Large-Scale Many-Class Prediction via Flat Techniques

Omid Madani[1] and Jian Huang[2]

[1] SRI International, AI Center, Menlo Park, CA 94025, USA madani@ai.sri.com
[2] College of Information Sciences and Technology
Pennsylvania State University, State College PA 16802, USA 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 an extension of our short report paper on the competition, and includes a proof of convergence, with a mistake bound, for a variant of indexing in the two-class setting for the separable case.

1 Introduction

Many prediction tasks, in a variety of domains including text (*e.g.,* [14, 10]), speech, and images (*e.g.,* [1, 7, 12]), 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.,* [10, 11]). 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) [10, 11]. 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.

In this paper, we report on the performance of the algorithms we used in the five-month-long Large-Scale Hierarchical Text Classification PASCAL challenge, and establish the convergence of an indexing algorithm in the binary-class setting. We first briefly describe the PASCAL challenge. In that 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. The tasks differed on the 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 tree 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 [11] and a Passive-Aggressive (PA) variant [3],

on l2 normalized tfidf representation of instances. Both approaches obtained competitive accuracies, although we found that we needed to relax the efficiency constraints on OOZ for improved accuracy. 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.

OOZ and some other indexing algorithms constrain the features' weights to a bounded interval such as $[0, 1]$. This property makes it easier to dynamically decide which (feature-to-class) edges to keep or drop. Weight dropping is crucial for memory and time efficiency in the online setting. In the appendix, we include a proof of convergence of a variant of OOZ in the two-class case (a weight-bounded perceptron), and discuss open problems (generalization to multiple classes) and connections to previous results.

2 Algorithms

Both OOZ (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 score class c obtains, s_c can be expressed as the dot product $s_c = x^T \cdot \mathbf{w}^c$, where $\mathbf{w}^c$ is column c of W (the column of W corresponding to class c). The highest-scoring class (breaking ties arbitrarily), $\mathrm{argmax}_c\, s_c$, is the predicted class. OOZ 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 OOZ Update

After scoring, let s_{c_x} denote the score of the true class c_x. OOZ (Figure 1) updates whenever $\exists c' \neq c, s_{c'} \geq s_{c_x} - \delta$, where c' denotes another class, and δ 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). OOZ also keeps each matrix row further sparse by using top d weights (at most) for each feature during scoring, and OOZ 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.

OOZ_Update(x**,** c_x**,** $\tilde{C}_x$**,** δ**,** β**)** {
 $\beta \leftarrow$ **min**$(\delta/2, \beta)$ **/* possibly lower learning rate */**
 $D \leftarrow$**Compute_Deductions(**k**,** $s_{c_x} - \delta$**,** β**,** $\tilde{C}_x$**)**
 $R \leftarrow D$**. /* R is remainders, initialized to D */**
 $\forall f \in x$**, Shift_Connection_Weights(**f**,** x_f**,** $x_f\beta$**,** D**,** R**,** c_x**)**
}

Compute_Deductions(k**,**s_{min}**,** β**,** $\tilde{C}_x$**)** {
 $Y \leftarrow \emptyset, D \leftarrow \emptyset$**. /* D is a map */**
 $\forall c \in \tilde{C}_x$**, if** $s_c > s_{min}, Y \leftarrow Y \cup \{(c, s_c)\}, D(c) \leftarrow 0$
 s $\leftarrow$ **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 first shifts weights from offending negatives and then the free "dummy" source to the true positive weight w_{f,c_x}. Initially, $w_{f,c_0} = 1.0$.

The maximum amount to deduct from each negative class is given by the function `compute_deductions` (and explained in [11]). 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 OOZ is also somewhat simpler than the one given in our prior work [11], 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.

In the Appendix, we show that an OOZ variant converges in the binary-class setting, given that there exists a perfect separator, while the general multiclass setting remains open. The proof is via showing reduction in Euclidean distance to the separator after each update.

2.2 The Passive Aggressive (PA) Update

While OOZ 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 $s_c = x^T \cdot \mathbf{w}^c$.

The original PA algorithm [3] learns the category prototype in an online manner by solving the following constrained optimization problem:

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

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 [3]). C is an aggressiveness parameter ($C = 1$ in our experiments, and $\|\mathbf{x}\|^2 = 1$ with our l2 normalization).

