



# Speeding-up one-versus-all training for extreme classification via mean-separating initialization

Erik Schultheis<sup>1</sup> · Rohit Babbar<sup>1</sup>

Received: 21 February 2022 / Revised: 14 June 2022 / Accepted: 22 July 2022 /  
Published online: 18 October 2022  
© The Author(s) 2022

## Abstract

In this paper, we show that a simple, data dependent way of setting the initial vector can be used to substantially speed up the training of linear one-versus-all classifiers in extreme multi-label classification (XMC). We discuss the problem of choosing the initial weights from the perspective of three goals. We want to start in a region of weight space (a) with low loss value, (b) that is favourable for second-order optimization, and (c) where the conjugate-gradient (CG) calculations can be performed quickly. For margin losses, such an initialization is achieved by selecting the initial vector such that it separates the mean of all positive (relevant for a label) instances from the mean of all negatives – two quantities that can be calculated quickly for the highly imbalanced binary problems occurring in XMC. We demonstrate a training speedup of up to 5× on Amazon-670K dataset with 670,000 labels. This comes in part from the reduced number of iterations that need to be performed due to starting closer to the solution, and in part from an implicit negative-mining effect that allows to ignore easy negatives in the CG step. Because of the convex nature of the optimization problem, the speedup is achieved without any degradation in classification accuracy. The implementation can be found at <https://github.com/xmc-aalto/dismecpp>.

**Keywords** Large-scale multi-label classification · Linear classification · 2nd order optimization · Class imbalance · Weight initialization

## 1 Introduction

*Extreme classification:* In this work we consider *extreme multi-label classification* (XMC) problems, where the number of labels  $l$  is very large, possibly in the millions. Such problems arise in various domains, such as annotating large encyclopedia (Partalas et al., 2015),

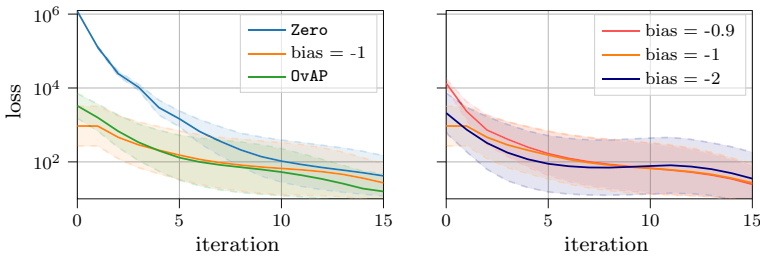
---

Editors: Krzysztof Dembczynski and Emilie Devijver.

✉ Erik Schultheis  
erik.schultheis@aalto.fi

Rohit Babbar  
rohit.babbar@aalto.fi

<sup>1</sup> Department of Computer Science, Aalto University, Helsinki, Finland



**Fig. 1** Objective value at different iterations for starting with **Zero** vector, **OvAP** and with **Bias**-initialization  $\mathbf{w}_b$ . The right side shows the progress for different values of the bias weight. The shaded area spans the 5–95% quantiles. Note that the slight increase in objective for the later iterations is just an artifact due to the fact that we plot the loss values averaged for the binary sub-problems that have not yet terminated at the given iteration. The sub-problems with low loss value terminate earlier, causing an increase in average of the remaining trajectories. Figure 12 shows the number of iterations until convergence for the different bias values

image-classification (Deng et al., 2010), next word prediction (Mikolov et al., 2013; Mnih et al., 2009), as well as recommendation systems, web-advertising and prediction of related searches (Agrawal et al., 2013; Prabhu & Manik, 2014; Jain et al., 2019).

We assume that the fraction of positive training instances for most labels is very low – the label frequency has a long-tailed distribution (Dekel et al., 2010). This is a good approximation for the large dataset from the extreme classification repository (Bhatia et al., 2016) (cf. Fig. 1 in Qarai et al. (2021), Babbar & Schölkopf (2017), and Partalas et al. (2015) for some examples), and for many types of data that are gathered at internet-scale (Adamic et al., 2002).

*One-versus-all classifiers:* Many XMC methods employ some form of a *One-versus-all* (OvA) classifier as the last stage of the classification procedure: For each label  $j \in [L]$ , a score is calculated as  $\mathbf{w}_j^T f(\mathbf{x})$ , and the  $k$  highest-scoring labels are selected as the prediction of the classifier,

$$\hat{y} = \text{top}_k \{ \mathbf{w}_j^T f(\mathbf{x}) : j \in [L] \}, \tag{1}$$

where  $\mathbf{w}_j$  is the weight vector for label  $j$  and  $f$  is a *fixed* function of the instance. The training objective is to minimize, with some additional regularization terms, a binary margin loss  $\phi$  for each label  $j$ ,

$$\phi(y_j \mathbf{w}_j^T f(\mathbf{x})), \tag{2}$$

where  $y_j \in \{-1, 1\}$  indicates whether the label is relevant to the instance.

Algorithms that follow this general structure are DiSMEC (Babbar & Schölkopf, 2017) and ProXML (Babbar & Schölkopf, 2019) for sparse representations  $f$ , and Slice (Jain et al., 2019) where  $f$  is some pre-trained mapping of instances to dense vectors. Other examples are Parabel (Prabhu et al., 2018), Bonsai (Khandagale et al., 2020) and PPDSParse (Yen et al., 2017). Embedding-based approaches where the mapping  $f$  itself is learnt, such as XML-CNN (Liu et al., 2017) and AttentionXML (You et al., 2019) are incompatible with the method presented in this paper. In some cases the training is organized in two phases, first the deep network  $f$  is trained and then frozen, after which the OVA classifier can be trained. This approach is taken e.g. by X-Transformer (Chang et al., 2020) and its successor XR-Transformer (Zhang et al., 2021).

Despite the substantial contemporary research effort on designing deep classifiers in the XMC setting, efficient learning of sparse One-versus-rest linear classifiers remains useful. This is attributed to their: (i) easier interpretability for explaining the model outputs in terms of input features, (ii) comparable, and at times better performance, compared to deeper methods in short-text XMC problems (DiSMEC vs AttentionXML in Table 1 of Dahiya et al. (2021)), and (iii) complementary learning capabilities using sparse features vis-à-vis dense representation using deep transformer models, as shown in the recent work XR-Transformer (Zhang et al., 2021).

*Fast training:* Minimizing the objective in eq. (2) using a gradient-based iterative algorithm is very computation-intensive. To reduce compute requirements – and thus energy consumption – we can make use of three generic principles:

1. Try to achieve more progress per step, e.g. by using a second-order method (Babbar & Schölkopf, 2017; Keerthi et al., 2005; Galli et al., 2021) or an adaptive step size. (Duchi et al., 2011; Ruder, 2017)
2. Make the computations of each step faster, e.g. by approximating the true loss with negative mining. (Jain et al., 2019; Dahiya et al., 2021; Prabhu et al., 2018; Yen et al., 2017; Reddi et al., 2019).
3. Reducing the number of necessary steps by taking a good guess for the initial weight vector in the iterative optimization process. (Fang et al., 2019; Keerthi et al., 2005).

In this work, we focus on the second-order optimization approach using a truncated conjugate-gradient (CG) Newton optimization, as it is used in recent versions of Liblinear (Galli et al., 2021).<sup>1</sup>

*Implicit negative-mining with hinge-like losses:* To a degree, faster per-step calculations are achieved implicitly when using a loss function that is zero for points classified correctly with sufficient margin. Such a training point makes no contribution to the gradient or Hessian, and thus can be skipped in the computations. This can be seen as a form of implicit negative-mining, where the list of negatives is updated each iteration (similar to Yen et al. (2017)) based on the values of  $y_i \mathbf{w}^\top f(\mathbf{x}_i)$ . Unlike explicit negative-mining, this process is not an approximation. However, it does not enforce sparsity, but only exploits its existence. Fortunately, close to convergence, when most instances are correctly classified, this leads to very sparse computations for the CG procedure. Thus, one aim in choosing the initial weights is to get into this fast regime as quickly as possible.

*Choice of initial vector:* The third idea, choosing a suitable starting weight, is used in Liblinear to speed up hyperparameter sweeps. It is assumed that the solutions for two similar hyperparameter values are closely related, and as such the final weight vector of one training run can be used to warm-start the next. The feasibility of this approach in cold-start XMC has been shown in Fang et al. (2019), which presented *OvA-Primal* (OvAP) and *OvA-Primal++* (OvAP++) as two strategies for finding initial vectors that result in low initial loss. We provide a short summary of these methods at the end of Sect. 2.

*Shortcomings of the existing work:* OvAP, though simple and easy to implement, completely ignores the positive instances for each label. We will show that there is a computationally cheap method that allows taking them into account.

<sup>1</sup> Older versions (Fan et al., 2008) use a trust-region Newton method.

In contrast,  $\text{OvAP}^{++}$  makes use of the positives, but requires much more care to implement, as it breaks the embarrassingly parallel nature of the original OvA approach. Hence, the individual binary problems need to be solved in a certain (partial) order. This also precludes the application of this strategy to situations where all labels have to be trained simultaneously, e.g. when fine-tuning the feature representations  $f$ .

The existing analysis in Fang et al. (2019) focuses only on finding an initial weight vector that has loss as low as possible. Important properties of the loss function (such as local smoothness / Lipschitz constant) sometimes are more favourable after some training epochs than right at initialization time, as evidenced e.g. by the success of cyclical learning rates (Smith & Leslie, 2017) and warmup phases (Goyal et al., 2018). However, the impact of these properties is not considered in Fang et al. (2019).

*Contributions:* Our contributions in this work are:

1. We explicate via extensive empirical evaluation that, when employed in conjunction with the squared-hinge loss, the benefit of initialization methods is not limited to just starting closer to the final weight vector. Good initialization also leads to much faster computations in each training step, because of an implicit negative-mining effect.
2. We present a novel initialization strategy that is computationally efficient, keeps the simplicity of  $\text{OvAP}$ , but incorporates information about the positive instances for each label to find even better initial weights. This is achieved by selecting the initial vector such that it separates the mean of all positive instances from the mean of all negatives.
3. We show experimentally that this leads to a speedup of up to  $5\times$  relative to  $\text{Zero}$ -initialization for our code-base on the Amazon-670K dataset, and demonstrate that this enables training of DiSMEC-style models with up to 3 million labels on a single 128-core node in a reasonable amount of time.
4. We provide a new implementation of DiSMEC that can adapt to the NUMA characteristics of modern many-core processors. Combined with the new choice of the initial vector, this can lead to a speedup of up to  $14\times$  in wallclock time.

