is given by:

$$\min_{\boldsymbol{w}_k} f(\boldsymbol{y}_k, \boldsymbol{w}_k) := \mathcal{R}(\boldsymbol{w}_k) + C\sum_{i=1}^{N} l(\boldsymbol{y}_{k,i}\boldsymbol{w}_k^T\boldsymbol{x}_i) \tag{3.1}$$

where $\boldsymbol{w}_k$ is the classifier for the k-th label. $l(\cdot)$ is a loss function (e.g. logistic loss, L1-hinge loss or L2-hinge loss), $\mathcal{R}$ is the regularization and C is the balancing parameter that controls the trade-off between empirical risk and regularization.

Note that we write $f(\cdot)$ as a function not only with variable $\boldsymbol{w}$ but also with variable $\boldsymbol{y}$; This notation will help our analysis and its use will be clear in the following sections. We assume the problem has the following properties.

Assumptions:

- Each sample is only involved with a small number of true labels. $\forall i \in [N],\ |\mathcal{L}_i| \ll K$.
- For the simplicity of analysis, we assume that $\|\boldsymbol{x}_i\|_2 \leq 1$ for $i = 1, 2, ..., N$.
- The average number of labels per sample $\overline{L}$ and the average number of samples per label $\frac{N\overline{L}}{K}$ does not scale with N or K, which is usually true for extreme classification datasets.
- Each subproblem (3.1) is solved by an iterative solver $\mathcal{A}$.

3.2 Key Observations Let $\boldsymbol{w}_k^*$ be the minimizer of $f(\boldsymbol{y}_k, \boldsymbol{w}_k)$ with respect to $\boldsymbol{w}_k$. Also define $l_H(\boldsymbol{y}_p, \boldsymbol{y}_q) = \sum_{i=1}^{n} \mathbf{1}\{\boldsymbol{y}_{p,i} \neq \boldsymbol{y}_{q,i}\}$ as the Hamming loss between $\boldsymbol{y}_p$ and $\boldsymbol{y}_q$. For label p and label q, if $l_H(\boldsymbol{y}_p, \boldsymbol{y}_q) = 0$, then obviously $\boldsymbol{w}_p^* = \boldsymbol{w}_q^*$. A question naturally arises: when $l_H(\boldsymbol{y}_p, \boldsymbol{y}_q)$ is small, will the iterative solver $\mathcal{A}$ converges to $\boldsymbol{w}_q^*$ in fewer iterations if it is initialized by $\boldsymbol{w}_p^*$ instead of the naive zero initialization? If this is true, when we have a large number of subproblems, can we exploit the above property to gain significant speedup for training OVA classifiers? We will formally describe the property and show how we use it to speed up OVA for extreme classification in this section.

Copyright © 2019 by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 11/23/20 to 27.7.29.164. Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/page/terms

LEMMA 3.1. *Assume $l(\cdot)$ is α-Lipschitz and $\|\boldsymbol{w}_k^*\|_2 \leq B\ \forall k \in [K]$. Let $p, q \in [K]$ and $\boldsymbol{y}_p, \boldsymbol{y}_q \in \{-1, +1\}^N$. Then*

$$f(\boldsymbol{y}_q, \boldsymbol{w}_p^*) - f(\boldsymbol{y}_q, \boldsymbol{w}_q^*) \leq 4 l_H(\boldsymbol{y}_p, \boldsymbol{y}_q) C \alpha B. \tag{3.2}$$

Please see supplementary material for the proof. Lemma 3.1 shows that if $l_H(\boldsymbol{y}_p, \boldsymbol{y}_q)$ is small, then the initial objective value will be close to the optimal objective value if we solve subproblem q with initialization $\boldsymbol{w}_p^*$. However, if we use zeros as initialization, the best upper bound for $f(\boldsymbol{y}_q, \boldsymbol{0}) - f(\boldsymbol{y}_q, \boldsymbol{w}_q^*)$ is $CNl(0)$. Note that in Lemma 3.1 we do not make any assumptions on our regularizer $\mathcal{R}$, which means our analysis and proposed method are flexible and could be applied to any kind of regularization.

This result motivates us to construct a "fake" subproblem with label $\boldsymbol{y}_0$. Assume the solution of this fake subproblem is $\boldsymbol{w}_0^*$, we then use $\boldsymbol{w}_0^*$ as the initialization for all the subproblems. Of course, forcing all subproblems to use the same initialization may not be optimal, and we will describe a more general method in section 3.4. Now we study how to select the best $\boldsymbol{y}_0$. To minimize the upper bound of the overall initial objective gap $\sum_{k=1}^K |f(\boldsymbol{y_k}, \boldsymbol{w}_0^*) - f(\boldsymbol{y}_k, \boldsymbol{w}_k^*)|$, based on Lemma 3.1 we need to minimize$\sum_{k=1}^K l_H(\boldsymbol{y}_0, \boldsymbol{y}_k)$. It is easy to show that the optimal solution is

$$\boldsymbol{y}_{0,i} = \text{sign}(\sum_{k=1}^K \boldsymbol{y}_{i,k}), \tag{3.3}$$

where $\text{sign}(z) = 2 \times \mathbf{1}\{z \geq 0\} - 1$.