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 Fig. 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 Fig. 2. Note that in each update, only two prototypes are updated, which makes the update step simple.

To handle the problem of noise in the training instances, the PA-I and PA-II variants add slack terms $C\xi$ and $C\xi^2$ in the optimization function in Equation 1 (similar to that in SVM), respectively (and apply a more lenient constraint $\mathcal{L} \leq \xi$, $\xi \geq 0$). We adopt the PA-II variant which showed the best performance in all our experiments. We implement an efficient index-based sparse representation of the prototypes: the prototypes expand on-demand during the online training process to dramatically reduce space usage.

We note that the PA algorithm is more efficient in updating only two prototypes than its predecessor Multiclass Multilabel Perceptron (MMP) [4]. However, as a category prototype algorithm, PA needs to evaluate the similarity of an input vector with *each* category prototype in both training and prediction. Also, after learning the class prototypes (and hence the weight matrix W) are not sparse (we will investigate sparsification as a means to scale up the PA algorithm in the experiments section). In the many-class learning setting (with tens of thousands of categories), PA is less efficient than an index-based learning method such as OOZ (which updates active features with only limited fan-outs).

2.3 Strategies for Improving Categorization Accuracy

In the previous sections, we focused our attention on online learning algorithms for multi-class learning. In this section, we present several techniques that can improve the categorization performance in the many class setting.

We first note that it is often necessary to run the online learning algorithm for several passes to reach highest accuracy. In the case of PA (and to an extent OOZ), this is in part because only two prototypes are updated per instance, and/or the learning rate maybe small. It may therefore be necessary to promote the true positives or demote the false positives with several update operations (achieved in several passes).

Second, we take advantage of the fact that committee learning can generally help to reduce the variance in prediction. Online learning algorithms typically learn slightly different models when the training instances are presented in different orders. For both OOZ and PA, observing two instances in different orders can lead to different update behaviors and ultimately different

Algorithm 1 Committee learning.

Input: Size of committee J; number of training passes P; training problem $\{(\mathbf{x}_t, Y_t\}_{t=1}^{T}$.
1: Feature normalization (e.g. using `tf-idf` normalization scheme).
2: **for** $j \leftarrow 1$ to J **do**
3: Permute instances using random seed s^j: $(i_1, .., i_T) = \texttt{Perm}((1, ..T); s^j)$.
4: **for** $p \leftarrow 1$ to P **do**
5: $M_j^{(p)} \leftarrow \texttt{Online\_Component\_Learner}(M_j^{(p-1)}, \{(\mathbf{x}_{i_t}, Y_{i_t}\}_{t=1}^{T})$
6: **end for**
7: **end for**
Output: A committee of trained models $\{M_j^{(P)}\}_{j=1}^{J}$

models. In particular, we noticed that the class predictions of the learned models differed from one pass to another for both OOZ and PA (see experiments Section 3). The variation in ranking is not unexpected, specially when so many classes are ranked. Therefore, we used the models after certain passes to form a committee (see Algorithm 1). We experimented with several aggregation strategies for committee prediction:

$$c_{maj}(\mathbf{x}) = \arg\max_{c} \sum_{j=1}^{J} I[M_j^{(P)}(\mathbf{x}) = c] \qquad \text{(plain majority vote)} \tag{2}$$

$$c_{min}(\mathbf{x}) = M_i^{(P)}(\mathbf{x}), \text{ where } i = \arg\min_{j} Freq(M_j^{(P)}(\mathbf{x})) \tag{3}$$

Where $M_j^{(P)}(\mathbf{x})$ denotes the class ranked highest (predicted) by model (or committee member) j (after pass P), and $Freq(c)$ is the frequency of class c in the training set. The first aggregation strategy is simple majority vote, which we found to significantly improve accuracy over individual models. The second strategy is a variation. It is biased towards the category with the lowest frequency in training data. We found that this aggregation strategy yields the best results that not only improves macro F1 over single models or the plain majority, but also (micro) accuracy over individual models (while underperforming plain majority somewhat in accuracy).