*Outline:* In Sect. 2, we formally introduce our learning setting and recap the key concept of optimization using conjugate-gradients. This is followed by a presentation of the key results from Fang et al. (2019), the  $\text{OvAP}$  and  $\text{OvAP}^{++}$  initialization methods. In Sect. 3, we investigate which properties an initial weight vector needs to have to be beneficial for optimization. In particular, we show that low loss alone is not a sufficient criterion by providing a counter-example. We investigate why it fails, and thus highlight the importance of the local loss landscape and the implicit negative-mining effect. The description of our proposed method is given in Sect. 4, together with an evaluation of its effect on the local loss landscape and implicit negative-mining. In Sect. 5, an extensive empirical evaluation of the new initialization method with a variety of datasets and settings is presented. Finally, Sect. 6 discusses the limitations of our approach and given an outlook on how it could be applied more broadly in XMC.

## 2 Background

*Setup:* We are given a dataset  $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i) : i \in [n]\}$  with  $n$  training instances  $\mathbf{x}_i \in \mathcal{X} = \mathbb{R}^d$  and their corresponding labels  $\mathbf{y}_i \in \mathcal{Y} = \{-1, 1\}^l$ . For a label  $j \in [l]$  we denote with  $P(j) := \{\mathbf{x}_i : y_i^{(j)} = 1\}$  the set of instances for which the label is relevant, and analogously

$N(j) := X \setminus P(j)$ , where  $X := \{\mathbf{x}_i : i \in [n]\}$  is the set of all training instances. This corresponds to a label vector  $\mathbf{y}^{(j)}$  which has entries 1 for each positive instance,  $y_i^{(j)} = 1 \Leftrightarrow i \in P(j)$ , and  $-1$  otherwise.

In the linear OVA setting, we have to solve  $l$  independent problems of size  $O(n)$ . We assume that the number of labels  $l$  is large enough so that an additional  $O(n)$  computation does not significantly influence the overall training time. In particular, the initialization method that will be presented in Sect. 4 requires calculating the mean of all instances,  $\bar{\mathbf{x}} = \sum_{i=1}^n \mathbf{x}_i$ , which is a cheap operation compared to solving the  $l$  subproblems. In addition, we assume that the labels are sparse, in the sense that for most  $j \in [l]$  we have  $|P(j)| \ll |N(j)|$ . These two assumptions are part of the characterization of an XMC problem.

The goal in linear XMC is to find a weight vector  $\mathbf{w}_j^*$  for each label  $j \in [l]$  that minimizes the risk of the corresponding binary problem with some convex margin loss  $\phi$ , combined with a regularizer  $\mathcal{R}$ :

$$\mathcal{L}[\mathbf{w}, \mathbf{y}^{(j)}] := \sum_{i=1}^n \phi\left(y_i^{(j)} \mathbf{w}^\top \mathbf{x}_i\right) + \mathcal{R}[\mathbf{w}] \quad (3)$$

$$\mathbf{w}_j^* := \mathbf{w}^*(X, \mathbf{y}^{(j)}) := \underset{\mathbf{w}}{\operatorname{argmin}} \mathcal{L}[\mathbf{w}, \mathbf{y}^{(j)}]. \quad (4)$$

Throughout this paper, we use the  $L_2$ -norm  $\mathcal{R}[\mathbf{w}] = 0.5\|\mathbf{w}\|_2^2$  as the regularizer. From hereon, we will focus on the independent binary sub-problems, and consequently drop the superscript  $\mathbf{y}^{(j)}$  and write  $\mathbf{y} \in \{-1, 1\}^n$  for the vector indicating the relevance of instances for a given label. Following Jain et al. (2019), Babbar & Schölkopf (2017, 2019), as the margin loss  $\phi$  we use the squared hinge-loss

$$\phi(m) = \max(0, 1 - m)^2. \quad (5)$$

Even though this function's second derivative does not exist at  $m = 1$ , it can be used for second-order optimization (Galli et al., 2021).

*Conjugate-gradient Newton optimization:* The minimization of eq. (3) is carried out using a CG Newton procedure inspired by Liblinear (Fan et al., 2008; Galli et al., 2021). The algorithm works as follows: First, a descent direction  $\mathbf{p}$  is determined by minimizing a local quadratic approximation  $\hat{\mathcal{L}}$  to the loss function

$$\hat{\mathcal{L}}(\mathbf{w} + \boldsymbol{\delta}) = \mathcal{L}(\mathbf{w}) + \nabla \mathcal{L} \cdot \boldsymbol{\delta} + \frac{1}{2} \boldsymbol{\delta}^\top \mathbf{H}_{\mathcal{L}} \boldsymbol{\delta}, \quad (6)$$

where  $\mathbf{H}$  denotes the Hessian of  $\mathcal{L}$ . The stationarity condition for a local minimum,

$$0 = \nabla \mathcal{L} + \mathbf{H}_{\mathcal{L}} \boldsymbol{\delta}^*, \quad (7)$$

shows that the optimal change  $\boldsymbol{\delta}^*$  of weights – in quadratic approximation – can be found by solving a system of linear equations. The conjugate-gradient algorithm provides a way of solving the equations without ever requiring an explicit representation of the matrix  $\mathbf{H}_{\mathcal{L}}$ . Instead, only Hessian-vector products are needed, which can be calculated for a pointwise regularization  $\mathcal{R}[\mathbf{w}] = \sum_{j=1}^m \rho(w_j)$  as (Keerthi et al., 2005)

$$\mathbf{H} \mathbf{d} = \left( \sum_{i=1}^n \nabla_{\mathbf{w}}^2 \phi(y_i \mathbf{w}^\top \mathbf{x}_i) + \sum_{j=1}^m \nabla_{\mathbf{w}}^2 \rho(w_j) \right) \mathbf{d} \quad (8)$$

$$= \left( \sum_{i=1}^n y_i \mathbf{x}_i \nabla_{\mathbf{w}} \phi'(y_i \mathbf{w}^\top \mathbf{x}_i) \right) \mathbf{d} + \text{diag}(\rho''(\mathbf{w})) \mathbf{d} \tag{9}$$

$$= \left( \sum_{i=1}^n y_i^2 \mathbf{x}_i \mathbf{x}_i^\top \phi''(y_i \mathbf{w}^\top \mathbf{x}_i) \right) \mathbf{d} + \text{diag}(\rho''(\mathbf{w})) \mathbf{d} \tag{10}$$

$$= \sum_{i=1}^n \mathbf{x}_i \phi''(y_i \mathbf{w}^\top \mathbf{x}_i) \langle \mathbf{x}_i, \mathbf{d} \rangle + \text{diag}(\rho''(\mathbf{w})) \mathbf{d} \tag{11}$$

For the case of  $L_2$  regularization, the second term simplifies to  $\text{diag}(\rho''(\mathbf{w})) = \mathbb{1}$ . In practice, (7) need not be solved exactly, and the conjugate-gradient procedure can be terminated after a few iterations.

Once the quadratically optimal step  $\mathbf{p}$  has been found, a line-search using a backtracking approach is carried out over  $\mathbf{w} + \lambda \mathbf{p}$ ,  $\lambda \in (0, 1]$ . This ensures that the loss at the new weight vector is actually an improvement,  $\mathcal{L}[\mathbf{w} + \lambda^* \mathbf{p}] < \mathcal{L}[\mathbf{w}]$ , even if the quadratic approximation deviates from the true loss. If the new point fulfills the stopping criterion as discussed below, the optimization terminates. Otherwise a new quadratic approximation around  $\mathbf{w} + \lambda^* \mathbf{p}$  is formed and the next update step is calculated.

*Stopping criterion:* Whether or not the optimization is stopped depends on the magnitude of the gradient. We use the criterion given by Liblinear:

$$\|\nabla \mathcal{L}[\mathbf{w}^*]\| \leq \epsilon \|\nabla \mathcal{L}[\mathbf{0}]\|. \tag{12}$$

The scaling factor  $\epsilon$  depends on the imbalance in the data and is calculated through

$$\epsilon = \epsilon_0 \cdot \frac{\max(1, \min(|P|, |N|))}{n}. \tag{13}$$

This means that the convergence criterion gets stricter the more imbalanced the training data, which ensures that simply predicting the majority class for all instances is not accepted as a solution. Unless otherwise noted, we use  $\epsilon_0 = 0.01$ , the default value of Liblinear.

*OvA-primal and OvA-primal++:* In order to reduce the number of iterations needed to achieve the stopping criterion, Fang et al. (2019) propose two schemes for selecting the initial weights  $\mathbf{w}_0$ , based on the following observation: For  $\alpha$ -Lipschitz loss functions and bounded weights  $\|\mathbf{w}\|_2 < B$ , the optimal weight vector  $\mathbf{w}^*(\mathbf{y})$  for one label vector  $\mathbf{y}$  results in good performance also for a different label vector  $\mathbf{y}'$  if the two label vectors are close in Hamming distance  $\ell_H$ :

$$\mathcal{L}[\mathbf{w}^*(\mathbf{y}), \mathbf{y}'] - \mathcal{L}[\mathbf{w}^*(\mathbf{y}'), \mathbf{y}'] \leq \text{const} \cdot \ell_H(\mathbf{y}, \mathbf{y}'). \tag{14}$$

Given that for tail labels it holds that  $\ell_H(\mathbf{y}, -\mathbf{1}_n) \ll n$ , they conclude that using  $\mathbf{w}^*(-\mathbf{1}_n)$  makes the initial vector result in much lower loss than using  $\mathbb{Z}_{\infty 0}$ -initialized weights. Formally, the goal is to find the optimal weight for all-negative labels by solving

$$\mathbf{w}^*(-\mathbf{1}_n) = \underset{\mathbf{w}}{\text{argmin}} \sum_{i=1}^n \phi(-\mathbf{w}^\top \mathbf{x}_i) + \mathcal{R}[\mathbf{w}]. \tag{15}$$

This strategy is called OvA-Primal (OvAP). As discussed above, the number of labels  $l$  is assumed to be large enough that this one additional training run will not take a significant portion of the overall training time.

