

# Large-Scale Many-Class Prediction via Flat Techniques

Omid Madani<sup>1</sup> and Jian Huang<sup>2</sup>

<sup>1</sup> SRI International, AI Center, Menlo Park, CA 94025, USA [madani@ai.sri.com](mailto:madani@ai.sri.com)

<sup>2</sup> College of Information Sciences and Technology  
Pennsylvania State University, State College PA 16802, USA [jhuang@ist.psu.edu](mailto:jhuang@ist.psu.edu)

**Abstract.** Prediction problems with huge numbers of classes are becoming more common. While class taxonomies are available in certain cases, we have observed that simple *flat* learning and classification, via *index* learning and related techniques, offers significant efficiency and accuracy advantages. In the PASCAL challenge on large-scale hierarchical text classification, the accuracies we obtained ranked in the top three in all evaluations. We also found that using committees of a few learned models boosted accuracy, and observed a tradeoff between accuracy versus memory and time efficiency. This paper is a short report on our approach.

## 1 Introduction

Many prediction tasks, in a variety of domains including text, speech, and images, can be viewed as problems with a huge number of classes. We have been developing techniques for efficient learning of compact discriminative models, or *indices*, for such many-class tasks (e.g., [3, 4]). An index is a sparse weighted mapping connecting each feature to relatively few classes (a sparse weight matrix). We have discovered that index learning has a number of efficiency and accuracy advantages over one-versus-rest and hierarchical learning with linear (two-class) Support Vector Machines (SVMs) [3]. We also expect that there remains significant room for improving the efficiency-accuracy curve for large-scale data sets involving tens of thousands of classes. The five-month-long Large-Scale Hierarchical Text Classification PASCAL challenge provided a great opportunity for comparisons and for further development of our techniques.

In the challenge we were given a large multiclass problem, obtained from crawling ODP (the Open Directory Project), involving just over 12,000 classes and about 120,000 training instances and 35,000 test instances, all singly labeled (one class per instance), in each of four tasks differing on choice of features. The challenge could be approached with a variety of techniques such as nearest neighbors and SVMs. The test vectors were made available (without the labels), so the problem can be viewed as a case of semi-supervised learning and transduction. The internal classes that together with the labels form a taxonomy were also given, and can potentially be used for accuracy or efficiency gains. We focused on the simpler problem of the inductive learning of a linear weight matrix (or an index), disregarding the taxonomy information and the internal classes. Thus our methods are “flat” rather than hierarchical. The index, given an instance, ranks the classes by the scores they obtain. The goal of such flat learning is, roughly, to obtain a good ranking of the true label for each instance. We experimented with several flat techniques, in particular an OOZ (“ooze”) variant [4] and the Passive-Aggressive (PA) algorithm [1]. Both approaches obtained competitive accuracies, although we found that we needed to relax the efficiency constraints on OOZ. Furthermore, we discovered that committees of the learned models boosted accuracy. With small committees of PA-learned models, our accuracies ranked first in 5 of 12 evaluation tracks among 19 participants, and otherwise ranked in the top 3. We next describe the algorithms briefly. We then present some of our findings.

## 2 Algorithms

Both OOO (or Online Optimization of Z [or slack] variables) and PA algorithms keep and update a weight matrix  $W$ , where the features correspond to rows, and classes to columns. Classification amounts to first scoring the classes. The scores are obtained via the dot product  $x^T W$ , where  $x$  denotes the column vector representation of the instance (in our experiments: l2-normalized tfidf representation). The highest-scoring class (breaking ties arbitrarily),  $\operatorname{argmax}_c x^T W$ , is the predicted class. OOO assumes nonnegative feature values. Both learning algorithms begin with a zero weight matrix. We implement the matrix as a sparse index, *i.e.*, for each feature  $f$ , we keep a list of nonzero weights or connections (or edges) of  $f$  (corresponding to nonzero weights in row  $f$  of the matrix). At the outset, every feature’s list is empty (the zero matrix). We refer to the number of connections of  $f$  as the outdegree. Both algorithms require several passes over the training data to attain maximum accuracy. On each pass, we randomly permuted the training instances.