Finally, observing that there are about 10% of the training instances with categories appearing less than 10 times in the training data, we experimented with the approach of oversampling the instances from rare categories, a common strategy in dealing with the class imbalance problem. We found that this yields significant improvement of macro F1 especially in the earlier passes, meanwhile it retains similar accuracy. The peak of the accuracy and macro F1 however in later passes is not as high when compared to no over-sampling. We sampled only once in the very begining. Resampling at the beginning of each pass and other sampling procedures may improve results. We leave this as an option when early convergence is desired.

3 Data and Experiments

The distribution of ODP categories was highly skewed as is often the case in many real world large scale text categorization problems. 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

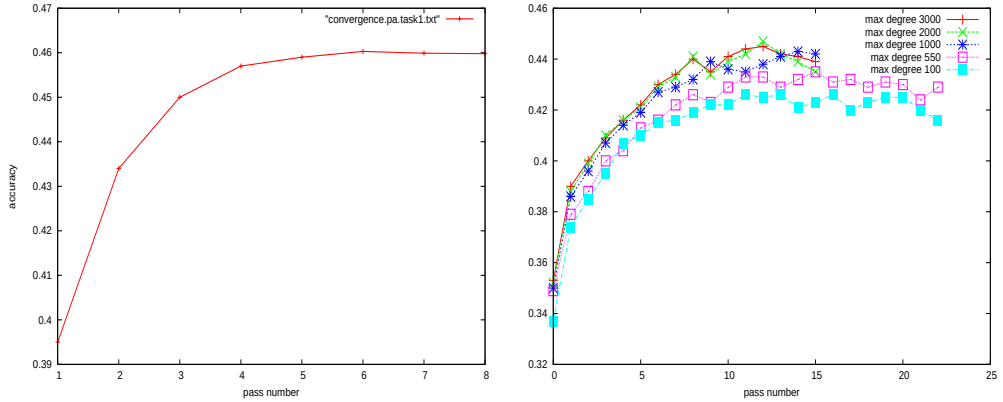

Fig. 3. (a) Accuracy of PA after each pass for task 1, reaching a maximum of 0.46. (b) Accuracy of OOZ under four max degree d settings: $d = 3000, 2000, 1000, 500, 100$ (from top to bottom). Pass durations differed substantially between PA and OOZ. See Convergence & Efficiency subsection.

the 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 tf-idf and 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). We also applied two other algorithms, MMP [4], and EMA [11], but OOZ and PA, with their focus on hinge-loss reduction, performed significantly better on this data.

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).[3] 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 (Section 2.3). With four random starts and indices (models) taken from 3 passes (5, 6, and 7), a total of 12 models, we obtained around a 0.5% accuracy boost. Tree loss and macro F1 improved as well. This accuracy gain was observed on all four 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 would increase performance, but as would be expected it has diminishing returns. Another possibility for improved performance is the use of EDGE [9], although our preliminary experiments indicated similar performance to committee voting. We also experimented with over sampling the minority classes. While macro F1 improved for the first few passes, as we mentioned earlier, its accuracy peak did not match that with the original training problem.

Performance on Other Datasets. In our previous experiments [11], we had used l2 normalized term frequency vector representation. We noted the improvement in R_1 (by at least 2%) in the chanllenge, using l2-normalized tfidf representations, when compared to plain l2-normalized

[3] The original PA algorithm [3] is not concerned with the many-class setting and in particular the space and time efficiency aspects.

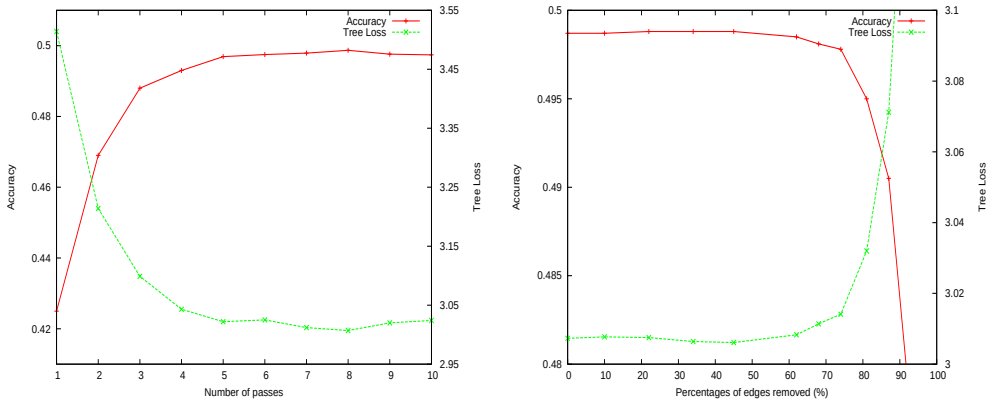

Fig. 4. (a) Accuracy and tree loss of PA after each pass for task 4 on the test data. (b) Performance vs. percentages of weights removed from PA model on task 4.