In the setting of extreme classification, usually we have $|\mathcal{L}_i| \ll K$, thus almost always $\boldsymbol{y}_0 = -\mathbf{1}_n$ which is a vector with all -1 elements. Based on lemma 3.1, we develop the following theorems to show that we could significantly reduce runtime upper bound with the improved initialization.

THEOREM 3.1. *Assume that $l(\cdot)$ is α-Lipschitz and we have a solver $\mathcal{A}$ with sublinear convergence rate that can solve each subproblem k with ϵ precision in $T_k = O\left(\frac{f(\boldsymbol{y}_k, \boldsymbol{w}_0) - f(\boldsymbol{y}_k, \boldsymbol{w}_k^*)}{\epsilon^p}\right)$ iterations. With $\boldsymbol{w}_0 = \boldsymbol{w}_0^*$ (using the solution of the fake subproblem to initilalize), we can bound the total number of iterations of OVA by*

$$T_{total} = \sum_{k=1}^K T_k = O\left(\frac{\sum_{k=1}^K l_H(\boldsymbol{y}_k, \boldsymbol{y}_0)}{\epsilon^p}\right) = O\left(\frac{N\overline{L}}{\epsilon^p}\right).$$

REMARK 1. *Recall that the best upper bound for $f(\boldsymbol{y}_k, \boldsymbol{0}) - f(\boldsymbol{y}_k, \boldsymbol{w}_k^*)$ is $O(N)$. Therefore under a naive zeros initialization $\boldsymbol{w}_0 = \boldsymbol{0}$, using a solver $\mathcal{A}$ described in theorem 3.1, the best bound we can achieve is*

$$T_{total} = \sum_{k=1}^K T_k = O\left(\frac{NK}{\epsilon^p}\right).$$

Compared with the result in Theorem 3.1, we can reduce the upper bound of run time by a factor of $\frac{K}{\overline{L}}$.

For extreme classification problem, $\frac{K}{\overline{L}}$ is usually quite large and Theorem 3.1 implies that our algorithm can significantly reduce the upper bound of runtime when using a solver with sublinear convergence rate such as *stochastic gradient descent* [13].

THEOREM 3.2. *Assume that $l(\cdot)$ is α-Lipschitz and we have a solver $\mathcal{A}$ that can solve each subproblem k with ϵ precision in $T_k = O\left(\log\left(\frac{f(\boldsymbol{y}_k, \boldsymbol{w}_0) - f(\boldsymbol{y}_k, \boldsymbol{w}_k^*)}{\epsilon}\right)\right)$ iterations. With $\boldsymbol{w}_0 = \boldsymbol{w}_0^*$, we can bound the total number of iterations of OVA by*

$$\begin{aligned} T_{total} = \sum_{k=1}^K T_k &= O\left(K \log\left(\frac{\sum_{k=1}^K l_H(\boldsymbol{y}_k, \boldsymbol{y}_0)}{K\epsilon}\right)\right) \\ &= O\left(K \log\left(\frac{N\overline{L}}{K\epsilon}\right)\right). \end{aligned}$$

REMARK 2. *Recall that the best upper bound for $f(\boldsymbol{y}_k, \boldsymbol{0}) - f(\boldsymbol{y}_k, \boldsymbol{w}_k^*)$ is $O(N)$, Therefore under a naive zeros initialization $\boldsymbol{w}_0 = \boldsymbol{0}$, using a solver $\mathcal{A}$ described in theorem 3.2, the best bound we can achieve is*

$$T_{total} = \sum_{k=1}^K T_k = O\left(K \log\left(\frac{N}{\epsilon}\right)\right).$$

Compared with the result in Theorem 3.2, we can reduce the upper bound of runtime by a factor of $\log N$ asymptotically if the average number of samples per label $\overline{N} = \frac{N\overline{L}}{K}$ does not scale with N or K.

When we have a solver with linear convergence rate, such as *gradient descent* for strongly convex functions, Theorem 3.2 implies that our algorithm can reduce a factor of $\Theta(\log N)$ for the upper bound of runtime.