Then, based on convergence rates of iterative minimizers, they estimate the speedup that OvA-Primal, given by  $\mathbf{w}_0 = \mathbf{w}^*(-\mathbf{1}_n)$ , provides over  $\mathbf{w}_0 = \mathbf{0}_d$ . For an optimization algorithm that has linear convergence, i.e. that needs

$$O\left(\log \frac{\mathcal{L}[\mathbf{w}_0, \mathbf{y}] - \mathcal{L}[\mathbf{w}^*(\mathbf{y}), \mathbf{y}]}{\epsilon}\right) \quad (16)$$

steps for training to precision  $\epsilon$ , the speedup in the asymptotic bound is  $\log n$  (Fang et al., 2019, Thm.3.2) if the average number of samples per label does not scale with  $n$  or  $l$ .

OvA-Primal++ (OvAP++) extends the idea of OvAP by additionally re-using the final weight vectors  $\mathbf{w}_j^*$  of one label  $j$  as the initial weights for similar labels  $j'$ . To achieve this, they first build a *minimum spanning tree* over the labels, including the virtual all-negative label as the root, where the cost of an edge is given by the Hamming distance between the label vectors. Starting from the root, one can initialize the descendants of a label by the final weights of that label. Due to the low Hamming-distance, this should provide a good starting point. This process is implemented as a blocked depth-first search over the tree as a compromise between parallelism and memory requirements. OvAP++ provides further speedups over OvAP, but at the cost of significantly increased implementation complexity. In particular, the sub-problems can no longer be solved independently from each other.

### 3 Analysis of criteria for initial weights

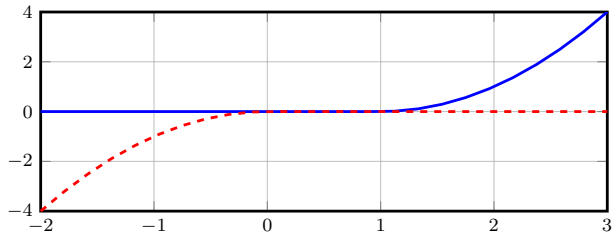
In this section, we investigate which properties an initial weight vector needs to fulfill in order to improve the training speed. The first criterion – to start with a low loss value, and correspondingly closer to the optimal weight vector – has been used to motivate the OvA-Primal method described above. Here, we show that low initial loss alone is not a sufficient criterion for successful initial weights. In particular, we present a counterexample, in which the weight vector leads to very low initial loss, orders of magnitude smaller than initializing by zero, yet does not provide any significant speed-up. We identify two reasons for this: First, the loss landscape around the initial vector needs to be favourable for second-order optimization. Second, a significant part of the speed-up is in fact not due to the reduced distance to the optimal vector and correspondingly smaller number of iterations, but is instead caused by faster computations in each iteration, owing to an *implicit negative-mining* effect.

To illustrate these phenomena, we take the AmazonCat-13K dataset (details in the appendix) from the extreme classification repository (Bhatia et al., 2016; McAuley & Jure, 2013) as a running example in this section and the next.

*Insufficiency of low loss:* A simple way to calculate an approximation to  $\mathbf{w}^*(-\mathbf{1}_n)$ , the optimal weight-vector that predicts the absence of the label for every instance, is to use a weight vector  $\mathbf{w}_b = (-1, 0, \dots)^\top$ , where we assume that the bias feature  $x_0 = 1$  is at index 0.<sup>2</sup>

<sup>2</sup> Such a strategy seems to have been considered by Fang et al. (2019), as it can be found in their code at <https://github.com/fanghg/XMC>, though it is not mentioned in the paper.

**Fig. 2** Approximation errors  $e(\delta; m_0)$  for  $m_0 = 0$  (solid blue) and  $m_0 = 1$  (dashed red) over  $\delta$ , as described by eq. (17) (Color figure online)



Then, the score for any instance  $\mathbf{x}$  will be  $m = \mathbf{w}_b^T \mathbf{x} = -1$ , which means that it is classified as negative with margin one. Therefore, this weight vector is a minimizer of the squared hinge-loss without regularization. As can be seen in Fig. 1 (left), the initial value of the loss is decreased by three orders of magnitude, to a lower value than with the  $\text{OvAP}$  initializer. Yet, after some iterations, the  $\text{OvAP}$  based optimization overtakes  $\text{Bias}$ . In particular, the  $\text{Bias}$ -initialized optimization does not make any substantial progress during the first iteration.

This does not contradict the previous result, because the asymptotic convergence rate based on the number of iterations does not need to agree with actual computation time, for example if each iteration provides more progress towards the minimum due to a benign loss landscape as described in the next paragraph, or if the computations for a single iteration become faster due to implicit negative-mining.

*Importance of loss landscape:* Certain regions in the weight space may be more benign towards the chosen minimization procedure than others. For example, in deep networks one often chooses an initialization procedure that preserves variance and mean over layers, to prevent vanishing or exploding gradients (Glorot et al., 2010). Such a technique does not provide any reduced loss for the starting point, but can be very effective in speeding up the training.

For the concrete case of Newton optimization, this means that we require the local quadratic approximation to fit the true loss well. As  $\phi$  is piecewise-quadratic, this is fulfilled only when  $m$  is far from the discontinuity at  $m = 1$ . Let  $\hat{\phi}_{m_0}$  be the quadratic approximation, then we can calculate the approximation error based on the step size as

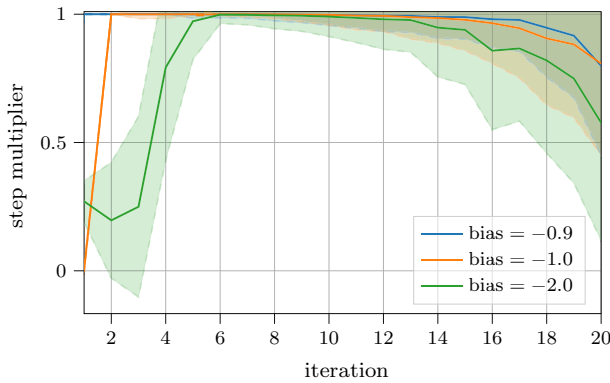
$$e(\delta; m_0) := \hat{\phi}_{m_0}(\delta) - \phi(m_0 + \delta), \tag{17}$$

$$\text{where } \hat{\phi}_{m_0}(\delta) = \phi(m_0) + \delta \phi' \Big|_{m_0} + 0.5 \delta^2 \phi'' \Big|_{m_0}. \tag{18}$$

Two examples are shown in Fig. 2. For an instance right at the discontinuity, improving the classification margin to exceed one causes the quadratic approximation to over-estimate the true loss value, leading to smaller proposed update vectors  $\mathbf{p}$ . On the other hand, for any instance classified with at least margin 1, the quadratic approximation cannot “see” the increased error as the margin shrinks, and may propose overly large steps  $\mathbf{p}$  that have to be shrunk using line search.

By choosing  $-1$  for the bias weight, the initial weights are exactly at the discontinuity of the Hessian. This suggests that it might be beneficial to use a different value for the bias, e.g.  $\mathbf{w}_0 = 0.9\mathbf{w}_b$  or  $\mathbf{w}_0 = 2\mathbf{w}_b$ . This intuition is confirmed by Fig. 1 (right), where we can see that both variations make significant progress in the first iteration, with  $2\mathbf{w}_b$  clearly outperforming  $\mathbf{w}_b$ . A more detailed look at this phenomenon is provided by Fig. 3, which shows that the acceptable step size along the chosen descent direction is very small if  $\mathbf{w}_0$  is close to  $\mathbf{w}_b$ . This indicates that the quadratic approximation is overly





**Fig. 3** Multiplier found by backtracking line search, i.e. the fraction of the descent direction as found by the CG solver that can be applied to the real objective function and still lead to sufficient reduction in loss. The shaded area indicates  $\pm$  one standard deviation. Having a value smaller than one indicates that the quadratic approximation to the loss is overly optimistic in the given direction, and the real step has to be scaled down. As discussed in the main text (Fig. 2), this happens when the direction in question is such that it reduces the margin of an instance that is classified correctly with margin more than one. This effect can be seen here: For bias = -0.9, all the negative instances still have non-zero error, so the update will choose a direction that makes them more negative, which is a direction in which the quadratic approximation is overestimating the loss. On the other hand, with bias = -1, the only nonzero signal is coming from the positive instances, which will cause the bias to be reduced. Thus, most of the negative instances will get nonzero error, and the quadratic approximation severely underestimates the true loss. Consequently, the line search shows that only a minute step towards the desired direction is allowed. For bias = -2, the situation is similar, but this time the distance for each negative instance until it gets nonzero error is much larger, and as such the underestimation effect is reduced and larger step size multipliers are possible

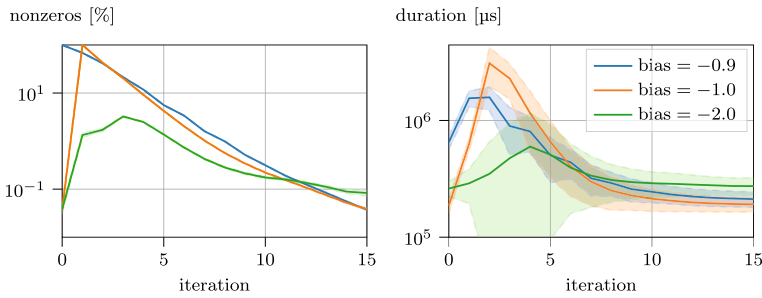
optimistic, and proposes much larger steps than optimal. This happens very strongly in the first epoch of bias=-1, and to some degree during the first few epochs of bias = -2.