	newsgroups, 20 classes, 20k instances				industry, 104 classes, 10k instances			
	no tfidf		tfidf		no tfidf		tfidf	
	R1	R5	R1	R5	R1	R5	R1	R5
OOZ	0.861	0.979	0.876 ± 005	0.983	0.90	0.95	0.921	0.959
PA	0.85	0.97	0.87 ± 004	0.979	0.87	0.93	0.928	0.959

Table 1. Accuracies ("recall" at rank 1, R_1, and rank 5, R_5) on two data sets (average over 10 splits each), newsgroups and industry (the same experimental set up of [11]). Both PA (PA II) and OOZ reach their best accuracies at around 20 passes on newsgroups (PA with margin 0.05 and learning rate of 0.01, and OOZ with margin 0.5 and rate of 0.1), and under 10 passes for industry. tfidf representations boost accuracy, and we see some improved performance compared to [11].

frequency reporesentation. We repeated the experiments of [11] on two of the datasets with the tfidf refpresenation. The accuracies are shown in Table 1 (R_1 or "recall" at rank 1, is simple accuracy, and R_5 is accuracy within top 5). We observe that the use of tfidf boosts performance on these datasets as well (and PA seems to benefit more).

4 Discussion and Conclusions

We investigated 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 [10, 11]). See also [5], who find flat PA-based techniques competitive with hierarchical versions on several smaller multiclass problems. In terms of taxonomy or tree-based losses, some hierarchical methods may show some advantages (*e.g.*, [5, 2]), though there remains the complexity of encoding the hierarchy, which may not be a tree (*e.g.*, the OHSUMED data set), or may not be available.

OOZ sparsifies the features (drops edges) dynamically during learning. We found that to achieve best accuracies on the challenge data, we couldn't limit the connectivity of features substantially during learning (unlike our previous experience [10]). However, the feature degree and minimum weight constraints provide knobs with which we can control the accuracy-efficiency tradeoff. We provided a convergence proof for an OOZ variant for the two-class case. 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. With increasing number of passes, online algorithms often improve in accuracy. However, overfitting also tends to increase as well. Furthermore, as we saw, the algorithms can also oscillate and not converge to better models. When batch training is possible, the balance between over- and underfitting may be controlled better, for improved generalization. Studying efficient batch index learning, with appropriate regularized objectives, is a promising future direction. Working toward a better understanding of the relation between various problem properties, such as the number of classes and instances and average vector length, on one hand, and the tradeoff between efficiency and accuracy on the other, should also be fruitful.

Acknowledgements

Many thanks to the PASCAL team for setting up the challenge and their assisting the teams along the way.

References

1. Aradhye, H., Toderici, G., Yagnik, J.: Video2text: Learning to annotate video content. In: IEEE Int. Conf. on Data Mining (ICDM) Workshop on Internet Multimedia Mining (2009)
2. Cesa-Bianchi, N., Gentile, C., Zaniboni, L.: Incremental algorithms for hierarchical classification. J. of Machine Learning Research (JMLR) 7 (2006)
3. Crammer, K., Dekel, O., Keshet, J., Shalev-Shwartz, S., Singer, Y.: Online passive-aggressive algorithms. JMLR 7 (2006)
4. Crammer, K., Singer, Y.: A family of additive online algorithms for category ranking. JMLR 3, 2003 (2003)
5. Dekel, O., Keshet, J., Singer, Y.: Large margin hierarchical classification. In: ICML (2004)
6. Duda, R.O., Hart, P.E., Stork, D.G.: Pattern Classification. Wiley-Interscience, 2 edn. (2000)
7. Fidler, S., Leonardis, A.: Towards scalable representations of object categories: Learning a hierarchy of parts. In: Proc. of IEEE Int. Conf. on Vision and Pattern Recognition (CVPR) (2007)
8. Grove, A.J., Littlestone, N., Schuurmans, D.: General convergence results for linear discriminant updates. Machine Learning (2001)
9. Huang, J., Madani, O., Giles, C.L.: Error-driven generalist+experts (EDGE): A multi-stage ensemble framework for text categorization. In: ACM CIKM (2008)
10. Madani, O., Connor, M., Greiner, W.: Learning when concepts abound. JMLR 10 (2009)
11. Madani, O., Huang, J.: On updates that constrain the number of connections of features during learning. In: ACM KDD (2008)
12. Rosenfeld, R.: Two decades of statistical language modeling: Where do we go from here? IEEE 88(8) (2000)