3.3 OVA-Primal: A Fast Primal OVA Solver

Motivated by Theorem 3.1 and Theorem 3.2, we propose a novel algorithm called OVA-Primal that speeds up the training of OVA classification. The detailed algorithm is shown in Algorithm 1. Compared with the naive

Copyright © 2019 by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 11/23/20 to 27.7.29.164. Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/page/terms

Algorithm 1 OVA-Primal

Input : $\{\boldsymbol{x}_i\}_{i\in[N]}, \{\boldsymbol{y}_k\}_{k\in[K]}, C, \mathcal{A}$
Construct the "fake" subproblem (label defined in (3.3))
Solve the "fake" subproblem by $\mathcal{A}$ with $\mathbf{0}$ initialization and get $\boldsymbol{w}_0^*$
for $k = 1$ to K **do** ◁ parallelizable
  Initialize $\boldsymbol{w}_0 = \boldsymbol{w}_0^*$
  Solve the k-th subproblem by $\mathcal{A}$ and get $\boldsymbol{w}_k^*$
  Optional: Compress and store $\boldsymbol{w}_k^*$ ◁ for high dimensional data
end

OVA classification, OVA-Primal only needs to solve one extra subproblem. It is faster than DiSMEC (naive OVA training) while enjoying the same advantage such as high parallelizability and small model size.

3.4 OVA-Primal++: Further Speeding up OVA-Primal with Tree-Structured Initialization In this part, we show that OVA-Primal can be further accelerated by using a tree structured ordering to train all subproblems and propose a corresponding algorithm called OVA-Primal++.

Imagine that we have the example shown in Figure 1. When we use OVA-Primal, we have $l_H(\boldsymbol{y}_0, \boldsymbol{y}_1) + l_H(\boldsymbol{y}_0, \boldsymbol{y}_2) = 3$, but this can be improved if we use $\boldsymbol{w}_0^*$ as the initialization to calculate $\boldsymbol{w}_1^*$ and then use $\boldsymbol{w}_1^*$ as the initialization to get $\boldsymbol{w}_2^*$. In this case, we will have $l_H(\boldsymbol{y}_0, \boldsymbol{y}_1) + l_H(\boldsymbol{y}_1, \boldsymbol{y}_2) = 2$. This example motivates us to use the classifier of the "nearest" label as the initialization to calucate current classifier instead of just initializing $\boldsymbol{w}_0^*$.

"fake" label $\boldsymbol{y}_0$	label 1 $\boldsymbol{y}_1$	label 2 $\boldsymbol{y}_2$	...
−1	+1	+1	
−1	−1	+1	
−1	−1	−1	...
−1	−1	−1	
−1	−1	−1	

Figure 1: Example

Formally speaking, after we add the fake subproblem, there are in total $K + 1$ subproblems, and we want to find the "ideal ordering" π to train these subproblems such that the total Hamming loss is minimized.

Interestingly, finding the best ordering of training these subproblems can be viewed as a graph problem, where we have in total $K + 1$ vertices $\{\boldsymbol{v}_k\}_{k\in[K+1]}$, and each $\boldsymbol{v}_k$ represents the k-th subproblem. The weight associated with edge $\boldsymbol{e}_{p,q}$ is set to be $l_H(\boldsymbol{y}_p, \boldsymbol{y}_q)$. Every possible ordering π corresponds to a tree on the graph. We start from the root and solve subproblem q with classifier $\boldsymbol{w}_p^*$ as initialization if $\boldsymbol{v}_p$ is the parent of $\boldsymbol{v}_q$. It is easy to see that the "ideal ordering" is the tree with minimum total weights and this is actually an *minimum spanning tree* problem.

The detailed algorithm of OVA-Primal++ is shown in Algorithm 2.

Recall that in Theorem 3.1 and 3.2, the upper bound of the runtime is controlled by the term $\sum_{k=1}^{K} l_H(\boldsymbol{y}_k, \boldsymbol{y}_0)$. By using this tree–structured training, we can guarantee that $\sum_{k=1}^{K} l_H(\boldsymbol{y}_k, \boldsymbol{y}_{\Pi(k)}) \leq \sum_{k=1}^{K} l_H(\boldsymbol{y}_k, \boldsymbol{y}_0)$, where $\Pi(k)$ denotes the parent of $\boldsymbol{v}_k$; thus the upper bound on the runtime of OVA-Primal++ is at least as good as OVA-Primal, and usually observed to be a lot better than OVA-Primal.

3.5 OVA-MST How to solve the *minimum spanning tree* problem is non-trivial because our graph is highly dense. In this section, we propose an efficient algorithm called OVA-MST to solve our minimum spanning tree problem by only considering *necessary* edges.

LEMMA 3.2. *For a weighted undirected graph $G(V, E)$, denote the cost of its minimum spanning tree as $c(G)$. If there are 3 vertices $\boldsymbol{v}_p, \boldsymbol{v}_q, \boldsymbol{v}_k$ such that $w_{p,k}+w_{q,k} = w_{p,q}$, then the cost of minimum spanning tree remains the same if we remove $\boldsymbol{e}_{p,q}$ from G.*