For the squared hinge-loss, a vector that minimizes eq. (15) perfectly will have similar characteristics to the Bias initialization  $w_b$ , in that for many instances the prediction will be close to the hinge location. Thus the starting point will be in a location unfavourable to the Newton optimizer. A remedy is to stop training of  $w^*(-1_n)$  early. This results in larger initial loss, but makes the training progresses more smoothly. More details on this phenomenon are presented in appendix 7.2.

Even though the local loss landscape can explain why the first Newton-optimization step makes so little progress, this alone is not sufficient to explain the vastly different training durations: For  $2w_b$ , training takes only 558 seconds, much faster than  $w_b$  and  $0.9w_b$  at 1161 and 995 seconds. In addition to fewer iterations, the initialization method can also influence the amount of computation necessary during each iteration, through an implicit negative-mining step as discussed below.

*Implicit negative-mining:* As the weight vector  $w$  approaches its optimum  $w^*$ , the calculations of Hessian-vector products become sparse, because only instances with non-zero loss need to be taken into account: For  $m > 1$  we have  $\phi(m) = \phi'(m) = \phi''(m) = 0$ , which implies that the sum over all data points in eq. (11) can be replaced with a sum over implicitly-mined hard instances

$$A = \{i : \phi''(x_i^T w \cdot y_i) \neq 0\} = \{i : x_i^T w \cdot y_i < 1\}. \tag{19}$$



**Fig. 4** The average sparsity of Hessian-vector product computations, and corresponding duration of one iteration of the Newton optimization. The averages are taken over the individual binary problems, and the shaded area shows the  $2\sigma$  error of the mean. The total computation time includes a (sparse) matrix multiplication and a varying number of CG steps. As this data shows, even though bias = -1 and bias = -2 start very sparse, much of the sparsity is lost during the first training steps. For the later iterations, -2 results in slightly less sparsity and slower iterations, however the effect on total running time is far overshadowed by the much faster earlier iterations (note the logarithmic axes). Furthermore, as indicated by Fig. 12, for bias = -2 fewer iterations are needed, so the timings of iterations after the tenth are less important

Therefore, the computations in eq. (11) can be written as

$$Hd = \sum_{i=1}^n \mathbf{x}_i \phi''(y_i \mathbf{w}^T \mathbf{x}_i) \langle \mathbf{x}_i, \mathbf{d} \rangle + \mathbf{d} = \sum_{i \in \mathcal{A}} \phi''(\mathbf{x}_i^T \mathbf{w} \cdot y_i) y_i \langle \mathbf{x}_i, \mathbf{d} \rangle \cdot \mathbf{x}_i + \mathbf{d}, \tag{20}$$

which can be computationally much more efficient than the full sum (Keerthi et al., 2005). The active set  $\mathcal{A}$  has to be re-calculated after each weight update. Because each weight update requires multiple  $Hd$  calculations for the conjugate-gradient procedure, this still results in a net speed-up.

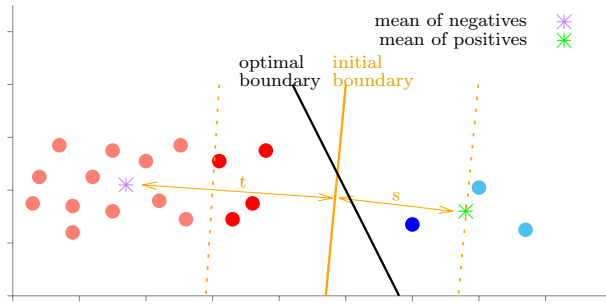
For different Bias-initializations  $\lambda \mathbf{w}_b$ ,  $\lambda \geq 1$ , the active set at the first iteration is given by are all the negatives,  $\mathcal{A} = N$ . However, for  $\lambda = 1$  the very first optimization step undoes the sparsity, so even though the first iteration is fast, it does not make progress towards the objective and makes the following iterations slow again, see Fig. 4. To a much lesser degree, this also happens for  $\lambda = 2$ . For  $\lambda < 1$ , the initial weights are not at the discontinuity of the Hessian, but the quadratic approximation now overestimates the actual loss function. Further, there is no sparsity in the first iteration, causing it to be very slow.

The implicit negative-mining effect, even though not explicitly considered by Fang et al. (2019), still contributes to the speed-up of OvAP and OvAP++. The OvAP leads to significant sparsity in the first iteration, as Fig. 6 below will indicate.

Thus, our goal is to find a new initialization method that starts with a high sparsity, in such a way that the first optimization steps do not turn the Hessian computations dense again as is the case with Bias. Such an initialization procedure is presented in the next section.

### 4 Mean-separating initialization

*Motivation:* In this section, we derive a simple way to generate an initial vector, motivated by the following observation: If the data is linearly separable with margin, then the final weight vector will separate the convex hull of the negatives  $N$  from that of the positives  $P$ , which in particular implies that it separates the centres of mass  $\bar{\mathbf{p}}$  of the positives and  $\bar{\mathbf{n}}$  of



**Fig. 5** Motivation for the Average-of-Positives initialization. If the data is linearly separable, then the feature-means of the positives (blue) and the negatives (red) will lie on the correct side of the optimal separating hyperplane (black line). Thus we may restrict our search for the initial vector to the set of the separating planes of  $\bar{\mathbf{p}}$  and  $\bar{\mathbf{n}}$  (orange). In order to achieve strong implicit negative mining in the first epoch, we choose a solution where the center of mass of the negatives is classified correctly with a large margin (dashed orange lines), so that most of the negatives (light red) will be classified correctly with a margin over 1 and thus not enter the Hessian computation (Color figure online)

the negatives. This means that we can restrict the search space of weight vectors to those that separate  $\bar{\mathbf{p}}$  and  $\bar{\mathbf{n}}$ . A sketch of this situation is given in Fig. 5.

Even though perfect separability might not be achieved on real data, it needs to hold at least approximately. Fortunately, the high-dimensional tf-idf<sup>3</sup> text data that is commonly used in XMC fulfills this property, as evidenced by the high performance achievable with linear classifiers.

*Derivation:* Without making use of any additional information about the training data (as we want a computationally cheap procedure), there are only very general conditions we can impose to choose among these hyperplanes. As a first step, we can parameterize the search space based on the margins of  $\bar{\mathbf{p}}$  and  $\bar{\mathbf{n}}$ , setting

$$\langle \mathbf{w}_0, \bar{\mathbf{p}} \rangle = s, \quad \langle \mathbf{w}_0, \bar{\mathbf{n}} \rangle = t, \tag{21}$$

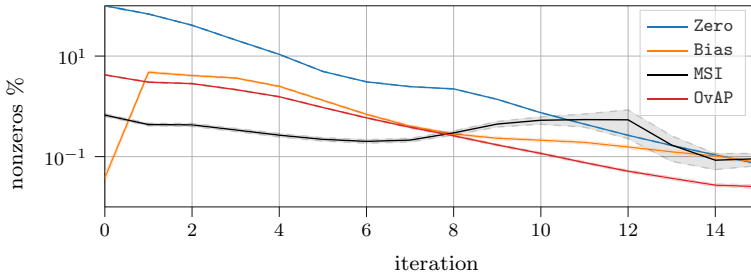
for two hyperparameters  $s$  and  $t$ .

These are only two linear constraints in the high dimensional weight space. However, for good generalization, we prefer weight vectors with minimal norm. If we want a minimum  $L_2$ -norm solution, then the search space becomes restricted to  $\text{span}(\bar{\mathbf{p}}, \bar{\mathbf{n}})$ , and we can parameterize  $\mathbf{w}_0 = u\bar{\mathbf{p}} + \tilde{v}\bar{\mathbf{n}}$ . This leads to a system of two linear equations in the two unknowns  $u$  and  $\tilde{v}$ , with a unique solution except in the unlikely event that  $\bar{\mathbf{p}}$  and  $\bar{\mathbf{n}}$  are linearly dependent.

*Reparameterization for efficiency:* Since most labels have only few positive instances, their mean  $\bar{\mathbf{p}}$  can be calculated quickly, but calculating  $\bar{\mathbf{n}}$  directly would be an  $O(n)$  operation for each label. However, we can precompute the mean of all instances  $\bar{\mathbf{x}}$ , and use the property

$$|N|\bar{\mathbf{n}} + |P|\bar{\mathbf{p}} = |X|\bar{\mathbf{x}}. \tag{22}$$

<sup>3</sup> term-frequency-inverse-document-frequency. Each entry in the feature vector represents how often a term appears in the source document, normalized by how often that term appears in the entire corpus, see e.g. (Shalev-Shwartz et al., 2014, Chapter 17)



**Fig. 6** Sparsity of the Hessian calculation over different training epochs. Even though Bias-initialization starts out very sparse, this property is lost after the first update step. In contrast, MSI maintains sparsity over the entire range of training iterations

Because  $\text{span}(\bar{\mathbf{p}}, \bar{\mathbf{n}}) = \text{span}(\bar{\mathbf{p}}, \bar{\mathbf{x}})$ , we can make the equivalent parameterization  $\mathbf{w}_0 = u\bar{\mathbf{p}} + v\bar{\mathbf{n}}$ , leading to the equations

$$s = \langle \mathbf{w}_0, \bar{\mathbf{p}} \rangle = \langle u\bar{\mathbf{p}} + v\bar{\mathbf{x}}, \bar{\mathbf{p}} \rangle = v\langle \bar{\mathbf{x}}, \bar{\mathbf{p}} \rangle + u\langle \bar{\mathbf{p}}, \bar{\mathbf{p}} \rangle$$

$$|N|t = |N|\langle \mathbf{w}_0, \bar{\mathbf{n}} \rangle = \langle u\bar{\mathbf{p}} + v\bar{\mathbf{x}}, |X|\bar{\mathbf{x}} - |P|\bar{\mathbf{p}} \rangle.$$

This is solved by

$$u = \frac{\langle \bar{\mathbf{x}}, \bar{\mathbf{p}} \rangle(t + (s - t)|P|/|X|) - s\langle \bar{\mathbf{x}}, \bar{\mathbf{x}} \rangle}{\langle \bar{\mathbf{p}}, \bar{\mathbf{x}} \rangle^2 - \langle \bar{\mathbf{p}}, \bar{\mathbf{p}} \rangle\langle \bar{\mathbf{x}}, \bar{\mathbf{x}} \rangle} \tag{23}$$

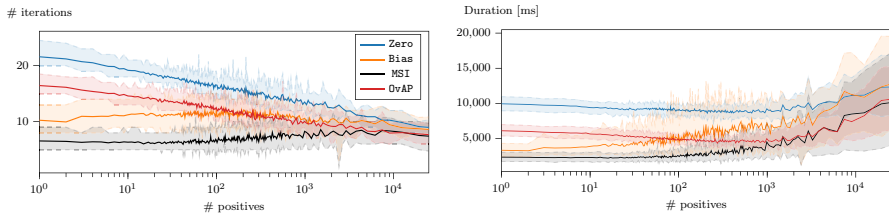
$$v = (s - u\langle \bar{\mathbf{p}}, \bar{\mathbf{p}} \rangle) / \langle \bar{\mathbf{x}}, \bar{\mathbf{p}} \rangle. \tag{24}$$

A short discussion on the situations if the denominators become zero is given in appendix 7.3.