### 2.1 The OOO Update

After scoring, let  $s_{c_x}$  denote the score of the true class  $c_x$ . OOO (Figure 1) updates whenever  $\exists c' \neq c, s_{c'} \geq s_{c_x} - \delta$ , where  $c'$  denotes another class, and  $\delta$  is a desired margin threshold ( $\delta = 0.005$  in reported experiments). For such instances, we refer to those classes  $c'$  with  $s_{c'} \geq s_{c_x} - \delta$  as the (offending) negative classes. In a nutshell, in every update, each active (*i.e.*, positive valued) feature  $f$  removes some weight from a subset of the negative classes it is connected to, and adds the deducted weights to the true class connection, denoted  $w_{f,c_x}$  (if  $w_{f,c} = 0$ , *i.e.*, not connected, it creates the connection, that is, adds the list entry). If  $f$  is not connected to any negative class or if insufficient weight is removed, it transfers some weight from a “free” (dummy) weight source  $w_{f,c_0}$ , initialized to 1.0 before training begins. The weights are always kept nonnegative, so that class scores are nonnegative as well (a major difference from PA). OOO also keeps each matrix row further sparse by using top  $d$  weights (at most) for each feature during scoring, and OOO drops tiny weights (we report performance with  $d = 100, 500, 1000$ , and 2000). In an update, for a feature  $f$  with value  $x_f$ , the total amount moved to the true connection is at most  $x_f \beta$ , thus the score of the true class increases by at most  $\sum x_f^2 \beta = \beta$  (as vectors are l2 or normalized).  $\beta = 0.001$  in our experiments.

The maximum amount to deduct from each negative class is given by the function `compute_deductions` (and explained in [4]). The parameter  $k$  was set to 15 (at most 15 negative classes to deduct from). The smaller the parameter  $k$ , the faster and simpler the update, but accuracy slightly degrades. The presented variant of OOO is also somewhat simpler than the one given in our prior work [4], and in particular this version deducts from the free source *after* deducting from the negatives, which led to better generalization (less overfitting) when multiple passes are involved.

### 2.2 The PA Update

While OOO focuses on each feature in updating, the rows of the weight matrix  $W$ , PA is best viewed as working on class prototypes, or the columns of  $W$  (Figure 2). Here each category  $c$  is associated with the column weight vector  $\mathbf{w}^c$  (column  $c$  of  $W$ ) called its *prototype*. The score class  $c$  obtains, can be expressed as the dot product  $s_c = x^T \cdot \mathbf{w}^c$ .

```

OOZ_Update( $x, c_x, \tilde{C}_x, \delta, \beta$ ) {
   $\beta \leftarrow \min(\delta/2, \beta)$  /* possibly lower learning rate */
   $D \leftarrow \text{Compute\_Deductions}(k, s_{c_x} - \delta, \beta, \tilde{C}_x)$ 
   $R \leftarrow D$ . /* R is remainders, initialized to D */
   $\forall f \in x, \text{Shift\_Connection\_Weights}(f, x_f, x_f \beta, D, R, c_x)$ 
}

Compute\_Deductions( $k, s_{min}, \beta, \tilde{C}_x$ ) {
   $Y \leftarrow \emptyset, D \leftarrow \emptyset$ . /* D is a map */
   $\forall c \in \tilde{C}_x, \text{if } s_c > s_{min}, Y \leftarrow Y \cup \{(c, s_c)\}, D(c) \leftarrow 0$ 
   $s \leftarrow \text{Sorted scores in } Y$ . /* an array in decreasing order */
   $cum \leftarrow 0, n \leftarrow 1, i \leftarrow 0$ .
  loop while ( $cum \leq \beta$ ) { /* update deduction values */
     $\Delta \leftarrow \beta - cum$ 
    if ( $i < k$ )  $\Delta \leftarrow \min(\Delta, s[i] - s[i + 1])$ 
     $\forall c, s_c > s[i + 1], D(c) \leftarrow D(c) + \frac{\Delta}{n}$ 
     $cum \leftarrow cum + \Delta, n \leftarrow n + 1, i \leftarrow i + 1$ 
  }
  return  $D$ 
}

/* Shift Weights From Negative Connections,
and free source */
Shift\_Connection\_Weights( $f, x_f, r, D, R, c_x$ ) {
   $boost \leftarrow 0$ .
  /* process the connected offending concepts */
  loop over  $c \in C$ , where  $w_{f,c} > 0$  and  $R(c) > 0$  {
    if  $r$  is 0, break the loop.
    /* Until no more allowance */
     $h \leftarrow \min(R(c)/x_f, w_{f,c}, x_f D(c), r)$ 
     $boost \leftarrow boost + h, w_{f,c} \leftarrow w_{f,c} - h$ 
     $R(c) \leftarrow R(c) - hx_f, r \leftarrow r - h$ 
  }
  /* deduct from free weight src */
   $boost \leftarrow \min(w_{f,c_0}, r) + boost$ 
   $w_{f,c_x} \leftarrow w_{f,c_x} + boost$ 
}