13. de Sa, J.M., Felgueiras, C.A.S.: Perceptron learning with discrete weights. In: European Symposium on Artifical Neural Networks (ESANN) (2005)
14. Xue, G., Xing, D., Yang, Q., Yu, Y.: Deep classification in large-scale text hierarchies. In: Proc. Int. ACM SIGIR Conf. on Research and Development in Information Retrieval (SIGIR) (2008)

A Convergence of OOZ in the Binary-Class Case

The OOZ algorithm becomes a variant of the perceptron algorithm in the case of two classes. The difference from plain perceptron are: the learned weights are bounded, and the algorithm accepts non-negative feature values only. We may refer to this variant as the *weight-bounded perceptron*. In this variant, each feature keeps two weights, one for c_1 (the first class) and another for c_2. The weights are nonnegative and add to 1.0 (therefore, one weight per feature is sufficient).[4] An update occurs iff the score of the false class ties or is greater than the score of the true class. Whenever a feature updates, it subtracts an amount from the negative (false) class and adds it to the true class, unless the feature already has 1.0 weight for the true class.[5] Below, assume c_1 is the true class, and let β denote the learning rate ($0 < \beta \leq 1$), x_i the value of feature i in the instance, and w_{i1} be current weight of feature to c_1. Then in an update, we have $w_{i1} \leftarrow w_{i1} + \min(x_i\beta, 1 - w_{i1})$ (and symmetrically, $w_{i2} \leftarrow w_{i2} - \min(x_i\beta, 1 - w_{i1})$). So there is no change if already $w_{i1} = 1$ (feature i is saturated).

In this section, we show convergence of OOZ (or the weight-bounded perceptron) in this binary-class setting, under the assumption of separability, and derive a mistake bound similar to that for perceptron. We are not aware of any convergence result in this setting. Generalization learning bounds (but not convergence bounds) for perceptrons with bounded integer weights have been studied [13]. In that work, it is assumed a perceptron minimizing empirical zero-one error is given, and the problem is deriving bounds on the generalization zero-one error.

Assume there exists a separator W^* (the weights in W^*, w^*_{ij} also satisfy the boundedness: $\forall i, 0 \leq w^*_{i1} \leq 1, w^*_{i1} + w^*_{i2} = 1$), and that the instances are L2 normalized $\sum_i x_i^2 = 1$, and $x_i \geq 0$, and there is a positive separation: for some $u \geq 3$ (we show 3 suffices), on any instance obtained via sampling from the distribution, $s_{c_x} - s_{\bar{c_x}} \geq u\beta$ for some u, where s_{c_x} is the score obtained by true class and $s_{\bar{c_x}}$ is the score obtained by the negative class. The proof is based on showing a reduction in (squared) Euclidean distance between W (our current weight vector or hypothesis) and W^* after each mistake (and subsequent update). There are at least two major proof methods establishing convergence for the plain perceptron algorithm: showing the dot-product between the current hypothesis and separator increases (see *e.g.,* [8]), and showing Euclidean distance decreases (as in [6]). We note that, due to the bounded condition on weights, the dot-product property does not hold for the bounded-weight perceptron (simple counter examples exist).

Without loss of generality, we will focus on the first column (weights to c_1) for both W and W^*, wherein after an update some weights to c_1 (the true class) must increase. For simplicity, we write w_i for w_{i1}, and $W = W_1$ (the first column). Let W' denote the weight vector after an

[4] Equivalently, we could assume each feature keeps one weight parameter, which ranges in $[-1, 1]$ (the difference of the weight in the two-weight case).