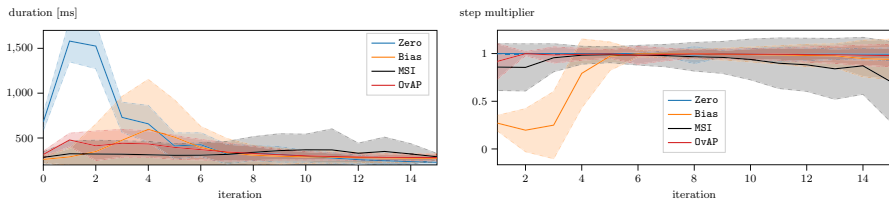
As we want the loss vector corresponding to the initial weights to be sparse, we want the distance of  $\bar{\mathbf{n}}$  to the decision boundary to be larger than that of  $\bar{\mathbf{p}}$ . On the other hand, it needs to be small enough that the resulting weight vector is not too far from the optimal weights. Empirically, we found  $s = 1, t = -2$  (cf. appendix 7.3) to work well.

*Evaluation:* Does this initialization method work as we expect? As shown in Fig. 6, the method induces loss vectors that are more sparse than the ones for OvAP, and keeps this property over the course of training, resulting in faster update times for each iteration. As expected, tail labels benefit most, because the situation depicted in Fig. 5 is better approximated in the tail than for head labels. Because most labels in XMC are tail labels, training time improvements for them are most important for overall speed-up. Still, our method is faster or at least as fast as any of the other methods across the entire range of number of positives.

Figure 7 shows that using the MSI initialization drastically reduces the number of iterations needed for convergence for tail labels. Whereas for Zero and OvAP initialization the number of iterations increases as the number of positive instances decreases, for Bias and MSI initialization it remains almost constant. This cannot be explained by looking only at the distance the optimization algorithm has to travel, neither in terms of the initial loss value, nor in terms of the distance between initial and final weight vector. The corresponding graphs are presented in appendix 7.3.



**Fig. 7** Number of Newton optimization steps required for convergence (left) and training time for a single label (right). The average and the 95% quantiles are plotted as a function of the number of positive instances



**Fig. 8** Duration per iteration (left), and line search step multiplier (right). The MSI initialization manages to have the early iterations as fast as the later ones, while still working much better in terms of the quadratic approximation than the `Bias` initialization. The decrease of the step multiplier, and the slight increase in duration, for late iterations is not relevant, because as Fig. 7 shows most of the binary problems will have finished by iteration 10

Until about iteration 8, the duration per iteration with MSI is faster than the other initialization methods, as depicted in Fig. 8. Because MSI very rarely needs more than 10 iterations of training, and on average if finished before 8 iterations (Fig. 7), the slight speed advantage of the other methods for later iterations only applies to very few binary sub-problems, and such has little effect on the overall training time.

## 5 Benchmarks

In this section, we demonstrate the effectiveness of MSI initialization on a wide range of datasets and settings. We first provide timings for training with sparse tf-idf data on a single, large compute node, the setting in which we expect linear OVA models to be run typically. We then show how changes in the training data affect the speed-up of our method: In particular, we present timing for the case of training with a dense input representation, and timings for training with the logistic loss, which does not admit the implicit negative-mining effect. Finally, to allow for a meaningful comparison with the method by Fang et al. (2019), we also run training on smaller hardware.

*Infrastructure and training configuration:* Our code is a re-implementation of the conjugate-gradient Newton procedure of recent versions of Liblinear (Galli et al., 2021). The hyperparameters related to stopping condition for conjugate-gradient iterations ( $\epsilon = 0.5$ ), preconditioner ( $\alpha = 0.01$ ), back-tracking line search ( $\alpha = 0.5$ ,  $\eta = 0.01$ ,  $\text{max\_steps} = 20$ ), and stopping condition for the optimization ( $\epsilon_0 = 0.01$ ) have been taken from their code. To reduce model size, all weights below a threshold of 0.01 have been clipped to zero.

Our main experiments were run on a single compute node equipped with two AMD Rome 7H12 CPUs. The code is NUMA-aware and produces a copy of the training data for each NUMA-node, but memory bandwidth still is the limiting factor in scalability. We observed very good scaling for 64 cores, but sublinear improvements for more cores.

All nodes in the compute cluster have identical hardware and jobs were run with exclusive access to the node. In the absence of any other user-jobs running on the same node, and hence no significant randomness<sup>4</sup> in the run times, we have not systematically collected data from multiple runs of the same setting.

*Sparse data:* We tested the initialization methods on Eurllex-3k (Mencia & Loza, 2008), AmazonCat-13k (McAuley & Jure, 2013), WikiSeeAlsoTitles-350K, WikiTitles-500k, Amazon-670k (McAuley & Jure, 2013), WikiLSHTC-325k (Partalas et al., 2015), Amazon-3M (McAuley et al., 2015, 2015), and Delicious-200k (Wetzker et al., 2008) in their train/test splits as available from Bhatia et al. (2016). The runtime results are shown in the Tabel 1 under the columns Zero, Bias, OvAP, and MSI.

As shown in the Ratio column, the proposed method (MSI) is 3–4 times faster than Zero initialization, and also significantly faster than Bias and OvAP initialization. For Amazon-3M it is only possible to run the proposed MSI initializer due to the large computational cost involved for the Zero initialization baseline. Therefore, the relative speed-up cannot be reported. However, our training of 1360 minutes (< 1 day), using MSI as a starting point of the optimization process, shows that learning a DiSMEC-*style* (Babbar & Schölkopf, 2017) model is feasible even for datasets with millions of labels, *on a single node*.

The Zero setting corresponds to the vanilla DiSMEC algorithm, under our new and faster implementation. Due to improvements in implementation quality, in particular the NUMA-awareness of our code, our timings with this initialization are already much faster than running original DiSMEC on the same node.

Because the optimization problem is convex, the minimum is not affected by the choice of initial parameter. However, because in practice only an approximate minimum is found, there are slight variations (around 0.1%) in precision@k metrics for the different methods. The exact numbers can be found in Table 4.

*Dense data:* We also ran the tests on the same dataset of dense features as used for slice<sup>5</sup>. The default setting for the stopping condition as specified by Liblinear,  $\epsilon_0 = 0.01$ , is far too strict in this setting. Therefore, we increased this parameter to  $\epsilon_0 = 1$  here. For these settings, we observed larger variability in the classification accuracy, and a slightly reduced, but still significant speed-up.

*Logistic loss:* We also ran a test where we replaced the squared hinge-loss with the logistic loss. As the logistic loss only vanishes asymptotically, this is a setting that does not benefit from implicit negative-mining. This means that later iterations will take approximately as long as earlier ones, and there is less benefit from being able to skip the first iterations. The result is a much reduced benefit from our proposed initialization, as can be seen in Table 2.

Due to the vast number of negatives (for a tail label), the loss function's minimum will not be achieved when the negatives are classified correctly with margin one as is the case with squared-hinge, but will in fact train a larger margin for the negatives. This suggests that by decreasing the  $t$  parameter further, our initial guess for the separating hyperplane will be closer to the final weight vector. Thus we have used  $t = -3$ .

<sup>4</sup> e.g. multiple runs of Delicious-200k with zero initialization fluctuate between 28236 and 28483 seconds.

<sup>5</sup> <http://manikvarma.org/code/Slice/download.html>

**Table 1** Training time (in minutes) for different datasets and initialization methods

Dataset	Setting	Zero	Bias	OvAP	MSI	Ratio
Eurlex-4k	tf-idf	0.1	0.1	0.1	0.1	1.8
Wiki10-31k	tf-idf	3.2	3.1	2.6	1.9	1.6
Amazoncat-13k	tf-idf	19.7	11.5	12.6	5.6	3.5
Amazoncat-14k	tf-idf	78.8	34.0	48.0	23.2	3.4
WikiTitles-500k	tf-idf	320.6	187.8	301.8	71.1	4.5
Amazon-670k	tf-idf	409.1	337.3	259.4	81.3	5.0
Delicious-200k	tf-idf	470.6	416.2	337.5	283.8	1.7
WikiLSHTC-350k	tf-idf	1,111.0	479.5	634.8	310.5	3.6
Amazon-3M	tf-idf	–	–	–	1360.1	–
Eurlex-4k	dense	0.1	0.1	0.1	0.0	2.7
Amazon-670k	dense	782.4	376.5	489.1	321.2	2.4

The `Ratio` column denotes the speed-up of `MSI`-initialization over `Zero`-initialization under our implementation. Due to the sub-linear scaling beyond 64 cores, the reported times here cannot be directly converted to single-core timings by multiplying with 128

*Comparison with OvAP++:* Finally, in order to allow for a meaningful comparison with the code by Fang et al. (2019), we also ran trainings with the smaller datasets on a Xeon Gold 6148 processor, with both our own implementation and with their code. For these experiments, instead of taking the dataset directly from the XMC repository (Bhatia et al., 2016), we applied their preprocessing script. Because the two methods still use slightly different optimization procedures (theirs uses a trust-region instead of a line search, as in older Liblinear versions) and implementations, we still need compare the relative speed-ups between the different initialization methods. The results can be seen in Table 3. `MSI` initialization consistently leads to more speed-up than `OvAP++` initialization.