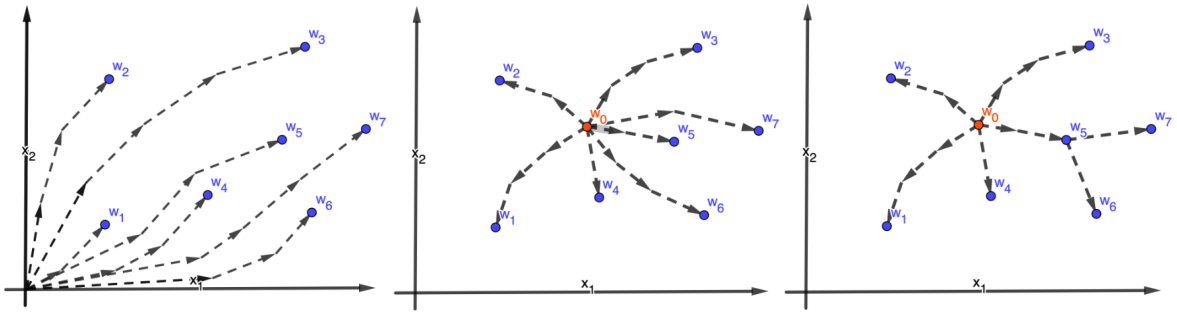
Lemma 3.2 implies that if there are no samples that have both label p and q, then we can exclude the edge $\boldsymbol{e}_{p,q}$ from G, since $l_H(\boldsymbol{y}_0, \boldsymbol{y}_p) + l_H(\boldsymbol{y}_0, \boldsymbol{y}_q) = l_H(\boldsymbol{y}_p, \boldsymbol{y}_q)$. With this observation, we develop the following theorem to bound the number of necessary edges.

THEOREM 3.3. *Denote G as the graph described in section 3.4, where each vertex of G stands for a subproblem and $w_{p,q} = l_H(\boldsymbol{y}_p, \boldsymbol{y}_q)$ for $0 \leq p \neq q \leq K$. The number of edges we need to consider is $O(K + \sum_{i=1}^{N} |\mathcal{L}_i|^2)$.*

By theorem 3.3, we bound the number of edges in $O(K + \sum_{i=1}^{N} |\mathcal{L}_i|^2)$, so if we run Kruskal's algorithm [14] to find the minimum spanning tree, we are guaranteed to find the MST in $O\left(\left(K + \sum_{i=1}^{N} |\mathcal{L}_i|^2\right) \log\left(K + \sum_{i=1}^{N} |\mathcal{L}_i|^2\right)\right)$ time. Due to the space limitation, we place the detailed graph formulation and OVA-MST algorithm in the supplementary materials. To summarize the result, the space complexity of OVA-MST is $O(K + \sum_{i=1}^{N} |\mathcal{L}_i|^2)$ and the time complexity is $O\left(\left(K + \sum_{i=1}^{N} |\mathcal{L}_i|^2\right) \log\left(K + \sum_{i=1}^{N} |\mathcal{L}_i|^2\right)\right)$, which is near linear in N and K if $\overline{L}$ does not scale with N or K. We use Kruskal's algorithm to solve the MST problem in this work, but it is also worth to note that there are some other clever algorithms [15] that we can use instead of Kruskal's algorithm, by which we are potentially able to solve the MST even faster by employing clever parallelization.

Copyright © 2019 by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 11/23/20 to 27.7.29.164. Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/page/terms



(a) Naive OVA (b) OVA-Primal (c) OVA-Primal++

Figure 2: Illustrative geometric interpretation of naive OVA, OVA-Primal and OVA-Primal++, please zoom.

Datasets	NUS-WIDE	EURLex	AmazonCat13k	AmazonCat14k	Wiki31k
# subtrees	889	2072	9510	10875	9816
Height of $\mathcal{T}$	4	5	8	8	10

Table 1: Statistics of minimum spanning trees

Algorithm 2 OVA-Primal++

Input : $\{\boldsymbol{x}_i\}_{i\in[N]}, \{\boldsymbol{y}_k\}_{k\in[K]}, C, \mathcal{A}$
Construct the "fake" subproblem (label defined in (3.3)), denote it as label 0.
Call OVA-MST and contruct the minimum spanning tree $\mathcal{T}$ that root at label 0.
Solve the "fake" subproblem by $\mathcal{A}$ with $\mathbf{0}$ as initialization and get $\boldsymbol{w}_0^*$
for each subtree connect with $root(\mathcal{T})$ **do** ◁ parallelizable
    Traverse subtree via depth-first search
    At each node, solve its corresponding subproblem by $\mathcal{A}$, initialized by its parent's classifier.
    When all children of a node are solved, optionally compress its classifier.
end

3.6 Parallelization for OVA-Primal++ In this section, we propose a method called *blocked depth-first search* that can parallelize OVA-Primal++ with small memory footprint.

Breath-First Search: We denote our minimum spanning tree that rooted at label 0 as $\mathcal{T}$. Note that each subproblem uses its parent's classifier as the initialization, so the children of the same parent can be solved "independently". This suggests using *breadth-first search* (BFS) for tree traversal, since it is easily parallelizable. However, using BFS requires us to store all the internal nodes' classifiers as the initialization for their children and we cannot apply DiSMEC's prunning heuristics(model compression) for high dimensional data after solving each subproblem, which makes the space complexity to be $O(KD)$ and hence memory inefficient.