[5] This "saturation" condition is the cause of the difference in our convergence analysis from the typical perceptron convergence argument.

update. Below, $d = ||W - W^*||_2^2$, and $d' = ||W' - W^*||_2^2$ (distances before and after update).

$$\begin{aligned} d - d' &= ||W - W^*||_2^2 - ||W' - W^*||_2^2 = \sum_{i \in x} (w_i - {w^*}_i)^2 - \sum_{i \in x} ({w'}_i - {w^*}_i)^2 \\ &= \sum_{i \in x} (w_i^2 - {w'}_i^2 + 2w_i^*({w'}_i - w_i)) \\ &= \sum_{i \in x} w_i^2 - (w_i + \min(\beta x_i, 1 - w_i))^2 + 2w_i^*(w_i + \min(\beta x_i, 1 - w_i) - w_i) \\ &= \sum_{i \in x} w_i^2 - \min(w_i + \beta x_i, 1)^2 + 2w_i^*(\min(\beta x_i, 1 - w_i)), \end{aligned}$$

where we have $0 \leq w_i \leq 1$, $0 \leq w_i^* \leq 1$, $\sum x_i^2 = 1$, $x_i \geq 0$, and the separation constraint, $\sum_i x_i w_i^* \geq (\sum_i x_i(1 - w_i^*)) + u\beta$ (and we show below, some $u \geq 3$ suffices), and, because an update took place, we have the update constraint $\sum_i x_i w_i \leq \sum_i x_i(1 - w_i)$.

In general some active features may only be partially "modifiable", *i.e.*, $w_i + \beta x_i > 1$, or $0 \leq 1 - w_i < \beta x_i$. We denote this set by $part(x)$. The constraints, in the form that we will use, are:

$$\begin{aligned} &\sum_i x_i w_i \leq \sum_i x_i(1 - w_i) \Rightarrow 2\sum_i x_i w_i \leq \sum_i x_i \Rightarrow \\ &2 \sum_{i \notin part(x)} x_i w_i + 2 \sum_{i \in part(x)} x_i w_i \leq \sum_i x_i \Rightarrow \\ &2 \sum_{i \notin part(x)} x_i w_i \leq \sum_i x_i - 2 \sum_{i \in part(x)} x_i w_i, \end{aligned}$$

and for W^*, the separation constraint can be written as:

$$\begin{aligned} &\sum_i x_i w_i^* \geq (\sum_i x_i(1 - w_i^*)) + u\beta \Rightarrow 2\sum_i x_i w_i^* \geq (\sum_i x_i) + u\beta \Rightarrow \\ &2 \sum_{i \notin part(x)} x_i w_i^* \geq (\sum_i x_i) + u\beta - 2 \sum_{i \in part(x)} x_i w_i^*. \end{aligned}$$

We have

$$\begin{aligned} d - d' &= \sum_{i \notin part(x)} w_i^2 - (w_i + \beta x_i)^2 + 2w_i^*(\beta x_i) + \sum_{i \in part(x)} w_i^2 - 1 + 2w_i^*(1 - w_i) \\ &= -\beta^2 \sum_{i \notin part(x)} x_i^2 + \sum_{i \notin part(x)} 2\beta x_i w_i^* - 2\beta x_i w_i + \sum_{i \in part(x)} w_i^2 - 1 + 2w_i^*(1 - w_i) \\ &\geq -\beta^2 \sum_{i \notin part(x)} x_i^2 + \beta(\sum_i x_i + u\beta - 2 \sum_{i \in part(x)} x_i w_i^* - \sum_i x_i + 2 \sum_{i \in part(x)} w_i x_i) + \\ &\quad \sum_{i \in part(x)} w_i^2 - 1 + 2w_i^*(1 - w_i) \\ &= -\beta^2 \sum_{i \notin part(x)} x_i^2 + u\beta^2 + \sum_{i \in part(x)} (2\beta x_i(w_i - w_i^*) + w_i^2 - 1 + 2w_i^*(1 - w_i)) \end{aligned}$$