```

**Fig. 1.** Pseudo-code for OOZ and its subroutines. OOZ shifts weights from offending negatives and the free “dummy” source to the true positive weight  $w_{f,c_x}$ . Initially,  $w_{f,c_0} = 1.0$ .

```

PA-II-Update( $x, c_x, C$ ) {
  If hinge loss  $\mathcal{L} = 1 - s_{c_x} + s_{c'} \geq 0$ , where  $s_{c'} = \max_{c \neq c_x} s_c$ , an update is performed:
  Let  $\tau = \frac{\mathcal{L}}{\|\mathbf{x}\|^2 + \frac{1}{2C}}$ 
  Update category prototypes:  $\mathbf{w}_{t+1}^{c_x} = \mathbf{w}_t^{c_x} + \tau \mathbf{x}$ ,  $\mathbf{w}_{t+1}^{c'} = \mathbf{w}_t^{c'} - \tau \mathbf{x}$ 
}

```

**Fig. 2.** The PA-II Update variant (after [1]).  $C$  is an aggressiveness parameter ( $C = 1$  in our experiments, and  $\|\mathbf{x}\|^2 = 1$  with our l2 normalization).

The PA algorithm [1] learns the category prototype in an online manner by solving the following constrained optimization problem:

$$\mathbf{w}_{t+1} = \arg \min_{\mathbf{w}} \frac{1}{2} \|\mathbf{w} - \mathbf{w}_t\|^2 \quad s.t. \quad \mathcal{L}(\mathbf{w}; (\mathbf{x}, c_x)) = 0. \quad (1)$$

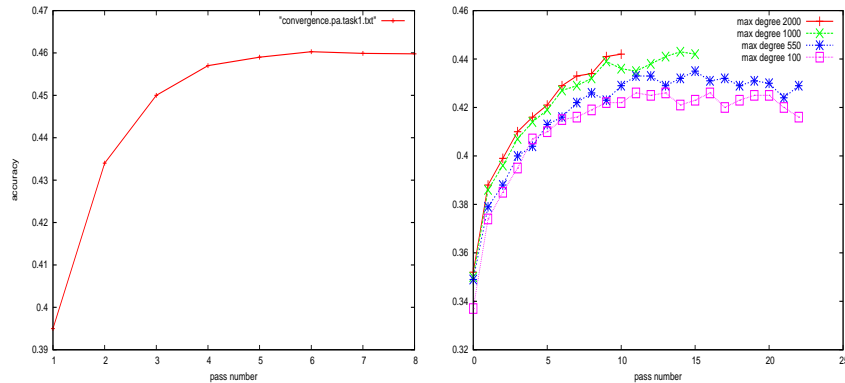
The PA algorithm *passively* accepts a solution whose hinge loss is zero, while it *aggressively* forces the new prototype vector to stay as close as possible to the one previously learned. Here the hinge loss is determined by the largest violation by the false positive and true positive classes, as shown in Algorithm 2. Note that the hinge loss is positive unless the classification outcome exceeds a *margin* of 1. The solution of the above optimization problem can be found using the Lagrangian method, and this leads to the update shown in Algorithm 2. Note that in each update, only two prototypes are updated, which makes the update step simple.<sup>3</sup>

<sup>3</sup> There are several variants to PA. We are presenting the PA-II variant, which performed the best in our experiments.