Depth-First Search: *Depth-first search* (DFS) could resolve the memory issue. We can compress a parent's classifer as long as all of its children are visited, so its space complexity is $O(hD + M)$, where h is the height of $\mathcal{T}$ and M is the model size after compression. However, DFS is hard to parallelize because each child needs its parent's classifier as its initialization, which makes naive DFS inappropriate for our problem.

Blocked Depth-First Search: For most large-scale multilabel classification datasets, we observe that for many labels, their "nearest" vertex is actually label 0 (the fake problem). This implies that our minimum spanning tree $\mathcal{T}$ is usually "short" and "wide". The statistics of $\mathcal{T}$ for different datasets are shown in Table 1. Based on this observation, we propose a method called *blocked depth-first search* to parallelize OVA-Primal++. Basically, we first solve the root's problem, then we divide the "short" and "wide" $\mathcal{T}$ into many subtrees whose roots are connected with the root of $\mathcal{T}$. For extreme classification datasets, there are usually a large number of subtrees and each subtree is small. For the subproblems on each subtree, we apply DFS and use different threads to solve different subtrees in parallel. The space complexity is $O(hD\tau + M)$, where τ is the number of threads.

3.7 Geometric Interpretation The geometric interpretation of naive OVA, OVA-Primal and OVA-Primal++ is shown in Figrue 2. Each arrow in the figure stands for one iteration of the solver. Note that

Copyright © 2019 by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 11/23/20 to 27.7.29.164. Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/page/terms

Datasets	tiny-imagenet	Aloi	NUS-WIDE	EURLex	AmazonCat13k	AmazonCat14k	Wiki31k
N_{train}	100000	97200	161789	15539	1186239	4398050	14146
N_{test}	10000	10800	107859	3809	306782	1099725	6616
D	512	128	128	5000	203882	597540	101938
K	200	1000	1000	3993	13330	14588	30938
$\overline{L}$	1	1	5.78	5.31	5.04	3.53	18.64
$\overline{N}$	500	97.2	935.22	25.73	448.57	1330.1	8.52
Density	1.0	0.2476	0.9925	0.0476	3.54×10^{-4}	8.13×10^{-5}	0.00661

Table 2: Data Statistics, where $\overline{L}$ stands for the average number of labels per sample and $\overline{N}$ denotes the average number of samples per label.

this is just an illustrative example to visualize the main idea behind OVA-Primal and OVA-Primal++.

4 Experiments

In this section, we compare our algorithm with other state-of-the-art solvers for multilabel classification problems in terms of training time and prediction accuracy. We include the following methods in our comparison:

- FastXML [7]: an efficient tree-based approach that perform partition on feature space.
- Parabel [6]: a very recently proposed tree-based approach that empirically works well with high dimensional sparse data.
- SLEEC [12]: the representative embedding-based approach that learns a sparse local embedding and is guaranteed to produce a small model.
- DiSMEC [1]: a highly parallelizable OVA model with L2-hinge loss and L2 regularization. It uses LIBLINEAR's [16] trust region Newton's method [17] to solve each subproblem and learn very sparse parameters by pruning out small weights after training.
- PPDSparse [5]: a recent method that uses a primal-dual method to solve OVA with L2-hinge loss and elastic-net regularization. PPDSparse also aims at accelerating the training of the OVA model, but it focus on solving the dual of each subproblem.
- OVA-Primal++: our proposed method using L2-hinge loss and L2 regularization as suggested by [1]. We solve each subproblem by LIBLINEAR's truncated Newton's method with backtracking line-search [18]. It is worth noting that the truncated Newton's method used in OVA-Primal++ has linear convergence rate in terms of the number of inner conjugate gradient (CG) iterations [17, 19] for strongly convex functions. Our code is public available at `https://github.com/fanghgit/XMC`.
- OVA-Naive: the baseline method. It has the same objective function as OVA-Primal++ and uses the same solver for each subproblem but with the naive zeros initialization.

For FastXML, Parabel, SLEEC, DiSMEC and PPDSparse, we use their online available code from the Extreme Classification Repository[1]. The statistics of the datasets are shown in Table 2, tiny-imagenet is a subset of the imageNet dataset[20] where all images are downsampled into 64×64 ones, we use the 512-dimensional representation extracted by a pretrained ResNet18[21] and here our task is fast fine-tuning the last layer. For NUS-WIDE, we use its 128 dimensional cVLAD+ features that downloadable from Mulan [2]. Aloi is downloaded from LIBSVM's website [3]. EURLex, AmazonCat13k, AmazonCat14k and Wiki31k are standard high-dimensinal text classification datasets that available from the Extreme Classification Repository.

Implementation Details

Following the setting in DiSMEC [1], we use the default train/test split for each dataset and apply a tf-idf transformation if it is text data. As suggested by [1, 5], we use L2-hinge loss, L2-regularization and set $C = 1$ for all experiments. For SLEEC, FastXML and PPDSparse, we use their suggested setting of hyper-parameters. All experiments are conducted on a server with 16 Intel Xeon E5-2640 2.40GHz CPUs and and 64GB RAM. For more implementation details, see the supplementary materials.