## 6 Discussion

We have provided a way to initialize the weights of a linear OvA extreme classifier in such a way as to reduce training times. Our experiments show that aside from the initial loss value investigated in Fang et al. (2019), the implicit sparsity and local smoothness properties of the loss landscape also play an important role in the success of the method.

*Limitations:* The initialization method discussed in this work is mainly applicable to linear One-versus-rest XMC algorithms. This rules out label-embedding schemes (Guo et al., 2019; Bhatia et al., 2015), decision-tree based classifiers (Prabhu & Manik, 2014; Majzoubi & Anna, 2020) and deep-learning methods in which the classifier is jointly learnt with the intermediate representations (You et al., 2019).

*Outlook:* The choice of initial vector is an underexplored design tool in XMC that is orthogonal to many other design choices such as explicit negative-mining (Reddi et al., 2019; Jain et al., 2019), training meta classifiers over buckets of labels (Medini et al., 2019; Dahiya et al., 2021), or the choice of regularizer. Future work should thus look into combining these, for example integrating our initialization into the OvA parts of Slice (Jain et al., 2019), Parabel (Prabhu et al., 2018) XR-Transformer (Zhang et al., 2021), Probabilistic Label Trees (Wydmuch et al., 2018), or Astec (Dahiya et al., 2021).

**Table 2** Training time (in minutes) for different datasets and initialization methods when minimizing the logistic loss

Dataset	Zero	Bias	OvAP	MSI	Ratio
Eurlex-4k	0.2	0.2	0.1	0.2	1.2
Wiki10-31k	4.1	3.4	2.4	3.5	1.2
Amazoncat-13k	35.6	28.2	24.8	28.2	1.4
Amazoncat-14k	160.1	127.6	111.3	125.3	1.3
WikiTitles-500k	366.8	311.0	270.4	315.7	1.2
Amazon-670k	633.6	522.8	420.4	525.5	1.2

**Table 3** Training time (in minutes) and speed-up on smaller hardware comparing OvAP++ and MSI

Dataset	Zero	OvAP++	Ratio	Zero	MSI	Ratio
Eurlex-4k	0.3	0.2	1.7	0.3	0.1	2.5
Wiki10-31k	9.7	5.7	1.7	7.3	3.1	2.4
Amazoncat-13k	61.7	30.6	2.0	39.7	10.5	3.8
Amazoncat-14k	208.0	94.8	2.2	145.7	39.2	3.7
WikiTitles-500k	521.9	236.4	2.2	535.2	114.9	4.7
Amazon-670k	1310.4	418.1	3.1	708.2	172.1	4.1

## Appendix

### Information on AmazonCat data

To evaluate the initialization methods and investigate their properties, we took the AmazonCat-13K dataset from the extreme classification repository Bhatia et al. (2016); McAuley and Jure (2013) as a running example during sects. 3 and 4. In this dataset the input features are sparse tf-idf values of a bag-of-words representation augmented by an additional bias feature that is set to 1. The train/test split is taken from the repository. The dataset consists of 1 186 239 training instances and 306 782 test instances in a 203 882 dimensional feature space. These instances are mapped to 13 330 labels, with an average of 5 labels per instance.

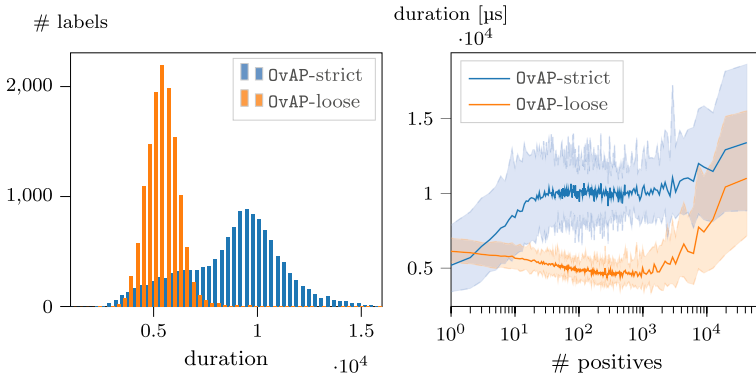
### Early stopping for OVA-primal

For the squared-hinge loss, a vector that minimizes (15) perfectly will have similar characteristics to the Bias initialization  $\mathbf{w}_b$ , in that for many instances the prediction will be close to the hinge location. Thus the starting point will be in a location unfavourable to the Newton optimizer. A remedy is to stop training of  $\mathbf{w}^*(-\mathbf{1})$  early. This results in larger initial loss, but makes the training progresses more smoothly.

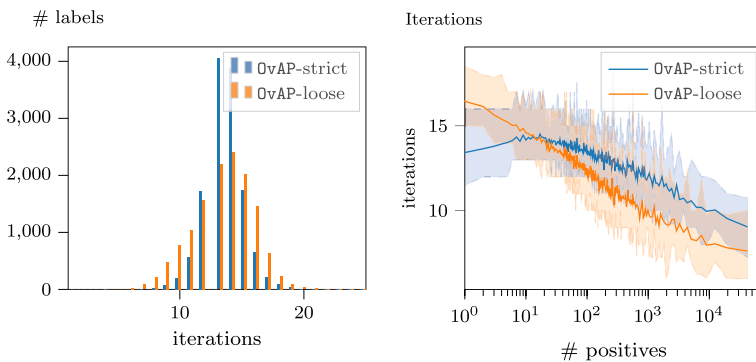
Since the label vector  $-\mathbf{1}$  is maximally imbalanced, according to equation (13) the weight should be trained to high precision  $\epsilon = \epsilon_0/n$ . If instead we use the much less strict  $\epsilon = \epsilon_0$ , the initial loss is much larger, but at the same time the training progresses more smoothly.

The resulting training durations are shown in Fig. 9. Overall, the loose stopping criterion performs much better, except for labels with less than three positives. This is because for these





**Fig. 9** Histogram (left) and average with 95% confidence interval in dependence of the number of positive instance (right) of the training time, measured in milliseconds. Overall, the `loose` training configuration is much faster, though for extreme tail labels with less than 3 positive instances, `strict` actually performs slightly better



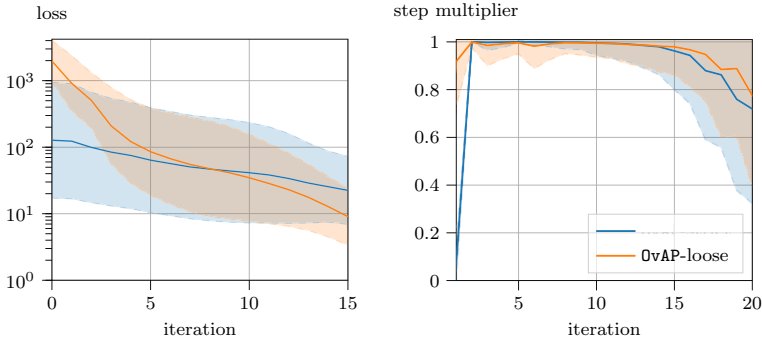
**Fig. 10** Histogram (left) and average with 95% confidence interval in dependence of the number of positive instance (right) of the number of weight updates until convergence

labels, the initial vector is very close to the final vector, and thus the number of iterations that are required for training decreases, cf. Fig. 10. This is in contrast to the bias = -1 setting, which has similar characteristics in the second-order optimization, but does not benefit from fewer iterations and consequently does not show a decrease of overall training time for the extreme tail.

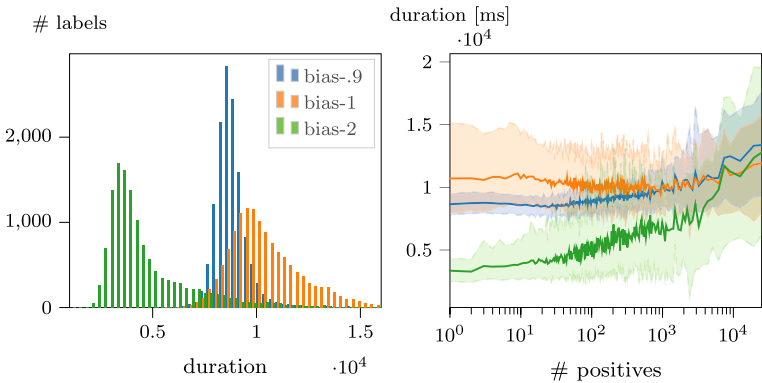
In Fig. 11 (left), the training progress over the iterations is shown. Here we can see that, similar to Bias initialization, the `strict` setting results in very little progress in the first iteration, and overall much slower progress also for the other iterations, such that `loose` turns out to be the faster setting, even though its initial loss is an order of magnitude larger. The right side of this figure shows the step multiplier during the line-search phase, which shows that during the first iteration, for `strict` the second-order approximation is overly optimistic and the real step size has to be much smaller than the CG procedure suggests.

### Additional Results for Bias Initialization

Some additional graphs showing details about the training process with Bias initialization. In Fig. 12 the training duration of different Bias variations is shown in terms of



**Fig. 11** Letting the OvA-Primal initial vector be the result of training the all-zero-label problem until convergence (strict) results in much worse performance than doing early stopping (loose). We suspect that this is because strict training brings the weight vectors into a regime that is qualitatively similar to the bias = -1 setting: All training instances will predict a negative label with a margin very close to 1. Even though this results in much lower initial loss (left), it also leads to a useless first iteration in which little progress is made (right)



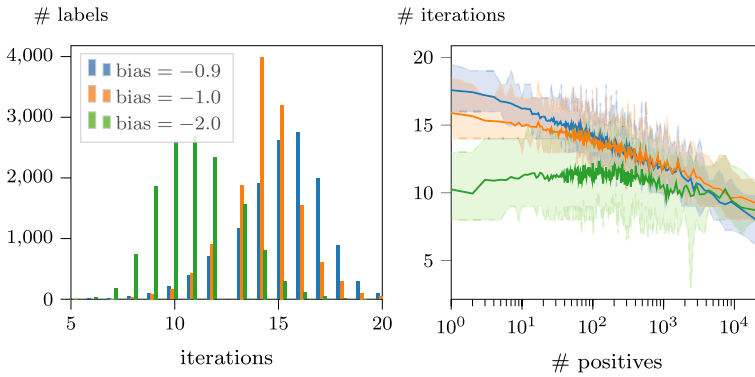
**Fig. 12** Total duration of the individual binary problems when trained with different Bias-init variations. The histogram on the left has bins corresponding to the number of binary problems taking a certain amount of time, on the right the average time and 95% quantiles are shown in dependence of the number of positive instances. Duration is measured in milliseconds. The graphs show that bias = -1 is a local maximum in terms of computation time

training time for each individual binary problem. For bias = -1, the binary problems take roughly the same amount of time independent of the number of positives, but for bias = -2 the tail labels can be trained much faster.