### 3 Data and Experiments

The class distribution was highly skewed as is often the case. While the average number of instances per class is 10, the most frequent class had almost 3000 instances. The top 2400 classes (top 20% in terms of number of positive instances), contained 80k instances out of the 120k total (thus the 80-20 rule roughly applies here). For task 1, the features were the terms that occurred on the page, and the average vector size (number of unique terms) was rather long at 173, with standard deviation of 200. The maximum length reached 7k. Vector size goes up for task 4 as it also includes terms from category descriptions. Tasks 3 and 4 switched the two feature types for training and test. We used tfidf l2-normalization for vector representation, and did no other vector processing. Our experiments were conducted on 64bit Linux machines, the 1st author using a 24GB memory and the 2nd using a 32GB with 2.8GHz.

**Dry Runs.** On the task 1 dry-run data set (smaller set for quick experiments), the PA algorithm obtained accuracy of 0.477 (on validation), reached within 10 passes, and OoZ, with parameter experimentation (choice of margin and learning rate), obtained a maximum of 0.488, reached in 60 passes (and under no degree constraints). We decided to apply PA in all the tasks because it performed better than all participants' entries, and because it appeared that we needed to remove all or most efficiency constraints for OoZ, to reach similar performance, and OoZ was parameter sensitive. However, we report on experiments on OoZ and PA when trained on the full training set on task 1 (training and validation set put together).



**Fig. 3.** (a) Accuracy of PA after each pass for task 1, reaching a maximum of 0.46. (b) Accuracy of OoZ under four max degree  $d$  settings:  $d = 2000, 1000, 500, 100$  (from top to bottom). At the time of writing, for  $d = 2000$  and  $d = 1000$ , we did not have the results for subsequent passes. Pass durations differed substantially between PA and OoZ. See Convergence & Efficiency subsection.

**Convergence and Efficiency.** Figure 4 illustrates the convergence behavior of the PA algorithm in task 4. The improvement is most significant in the first three passes. Then it levels off (or slightly deteriorate due to overfitting) after 7 passes. Similar observations are made in task 1 (Figure 3(a)). The convergence behavior of OoZ is shown in Figure 3(b), whose index is much smaller than the PA counterpart (reported below). Accuracy appears to reach maximum in under

20 training passes. By trading off sparsity (allowing more edge fanouts) and hence training time, the accuracy of OoZ improves by 1.5% to 2%.

In task1, the number of edges in the index for OoZ, with  $d = 100$ , ranged from 2.5 million edges to around 2.9 million edges for all subsequent passes. For  $d = 1000$ , the size ranged from 6.5 million (at pass 1) to under 6.7 million edges. For  $d = 2000$ , it ranged from 7.2 million (at pass 1) to just under 8 million edges. For PA, in task1, the number of edges ranged from 19 million in pass 1, to 35 million, 39 million, 42 million, and 45 million, respectively, in passes 4, 5, 6, and 7 (thus the number of edges seems to steadily grow, as there is no constraint on outdegree).<sup>4</sup> Thus, the number of edges used by OoZ on task 1 was within 5% to 10% of PA, while it underperformed by 2% to 4% in accuracy for the range of degrees tested.

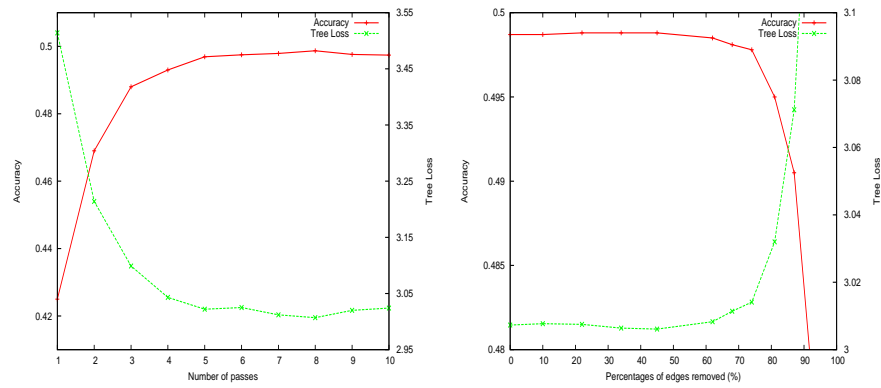
Similarly, each pass took progressively longer for PA. For task 1, the PA passes took respectively 12, 24, 27, 30, 33, and 35 hours. For OoZ, with  $d = 100$ , the duration was 1 hour for pass 1, increasing to a stable 2.5 hours for passes 10 to 20. For  $d = 1000$ , they took from 3.4 hours to around 5.5, and for  $d = 2000$ , they took from 4.6 hours to around 7.5. Classification of the 35k test instances with the PA models took around 3 hours, while with OoZ models at pass 14, with  $d = 100$  and  $d = 1000$  the times were respectively 3 and 13 minutes.