Evaluation Metrics

We evaluate different approaches by their training time T_{train} and prediction accuracy. The prediction accuracy is measured by `Precision@k` (denoted by `P@k`) defined as:

[1] `http://manikvarma.org/downloads/XC/XMLRepository.html`
[2] `http://mulan.sourceforge.net/datasets-mlc.html`
[3] `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`

Copyright © 2019 by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 11/23/20 to 27.7.29.164. Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/page/terms

Data & Metrics	1 core	Tree-based approaches, 10 cores		OVA-based approaches, 10 cores			
	SLEEC	**FastXML**	**Parabel**	**PPDSparse**	**DiSMEC**	**OVA-Naive**	**OVA-Primal++**
tiny-imagenet							
T_{train}	7043.2s	1104.05s	269.68s	122.2s	108.81s	101.81s	**65.21s**
P@1	39.78	41.29	**48.29**	27.88	46.81	47.32	47.56
P@3	19.33	18.78	22.39	15.10	21.59	22.46	**22.51**
P@5	12.94	12.54	14.83	10.79	14.33	14.99	**15.01**
Aloi							
T_{train}	921.3s	94.89s	16.32s	**11.1s**	49.75s	45.7s	25.09s
P@1	86.24	**88.63**	81.57	77.15	86.13	86.03	86.09
P@3	**31.87**	31.56	29.90	29.51	30.91	30.86	30.87
P@5	**19.51**	19.27	18.48	18.34	18.89	18.89	18.87
NUS-WIDE							
T_{train}	7923s	352.79s	970.72s	309.43s	329s	254.73	**115.17s**
P@1	16.24	17.13	16.94	16.22	17.13	**17.15**	**17.15**
P@3	12.80	13.47	13.44	12.71	13.44	**13.45**	**13.45**
P@5	10.89	**11.51**	**11.51**	10.82	11.41	11.41	11.41
EURLex							
T_{train}	310.35s	34.2s	**26.26s**	45.5s	64.05s	64.97s	**33.96s**
P@1	72.70	71.07	82.25	81.99	**82.83**	82.49	82.57
P@3	55.93	59.31	68.71	69.06	**69.89**	69.83	69.79
P@5	45.44	50.39	57.53	57.86	58.44	**58.66**	58.53
Amazon13k							
T_{train}	60353s	3396.93s	**2722s**	8370s	13536s	13286	**7330s**
P@1	89.20	93.08	93.02	93.04	**93.79**	93.75	93.75
P@3	75.18	78.16	79.14	78.09	**78.89**	**78.89**	**78.89**
P@5	61.09	63.35	64.51	62.50	63.66	**63.76**	63.66
Amazon14k							
T_{train}	147220s	**6224s**	8145s	28500s	37118s	38001s	**17796s**
P@1	88.84	88.45	89.27	88.28	89.91	**89.95**	**89.95**
P@3	69.07	68.01	69.36	66.04	68.99	69.37	**69.38**
P@5	**54.51**	53.11	54.44	49.71	53.77	54.42	54.32
Wiki31k							
T_{train}	2964.8s	**128.14s**	334.79s	1560s	2550s	2434s	**1364s**
P@1	84.28	82.91	84.19	**85.38**	84.17	84.16	84.17
P@3	72.05	67.89	72.46	**74.88**	74.72	74.70	74.73
P@5	61.80	57.78	63.37	65.54	**65.94**	65.92	65.92

Table 3: Experimental result for all methods. Note that DiSMEC, OVA-Naive and OVA-Primal++ have the same objective function and use the same stopping criterion; thus their prediction accuracy are very similar and their training time is comparable. Due to limited resources, we use 10 cores to parallelize all methods except SLEEC because their code does not support parallelization. Note that we highlight the best training time of OVA-based algorithms and non OVA-based algorithm separately since the focus of this paper is to speed up the training of OVA linear classifiers instead of developing a new model for multilabel classification.

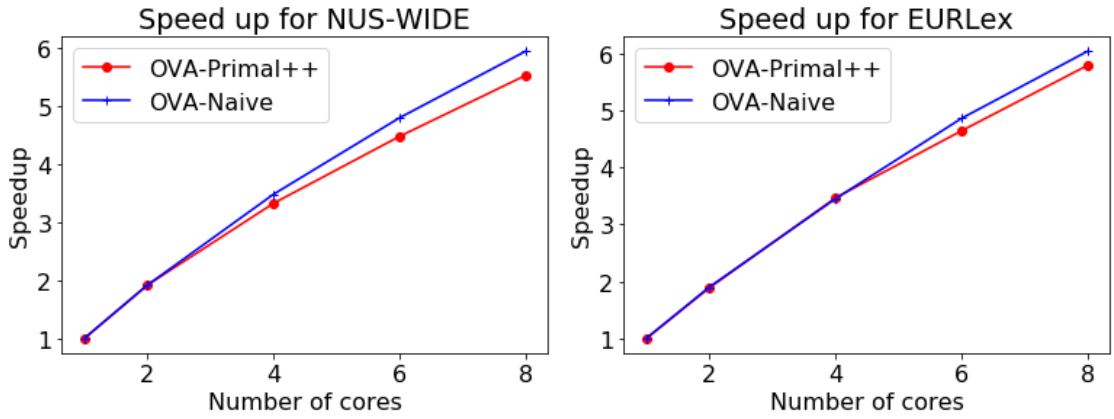
$$\texttt{P@k} := \frac{1}{k} \sum_{l \in \text{rank}_k(\hat{\boldsymbol{y}})} \boldsymbol{y}_l,$$