First, we simplify the last sum (into an expression involving βx_i only). Each summand $(2\beta x_i(w_i - w_i^*) + w_i^2 - 1 + 2w_i^*(1 - w_i))$ is decreasing in w_i^*, as the derivative is not positive: $-2\beta x_i + 2(1 - w_i) \leq -2\beta x_i + 2\beta x_i = 0$, using $1 - w_i \leq \beta x_i$ (from the assumption that $i \in part(x)$). Thus we can increase w_i^* to 1.0 and decrease the difference, so the lower-bound for distance becomes:

$$
\begin{aligned}
d - d' &\geq -\beta^2 \sum_{i \notin part(x)} x_i^2 + u\beta^2 + \sum_{i \in part(x)} (2\beta x_i(w_i - 1) + w_i^2 - 1 + 2(1 - w_i)) \\
&= -\beta^2 \sum_{i \notin part(x)} x_i^2 + u\beta^2 + \sum_{i \in part(x)} (2\beta x_i(w_i - 1) + (w_i - 1)(w_i + 1) + 2(1 - w_i)) \\
&= -\beta^2 \sum_{i \notin part(x)} x_i^2 + u\beta^2 + \sum_{i \in part(x)} (1 - w_i)(-2\beta x_i - (w_i + 1) + 2) \\
&= -\beta^2 \sum_{i \notin part(x)} x_i^2 + u\beta^2 + \sum_{i \in part(x)} (1 - w_i)(-2\beta x_i - w_i + 1) \\
&\geq -\beta^2 \sum_{i \notin part(x)} x_i^2 + u\beta^2 + \sum_{i \in part(x)} (1 - w_i)(-2\beta x_i) \text{ (dropped a non-negative term)} \\
&\geq -2\beta^2 \sum_{i \notin part(x)} x_i^2 + u\beta^2 + \sum_{i \in part(x)} (\beta x_i)(-2\beta x_i) = -2\beta^2 \sum_i x_i^2 + u\beta^2 = (u - 2)\beta^2.
\end{aligned}
$$

In the last step, we again used the fact that for $i \in part(x), 1 - w_i \leq \beta x_i$ (and that $-2\beta x_i \leq 0$). Thus, as long as $u > 2$, we are guaranteed a positive difference. With $u = 3$, we are guaranteed a reduction of β^2 at minimum. Since the maximum distance can be $\|F\|^2$ (the number of features), we obtain a mistake bound of $O(\|F\|^2/\beta^2)$. With $\max_x \|x\|^2 = R$, instead of 1, we obtain $O(R\|F\|^2/\beta^2)$, which has a similarity to the mistake bound for perceptron: $O(R\|W^*\|^2/\beta^2)$. Given we begin with all 0.5s weight vector, $\mathbf{0.5}$ (for our case, akin to the zero vector), the number of mistakes will be $O(R\|W^* - \mathbf{0.5}\|^2/\beta^2)$.

Now, it is interesting that for several simple generalizations of feature updating to multiple classes, while keeping the features' weights in $[0, 1]$ (and summing to 1.0), examples exist showing that the Euclidean distance reduction does not hold. For instance, for the update in which a feature deducts some weight from the negative class that obtained the highest score (if not connected, then it doesn't deduct), the following two matrices are a counter example:

$$
W^* = \begin{pmatrix} 0 & 0.1 & 0.02 & \cdots & 0.02 \\ 0.9 & 0.1 & 0 & \cdots & 0 \end{pmatrix} W = \begin{pmatrix} 0.5 & 0.5 & 0 & \cdots & 0 \\ 0 & 0 & 0.1 & \cdots & 0 \end{pmatrix}
$$

Here with both (Boolean) features active, under W^* c_1 (the true class) obtains 0.9, while c_2 obtains 0.2, and other classes obtain 0.02. So the score of c_1 is well separated from others. The number of classes is sufficiently large so that each row sums to 1.0. Under W, $s_{c_2} = s_{c_1} = 0.5$, and all other classes obtain a score of 0.1 or 0. So c_2 is the only class with score equal or exceeding c_1. We see here that only row 1 of W is modified, and we can verify that even for small β, after update W' will have a larger Euclidean distance.

Thus, two related questions arise immediately: (1) Is there a generalization of the binary-class OOZ variant to the multiclass case (keeping the connection weights bounded) that satisfies the Euclidean distance reduction property during an update (and thus converges)? (2) Does OOZ or a close variant converge in the multiclass case?