The training duration as measured in number of iterations is presented in Fig. 13.

### Additional results for mean-separating initialization

In Fig. 14 (left) we show how the initial loss depends on the number of positive instances. For Bias and MSI, we can see that tail labels result in very low initial loss, whereas for head labels MSI gives slightly larger loss than Bias and OvAP. On the other hand, OvAP’s loss remains significantly larger for tail labels. The right side of Fig. 14 shows how that initial loss translates to initial distance in weight space. Here,



**Fig. 13** Number of Newton optimization steps required for convergence. On the left, the data is shown as a histogram, with the bins corresponding to the number of binary problems that required the corresponding number of steps. On the right, the average number of steps, and the 95% quantiles, are plotted as a function of the number of positive instances. The data shows that the benefit of the bias = -2 initialization is not limited purely to faster iterations because of increased sparsity, but it also needs fewer steps, despite having slightly larger initial loss than bias = -1. This indicates that each step has to make more progress, i.e. the loss landscape is more benign to the Newton optimization around the bias = -2 trajectory. The right-hand graph also shows that most of the benefit comes from the tail labels

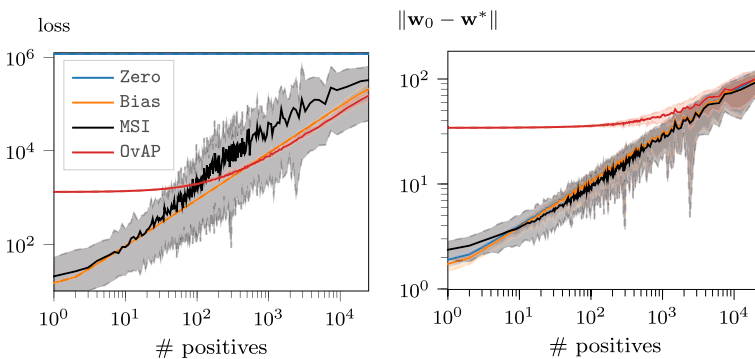
we can see that for tail labels,  $OvAP$  's initial weight vector is an order of magnitude further from the optimum than for the other methods.

*What happens if the denominator of MSI is zero* The expression in equation (23) diverges if  $\bar{\mathbf{p}}$  and  $\bar{\mathbf{x}}$  are linearly dependent, which implies that the means of features for negatives and positives only differs by a factor. In that case, we cannot (except in unlikely special cases) find an initial vector that fulfills our condition, so we return the zero vector.

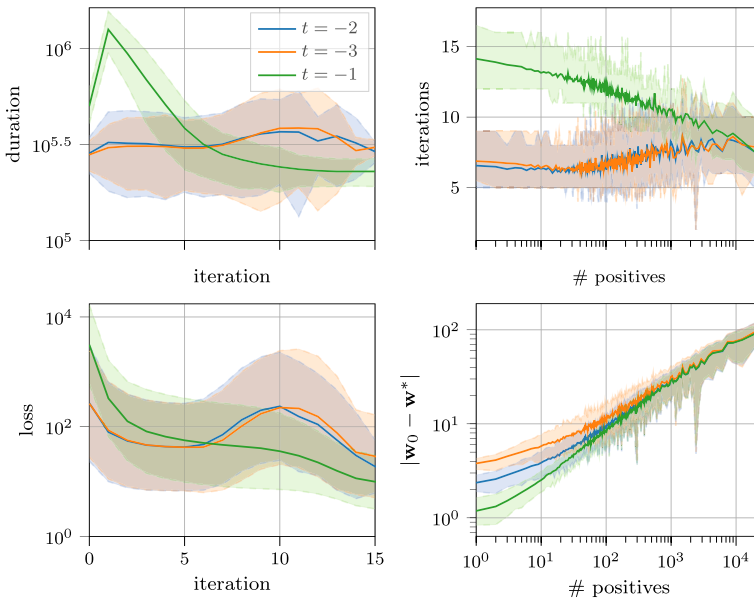
A second numerical instability occurs if  $|\langle \bar{\mathbf{x}}, \bar{\mathbf{p}} \rangle| \ll 1$ , during the calculation of  $v$ . To investigate this setting, we substitute  $\langle \bar{\mathbf{x}}, \bar{\mathbf{p}} \rangle$  with 0 in (23) and (24).

$$s = u \langle \bar{\mathbf{p}}, \bar{\mathbf{p}} \rangle \tag{25}$$

$$|N|t = v|X|\langle \bar{\mathbf{x}}, \bar{\mathbf{x}} \rangle - u|P|\langle \bar{\mathbf{p}}, \bar{\mathbf{p}} \rangle \tag{26}$$



**Fig. 14** Loss of the initial weight vector (left) and distance between initial and final vector (right). The shaded area shows 95% quantiles. For all except  $Zero$  initialization, loss and distance are much lower for tail labels than for head labels



**Fig. 15** Average duration of the iterations (top left, in  $\mu$  s), and number of iterations for different numbers of positive instances (top right). The bottom left graph shows the development of the objective function. At first glance, this seems paradoxical, as the algorithm is designed so that for each binary problem, the loss strictly decreases over the iterations. The apparent increase is caused by the fact that the sample of binary problems shrinks for later iterations, as the labels for which the loss is already low before will terminate the optimization. Finally, the bottom right graph shows the Euclidian distance between the initial weight vector and the approximately optimal weight vector for which the minimization is terminated. The shaded area marks the 95% quantile

This leads to

$$u = \frac{s}{\langle \bar{\mathbf{p}}, \bar{\mathbf{p}} \rangle}, \quad v = \frac{|\mathbf{N}|t + |\mathbf{P}|s}{|\mathbf{X}| \langle \bar{\mathbf{x}}, \bar{\mathbf{x}} \rangle}. \tag{27}$$

*Choice of hyperparameters for MSI* As argued in the main text, in order to achieve sparsity in the Hessian calculations, having the average of the negative instances be classified very strongly as a negative would be beneficial. Interestingly, Fig. 15 shows not only the expected speedup in per-iteration time, but also that  $t = -1$  requires significantly more iterations than the other two settings.

This is not too surprising, given that we expect that the large number of negative instances fill a larger proportion of the space than the few positives, so the separating hyperplane should be closer to the mean of the positives. One reason for this could be that  $t = -1$  induces a higher initial loss than the other two settings, as shown in the bottom left graph. However, if we measure the distance that the optimization procedure covers, i.e. the distance between the initial and final weight vector, it turns out that  $t = -1$  actually starts closest.

**Full result table**

Since the actual loss function is based on binary classification, we have also calculated the precision and recall as averaged over the individual binary problems. The metrics remain

**Table 4** Durations (as given in main paper) and precision@{1,3,5} for training different datasets with the given initialization methods

Dataset	Setting	Method	P@1	P@3	P@5	Prec	Rec	Duration
Eurlex-4k	tf-idf	Zero	83.8	70.8	59.1	79.2	38.5	0.15
Eurlex-4k	tf-idf	Bias	83.7	70.8	59.1	79.2	38.4	0.13
Eurlex-4k	tf-idf	MSI	83.7	70.8	59.0	79.2	38.4	0.08
Eurlex-4k	tf-idf	OvAP	84.0	70.7	59.1	79.2	38.4	0.10
Wiki10-31k	tf-idf	Zero	84.1	74.7	65.9	75.8	13.0	3.20
Wiki10-31k	tf-idf	Bias	84.1	74.6	65.9	75.7	13.0	3.10
Wiki10-31k	tf-idf	MSI	84.3	74.7	65.9	75.8	13.0	1.95
Wiki10-31k	tf-idf	OvAP	84.1	74.7	65.9	75.7	13.0	2.60
Amazoncat-13k	tf-idf	Zero	93.2	78.9	64.2	84.9	58.2	19.75
Amazoncat-13k	tf-idf	Bias	93.2	78.9	64.2	84.9	58.1	11.50
Amazoncat-13k	tf-idf	MSI	93.0	78.9	64.1	84.9	58.0	5.60
Amazoncat-13k	tf-idf	OvAP	93.2	78.9	64.2	85.0	58.1	12.58
Amazoncat-14k	tf-idf	Zero	89.7	69.5	54.9	87.3	66.1	78.81
Amazoncat-14k	tf-idf	Bias	89.6	69.5	54.9	87.3	66.0	34.02
Amazoncat-14k	tf-idf	MSI	89.5	69.5	54.9	87.3	65.8	23.25
Amazoncat-14k	tf-idf	OvAP	89.7	69.5	54.9	87.3	66.2	48.03
Amazon-670k	tf-idf	Zero	45.7	40.7	37.2	82.0	17.6	409.05
Amazon-670k	tf-idf	Bias	45.7	40.7	37.2	82.0	17.6	337.30
Amazon-670k	tf-idf	MSI	45.7	40.7	37.2	82.0	17.6	81.31
Amazon-670k	tf-idf	OvAP	45.6	40.7	37.2	82.0	17.6	259.42
WikiLSHTC	tf-idf	Zero	64.4	42.6	31.7	84.7	19.2	931.58
WikiLSHTC	tf-idf	Bias	64.4	42.6	31.6	84.7	19.2	803.45
WikiLSHTC	tf-idf	MSI	64.4	42.7	31.7	84.6	19.2	286.79
WikiLSHTC	tf-idf	OvAP	64.4	42.6	31.7	84.7	19.2	634.76
Delicious-200k	tf-idf	Zero	46.4	39.3	35.7	29.2	1.7	470.58
Delicious-200k	tf-idf	Bias	46.4	39.3	35.7	29.5	1.7	416.20
Delicious-200k	tf-idf	MSI	46.2	39.1	35.6	29.0	1.7	283.82
Delicious-200k	tf-idf	OvAP	46.4	39.3	46.4	29.2	1.7	337.47
WikiTitles-500k	tf-idf	Zero	40.1	21.4	15.0	68.0	5.6	320.59
WikiTitles-500k	tf-idf	Bias	40.1	21.4	15.0	68.2	5.6	187.78
WikiTitles-500k	tf-idf	MSI	40.1	21.4	15.0	68.2	5.6	71.13
WikiTitles-500k	tf-idf	OvAP	40.1	21.4	15.0	68.1	5.6	301.80
Amazon-670k	dense	Zero	35.6	32.1	29.4	74.6	23.0	782.39
Amazon-670k	dense	Bias	34.2	31.0	28.5	69.4	22.4	376.02
Amazon-670k	dense	MSI	35.6	32.1	29.5	73.5	23.4	321.19
Amazon-670k	dense	OvAP	35.6	32.0	29.4	74.5	23.0	489.15
Eurlex-4k	dense	Zero	77.2	63.4	51.3	72.8	36.3	0.13
Eurlex-4k	dense	Bias	75.3	62.1	50.2	76.0	26.4	0.07
Eurlex-4k	dense	MSI	76.6	63.0	51.4	67.7	39.1	0.05
Eurlex-4k	dense	OvAP	76.9	63.5	51.4	72.5	36.4	0.07
Amazon-3M	tf-idf	MSI	47.8	45.0	42.9	67.1	14.0	1,360.10