where $\boldsymbol{y} \in \{0,1\}^K$ is the true label vector and $\hat{\boldsymbol{y}} \in \mathbb{R}^K$ is our predicted score vector for all labels. $\text{rank}_k(\cdot)$ represent the k labels that correspond to the k highest scores.

4.1 Experimental Results on Training Time and Test Accuracy The result of training time and prediction accuracy are presented in Table 3, note that the prediction accuracy for tiny-imagenet is much less than the stat-of-the-art accuracy for imageNet since the figures in tiny-imagenet are resized to 64×64 pixels instead of 224×224 pixels in the original imageNet dataset. As shown in Table 3, our results validate the arguments in [1, 5] that OVA method is able to yield high prediction accuracy compared with tree-based and embedding-based approaches. From Table 3, we also found that OVA-Primal++ is consistently faster than other OVA-based approaches for both high dimensional sparse and low dimensional dense datasets, and usually $2 \sim 3$ times faster than DiSMEC and OVA-Naive. Note that all of DiSMEC, OVA-Naive and OVA-Primal++ are built on the C code from LIBLINEAR; they have the same objective function

Copyright © 2019 by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 11/23/20 to 27.7.29.164. Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/page/terms



(a) Speedup on NUS-WIDE

(b) Speedup on EURLex

Figure 3: Speedup with different number of cores on NUS-WIDE and EURLex

Datasets	NUS-WIDE	EURLex	AmazonCat13k	AmazonCat14k	Wiki31k
Run time of OVA-MST	0.608	0.088s	5.317s	4.087s	2.582s

Table 4: Runtime of OVA-MST

and use exactly the same stopping criterion, thus their training times are comparable. For the recently proposed tree-based algorithm Parabel, although it is fast for high dimensional sparse text data, its training time on low dimensional dense data is inferior. The experimental results of training time demonstrate the effectiveness of our proposed method.

4.2 Run time of OVA-MST In Table 4, we present the time spent on solving the minimum spanning tree problem. Note that tiny-imagenet and Aloi are not included in the table since they are multiclass datasets and OVA-Primal++ can be reduced to OVA-Primal in this case. Consistent to our analysis in section 3.5, OVA-MST is efficient and the run time of OVA-MST can be ignored when compared with the training time of OVA classifiers.

4.3 Evalution for Parallelization In this part, we empirically evaluate the parallelizability of OVA-Primal++ in a multi-core environment. Experimental result is presented in Figure 3. As shown in the figure, similar to OVA-Naive, OVA-Primal++ could achieve nearly linear speedup and this implies that we can potentially further reduce the training time with more computing resources. Note that here we only use NUS-WIDE and EURLex for the experiments since the training using 1 core for larger datasets is too expensive.

5 Conclusion

In this paper, we considered the training of OVA linear classifiers when the number of labels is large. By exploiting the underlying relationships between all K subproblems, we proposed a novel algorithm OVA-Primal++ that uses a tree-structured ordering of training and could solve each subproblem with a better initialization. OVA-Primal++ enjoys the same advantage as naive OVA—highly parallelizable with a small memory footprint. Furthermore, it is both theoretically and empirically faster than naive OVA implementation. Different from recent works [5, 6] that only consider high dimensional sparse data, our method is more flexible and could be potentially used for a wider range of applications.

Acknowledgements. We would like to thank Yifan Sun for proof reading the first version of the paper.

References

[1] Rohit Babbar and Bernhard Schölkopf. Dismec: Distributed sparse machines for extreme multi-label classification. In *ACM International Conference on Web Search and Data Mining*, pages 721–729, 2017.

[2] Yichuan Tang. Deep learning with linear support vector machines. In *Workshop on Representational Learning, ICML*, 2013.

[3] Dhruv Mahajan, Ross Girshick, Vignesh Ramanathan, Kaiming He, Manohar Paluri, Yixuan Li, Ashwin Bharambe, and Laurens van der Maaten. Exploring

Copyright © 2019 by SIAM
Unauthorized reproduction of this article is prohibited

Downloaded 11/23/20 to 27.7.29.164. Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/page/terms

the limits of weakly supervised pretraining. In *ECCV*, 2018.