Sparsifying the PA-learned model has the potential of helping memory and time efficiency without losing much on accuracy. Intuitively, some features and weights are more important than others for prediction. We ranked the weights of the PA-learned weight matrix by their absolute values. The learned model is sparsified by retaining the top weighted edges whose total number is below the size threshold. The models' accuracy and tree loss are presented in Figure 4(b) for task 4. Using a sparse model as small as 30% (14 million edges) of the original learned model (48 million edges) does not appear to hurt accuracy. After this point, however, the model will quickly decrease in performance. Sparsifying periodically, for example after each pass, may be an effective technique. Note that OoZ sparsifies each feature (with every update).

**Committee Voting and Other Extensions.** While the accuracy appears steady after a few passes, we observed that the classes assigned to test instances (top-ranked classes) differed from one pass to another for around 10% of the test instances. Starting the training with different random seeds also created somewhat more diversity. The instability of the models learned indicates that there is room for further optimization: the online algorithms are not converging to the best model. This instability also suggested the possibility of accuracy improvement by voting. With four random starts and indices taken from 3 passes (5, 6, and 7), a total of 12 models, we obtained around a 0.5% accuracy boost. Tree loss and max F1 improved too. This accuracy gain was observed on all tasks. Instead of taking the majority voted class, choosing the class with lowest frequency (but voted by at least one model) yielded the best macro F1 (while degrading accuracy only slightly from the maximum achievable accuracy). Increasing the ensemble size had diminishing returns. Another possibility for improved performance is the use of EDGE [2], although our preliminary experiments indicated similar performance to committee voting. We also experimented with over sampling the minority classes. While max F1 improved for the first pass, subsequent passes showed a degradation in accuracy.

---

<sup>4</sup> The original PA algorithm [1] is not concerned with the many class setting and in particular the space and time efficiency aspects.



**Fig. 4.** (a) Accuracy and tree loss of PA after each pass for task 4 on the test data. (b) Performance vs. percentages of weights removed from PA model on task 4.

**Conclusions.** We presented two flat techniques, OoZ and PA, for categorizing with more than 10,000 categories of ODP in the Large-Scale Hierarchical Text Classification PASCAL challenge. Although forgoing the hierarchical information, the flat algorithms were capable of achieving top-tier classification results in the challenge while retaining simplicity in algorithmic design and implementation (consistent with our previous findings [3, 4]). OoZ sparsifies the features (drops edges) as it learns. We found that to achieve best accuracies on this data, we couldn't limit the connectivity of features substantially during learning (unlike our previous experience [3]). We investigated the idea of sparsifying the PA-learned model, after training. For both OoZ and PA, we observed an accuracy-efficiency trade-off. We explored extensions, including committee voting, which improved performance. We expect there is significant room to push the envelop of accuracy-efficiency. Studying efficient batch index learning, with regularized objectives, is a promising future direction. Finally, can using the hierarchy somehow aid classification and in particular accuracies at the deeper classes? So far, our experience suggests otherwise.

**Acknowledgements.** Many thanks to the PASCAL team for setting up this challenge, and for assisting the teams along the way.

## References

1. Crammer, K., Dekel, O., Keshet, J., Shalev-Shwartz, S., Singer, Y.: Online passive-aggressive algorithms. *J. of Machine Learning Research (JMLR)* 7 (2006)
2. Huang, J., Madani, O., Giles, C.L.: Error-driven generalist+experts (EDGE): A multi-stage ensemble framework for text categorization. In: *ACM CIKM* (2008)
3. Madani, O., Connor, M., Greiner, W.: Learning when concepts abound. *JMLR* 10 (2009)
4. Madani, O., Huang, J.: On updates that constrain the number of connections of features during learning. In: *ACM KDD* (2008)