mostly unchanged for tf-idf data, but there is significant fluctuation when `slice` features are used (Table 4).

**Author Contributions** ES: Idea, Implementation, Writing RB: Supervision, Writing – Review & Editing, Original DiSMEC implementation

**Funding** Open Access funding provided by Aalto University. Academy of Finland (347707)

**Data availability** <http://manikvarma.org/downloads/XC/XMLRepository.html>

## Declarations

**Conflict of interest** \*.aalto.fi \*.poznan.pl

**Consent to participate** Not Applicable

**Consent for publication** Not Applicable

**Ethical approval** Not Applicable

**Code** version.aalto.fi/gitlab/xmc/dismecpp

**Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

## References

- Adamic, L. A., & Bernardo, A. H. (2002). Zipf's law and the Internet. *Glottometrics*, 3(1), 143–150.
- Agrawal, R., et al., (2013). "Multi-Label Learning with Millions of Labels: Recommending Advertiser Bid Phrases for Web Pages". In: Proceedings of the 22nd International Conference on World Wide Web. WWW '13. Rio de Janeiro, Brazil: Association for Computing Machinery, pp. 13–24.
- Babbar, R., & Schölkopf, B. (2017). DiSMEC: Distributed sparse machines for extreme multi-label classification. In Proceedings of the Tenth ACM International Conference on Web Search and Data Mining. WSDM 17. Cambridge, United Kingdom: Association for Computing Machinery, pp. 721–729.
- Babbar, R., & Schölkopf, B. (2019). Data scarcity, robustness and extreme multi-label classification. *Machine Learning*, 108(8), 1329–1351.
- Bhatia, K., et al. (2016). The extreme classification repository: Multi-label datasets and code. url: <http://manikvarma.org/downloads/XC/XMLRepository.html>.
- Bhatia, K., et al. (2015). Sparse local embeddings for extreme multi-label classification. *NIPS*, 29, 730–738.
- Chang, W.-C. et al. (2020). Taming pretrained transformers for extreme multilabel text classification". In: KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23–27, 2020. ACM, pp. 3163–3171.
- Dahiya, K. et al. (2021). DeepXML: A deep extreme multi-label learning framework applied to short text documents. In Proceedings of the 14th ACM International Conference on Web Search and Data Mining, pp. 31–39.
- Dekel, O., Ohad, S. (2010). Multiclass-multilabel classification with more classes than examples. In Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics. Ed. by Yee, W. T., and Mike, T. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, pp. 137–144.
- Deng, J. et al. (2010). What does classifying more than 10,000 image categories tell us? In: ECCV.

- Duchi, J., Elad, H., & Yoram, S. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61), 2121–2159.
- Fan, R.-E., et al. (2008). LIBLINEAR: A library for large linear classification. *The Journal of machine Learning research*, 9, 1871–1874.
- Fang, H. et al. (2019). Fast training for large-scale one-versus-all linear classifiers using tree-structured initialization. In Proceedings of the 2019 SIAM International Conference on Data Mining. SIAM, pp. 280–288.
- Galli, Leonardo, Chih-Jen, Lin (2021). A study on truncated newton methods for linear classification. In IEEE Transactions on Neural Networks and Learning Systems.
- Glorot, X., Yoshua, B. (2010). Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the thirteenth international conference on artificial intelligence and statistics. JMLR Workshop and Conference Proceedings, pp. 249–256.
- Goyal, P. et al. (2018). Accurate, large minibatch SGD: Training ImageNet in 1 Hour. [arXiv: 1706.02677](https://arxiv.org/abs/1706.02677) [cs.CV].
- Guo, C. et al. (2019). Breaking the glass ceiling for embedding-based classifiers for large output spaces. Advances in Neural Information Processing Systems. In Wallach H. et al. (Eds.), Vol. 32. Curran Associates, Inc.
- Jain, H. et al. (2019). Slice: Scalable linear extreme classifiers trained on 100 million labels for related searches. In WSDM, pp. 528–536.
- Keerthi, S., Sathiy, D., DeCoste, T. J. (2005). A modified finite Newton method for fast solution of large scale linear SVMs. *Journal of Machine Learning Research* 6(3).
- Khandagale, S., Han, X., & Rohit, B. (2020). Bonsai: diverse and shallow trees for extreme multi-label classification. *Machine Learning*, 109(11), 2099–2119.
- Liu, J., et al. (2017). Deep learning for extreme multi-label text classification. In Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval.
- Majzoubi, M., Anna, C. (2020). Ldsm: Logarithm-depth streaming multi-label decision trees. In International Conference on Artificial Intelligence and Statistics. PMLR, pp. 4247–4257.
- McAuley, J., Jure, L. (2013). Hidden factors and hidden topics: understanding rating dimensions with review text. In Proceedings of the 7th ACM conference on Recommender systems, pp. 165–172.
- McAuley, J., Rahul, P., Jure, L. (2015). Inferring networks of substitutable and complementary products. In Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining, pp. 785–794.
- McAuley, J., et al. (2015). Image-based recommendations on styles and substitutes. In Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval, pp. 43–52.
- Medini, T., et al. (2019). Extreme classification in log memory using count-min sketch: A case study of amazon search with 50m products. [arXiv preprint arXiv:1910.13830](https://arxiv.org/abs/1910.13830).
- Mencia, E., Loza, J. F. (2008). Efficient pairwise multilabel classification for large-scale problems in the legal domain. In Joint European Conference on Machine Learning and Knowledge Discovery in Databases. Springer, pp. 50–65.
- Mikolov, T., et al. (2013). Efficient estimation of word representations in vector space. [arXiv preprint arXiv:1301.3781](https://arxiv.org/abs/1301.3781).
- Mnih, A., Geoffrey, E. H. (2009). A Scalable Hierarchical Distributed Language Model. In Advances in Neural Information Processing Systems. Koller D. et al. (Eds.), Vol. 21. Curran Associates, Inc.
- Partalas, I., et al. (2015). Lshstc: A benchmark for large-scale text classification. [arXiv preprint arXiv:1503.08581](https://arxiv.org/abs/1503.08581).
- Prabhu, Y., et al. (2018). Parabel: Partitioned label trees for extreme classification with application to dynamic search advertising. In Proceedings of the 2018 World Wide Web Conference, pp. 993–1002.
- Prabhu, Y., Manik, V. (2014). Fastxml: A fast, accurate and stable tree-classifier for extreme multi-label learning. In Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 263–272.
- Qaraei, M., et al. (2021). Convex surrogates for unbiased loss functions in extreme classification with missing labels. In Proceedings of The Web Conference 2021. WWW '21. Ljubljana, Slovenia: Association for Computing Machinery.
- Reddi, S. J., et al. (2019). Stochastic negative mining for learning with large output spaces. In The 22nd International Conference on Artificial Intelligence and Statistics. PMLR, pp. 1940–1949.
- Ruder, S. (2017). An overview of gradient descent optimization algorithms. [arXiv: 1609.04747](https://arxiv.org/abs/1609.04747) [cs.LG].
- Shalev-Shwartz, S., Ben-David, S., (2014). Understanding machine learning: From theory to algorithms. Cambridge university press.

- Smith, L. N., (2017). Cyclical learning rates for training neural networks. In 2017 IEEE winter conference on applications of computer vision (WACV). IEEE, pp. 464–472.
- Wetzker, R., Carsten, Z., Christian, B. (2008). Analyzing social bookmarking systems : A del.icio.us Cookbook.
- Wydmuch, M., et al. (2018). A no-regret generalization of hierarchical softmax to extreme multi-label classification. In Advances in Neural Information Processing Systems 31. Bengio S. et al. (Eds.), Curran Associates, Inc., pp. 6355–6366.
- Yen, I. E. H., et al. (2017). Ppdsparse: A parallel primal-dual sparse method for extreme classification. In Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 545–553.
- You, R. et al. (2019). AttentionXML: Label tree-based attention-aware deep model for high-performance extreme multi-label text classification. In Advances in Neural Information Processing Systems. Wallach H. et al. (Eds.), Vol. 32. Curran Associates, Inc.
- Zhang, J. et al. (2021). Fast multi-resolution transformer fine-tuning for extreme multi-label text classification. In Advances in Neural Information Processing Systems 34.

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.