[4] Ian En-Hsu Yen, Xiangru Huang, Pradeep Ravikumar, Kai Zhong, and Inderjit S. Dhillon. Pd-sparse : A primal and dual sparse approach to extreme multiclass and multilabel classification. In *International Conference on Machine Learning*, pages 3069–3077, 2016.

[5] Ian En-Hsu Yen, Xiangru Huang, Wei Dai, Pradeep Ravikumar, Inderjit S. Dhillon, and Eric P. Xing. Ppdsparse: A parallel primal-dual sparse method for extreme classification. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 545–553, 2017.

[6] Y. Prabhu, A. Kag, S. Harsola, R. Agrawal, and M. Varma. Parabel: Partitioned label trees for extreme classification with application to dynamic search advertising. In *International World Wide Web Conference*, April 2018.

[7] Yashoteja Prabhu and Manik Varma. Fastxml: a fast, accurate and stable tree-classifier for extreme multi-label learning. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 263–272, 2014.

[8] Himanshu Jain, Yashoteja Prabhu, and Manik Varma. Extreme multi-label loss functions for recommendation, tagging, ranking & other missing label applications. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 935–944, 2016.

[9] Yao-Nan Chen and Hsuan-Tien Lin. Feature-aware label space dimension reduction for multi-label classification. In *Advances in Neural Information Processing Systems 25*, pages 1538–1546, 2012.

[10] Ashish Kapoor, Raajay Viswanathan, and Prateek Jain. Multilabel classification using bayesian compressed sensing. In *Advances in Neural Information Processing Systems 25*, pages 2654–2662, 2012.

[11] Ian En-Hsu Yen, Ting-Wei Lin, Shou-De Lin, Pradeep Ravikumar, and Inderjit S. Dhillon. Sparse random feature algorithm as coordinate descent in hilbert space. In *Advances in Neural Information Processing Systems 27*, pages 2456–2464, 2014.

[12] Kush Bhatia, Himanshu Jain, Purushottam Kar, Manik Varma, and Prateek Jain. Sparse local embeddings for extreme multi-label classification. In *Advances in Neural Information Processing Systems 28*, pages 730–738, 2015.

[13] Saeed Ghadimi and Guanghui Lan. Stochastic first- and zeroth-order methods for nonconvex stochastic programming. *SIAM Journal on Optimization*, 23(4):2341–2368, 2013.

[14] J. B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *American Mathematical Society*, 7(1):48–50, 1956.

[15] David A. Bader and Guojing Cong. Fast shared-memory algorithms for computing the minimum spanning forest of sparse graphs. *Journel of Parallel Distributed Computing*, 66(11):1366–1378, 2006.

[16] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

[17] Chih-Jen Lin, Ruby C. Weng, and S. Sathiya Keerthi. Trust region newton methods for large-scale logistic regression. In *International Conference of Machine Learning, ICML 2007*, pages 561–568, 2007.

[18] Chih-Yang Hsia, Ya Zhu, and Chih-Jen Lin. A study on trust region update rules in newton methods for large-scale linear classification. In *Asian Conference on Machine Learning*, pages 33–48, 2017.

[19] Ron S. Dembo, Stanley C. Eisenstat, and Trond Steihau. Inexact newton methods. *SIAM Journal of Numerical Analysis*, 19(2):400–408, 1982.

[20] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[22] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011.

[23] Wei Bi and James Kwok. Efficient multi-label classification with many labels. In *International Conference on Machine Learning*, volume 28, pages 405–413, 2013.

[24] Si Si, Huan Zhang, S. Sathiya Keerthi, Dhruv Mahajan, Inderjit S. Dhillon, and Cho-Jui Hsieh. Gradient boosted decision trees for high dimensional sparse output. In *International Conference on Machine Learning*, pages 3182–3190, 2017.

[25] Tat-Seng Chua, Jinhui Tang, Richang Hong, Haojie Li, Zhiping Luo, and Yan-Tao Zheng. Nus-wide: A real-world web image database from national university of singapore. In *ACM Conference on Image and Video Retrieval*, 2009.

[26] Julian J. McAuley and Jure Leskovec. Hidden factors and hidden topics: understanding rating dimensions with review text. In *ACM Conference on Recommender Systems*, pages 165–172, 2013.

[27] Julian J. McAuley, Rahul Pandey, and Jure Leskovec. Inferring networks of substitutable and complementary products. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794, 2015.

[28] Eneldo Loza Mencía and Johannes Fürnkranz. Efficient pairwise multilabel classification for large-scale problems in the legal domain. In *Machine Learning and Knowledge Discovery in Databases, European Conference, ECML/PKDD*, pages 50–65, 2008.

[29] Ryan M. Rifkin and Aldebaro Klautau. In defense of one-vs-all classification. *Journal of Machine Learning Research*, 5:101–141, 2004.

Copyright © 2019 by SIAM
Unauthorized reproduction of this article is prohibited