On Updates that Constrain the Features' Connections During Learning

Omid Madani
SRI International, AI Center
Menlo Park, CA 94025
madani@ai.sri.com

Jian Huang
College of Information Sciences and Technology
Pennsylvania State University, PA 16802
jhuang@ist.psu.edu

ABSTRACT

In many multiclass learning scenarios, the number of classes is relatively large (thousands,...), or the space and time efficiency of the learning system can be crucial. We investigate two online update techniques especially suited to such problems. These updates share a sparsity preservation capacity: they allow for constraining the number of prediction connections that each feature can make. We show that one method, exponential moving average, is solving a "discrete" regression problem for each feature, changing the weights in the direction of minimizing the quadratic loss. We design the other method to improve a hinge loss subject to constraints, for better accuracy. We empirically explore the methods, and compare performance to previous indexing techniques, developed with the same goals, as well as other online algorithms based on prototype learning. We observe that while the classification accuracies are very promising, improving over previous indexing techniques, the scalability benefits are preserved.

Categories and Subject Descriptors

I.2 [**Artificial Intelligence**]: Learning
General Terms Algorithms, Performance, Theory
Keywords Multiclass Learning, Many Classes, Index Learning, Online Learning

1. INTRODUCTION

Many prediction and classification tasks, such as large-scale topical text categorization, predicting words in text, and visual classification, can be viewed as large-scale *many-class* learning problems, *i.e.*, problems with large numbers of classes (in the thousands and beyond) in addition to large numbers of features and instances (millions and beyond) [16, 18, 17, 7]. In the text classification domain, text fragments, such as queries, advertisements, news articles, or web pages can be mapped (classified) into a large collection of categories, for example, the topics in the Yahoo! topic hierarchy or the Open Directory Project (http://dir.yahoo.com and http://dmoz.org) (*e.g.*, [5, 16, 19]), or Wikipedia topics. In these problems, the number of classes can range in the hundreds of thousands. Once a search query or some piece of text is sufficiently accurately classified into one or more informative classes (from thousands of possible candidates), a personalization system can then take effective actions, for instance, recommend relevant ads or related items. In other scenarios, such as personalization on a desktop [3, 13], the number of classes can be relatively small (tens or hundreds), but efficiency in space and time during learning and classification remains crucial in practical applications.

Recently, *index* learning was introduced to address the challenges posed by many-class tasks, in particular the problems of how to quickly classify given an instance, and how to efficiently learn such an efficient classification system [18, 19]. A characteristic aspect of the indexing approach making it efficient is constraining the number of prediction connections that each feature makes: on average, a feature should connect to only a small number of classes (*e.g.*, 10s or 100s). The general update technique was termed feature-focus (FF) (as opposed to prototype focused) to emphasize this aspect. FF updating easily allows for controlling the number of connections of a feature. In that work, the primary goal was assessing the viability of the general indexing approach. It was observed that indexing was much more efficient than the commonly used methods of one-versus-rest and top-down hierarchical classification using a taxonomy [16], while the accuracies were often better, in particular, as the number of classes is increased. The goal of this paper is to begin a systematic study of feature-focus update methods. We also compare to two other online methods (based on prototype learning), namely multiclass perceptron and passive-aggressive (PA) algorithms [1, 2]. These algorithms also learn to rank classes given an instance.

We explore two FF update methods. One is simpler and involves *exponential weighted averaging*. We refer to it as EMA (pronounce it "Emma"). We derive a number of properties of EMA, and in particular show that it is changing the weights of a feature in the direction of minimizing the quadratic loss, between the probabilities that the active features assign to the classes and the true probabilities (as we explain). We also present the version of EMA that handles multiple true classes (labels) per instance. Our second update is designed for the goal of obtaining a lower 0/1 error by minimizing a hinge loss. This approach, which we call OOZ (pronounced "ooze"), involves a linear optimization task and requires more book keeping. Our experiments show that both EMA and OOZ updates can yield significant improvements in accuracy over the previous indexing method, while trading off some speed. In accuracy, OOZ is best over all. OOZ also compares favorably to Passive Aggressive and multiclass perceptron algorithms [1, 2]. These indexing methods however, as we observe in our experiments, scale to many classes, providing substantial advantages in training and classification times. The algorithms offer a novel approach even in the binary class setting.

We next begin with preliminaries and an overview of indexing. We then describe the EMA and OOZ algorithms. Section 4 presents our empirical findings, Section 5 discusses related work, and Section 6 concludes. An expanded version of the paper with further details and proofs appears in our technical report [20].

2. PRELIMINARIES AND NOTATION

Our setting is standard multiclass supervised learning, but with many classes. A learning problem consists of a set S of instances, each training instance specified by a column vector of feature values, x, as well as a set of classes that the instance belongs to C_x. We use x to refer to the instance itself as well. A *positive* class, given an instance x is any class $c \in C_x$, and a *negative* class is any class $c \notin C_x$. We also use the expression $x \in c$ to denote that instance x belongs to category c (c is a category of x). We say a problem is *multilabeled* if $|C_x| > 1$ for some instances. F and C denote respectively the set of all features and categories. Our current indexing algorithms ignore features with nonpositive value, and in our data sets, features do not have negative value. x_f denotes the value of feature f in the vector x, $x_f \geq 0$. If $x_f > 0$, we say feature f is *active* (in instance x), and denote this aspect by $f \in x$. The number of active features is denoted by $|x|$. We use $o^{(i)}$ to denote the ith member in a sequence of values or observations. Good references on machine learning for text classification include Sebastiani [25] and Lewis *et. al.* [15].

Various existing multiclass algorithms suffer from a mix of drawbacks in large scale many-class settings. For instance, the strategy of training and using binary classifiers in a one-versus-rest manner becomes increasingly prohibitive as the number of classes increases. Plain nearest neighbors becomes impractical when S is a stream of instances of indefinite length. A plausible approach to efficient classification (and updating or learning) is to try to "recall" (or retrieve) a relatively small number of classes as each (active) feature of a given instance is examined, and then to score the retrieved classes, akin to the use of the inverted index for document retrieval in IR. A difference is that we learn this index. We next formalize the concepts. An index is a directed weighted bipartite graph and equivalently a weight matrix W. The index maps or connects each feature to zero or more classes. An edge connecting feature f to category c has a positive weight denoted by $w_{f,c}$, or $w_{i,j}$ for feature i and category j, $i \geq 1, j \geq 1$. On presentation of an instance, the index is used to score categories. The (nonnegative) score of a category is determined by the active features: $s_c = \sum_{f \in x} x_f w_{f,c}$. The scored classes can then be ranked, or those whose score exceeds a threshold can be reported (assigned to the instance). In the matrix interpretation, the rows can correspond to features, the columns to classes, and $w_{i,j}$ denotes the entry in row i and column j. Scoring is computing the dot product: $\tilde{C}_x = x^T W$. The *outdegree* of a feature is simply the number of (outgoing) edges (connections) of the feature in the index, and equivalently, it is the number of nonzeros in its corresponding row. The *total outgoing weight* of a feature, denoted w_f, is $\sum_{c \in C} w_{f,c}$. The vector of weights for feature f in W is denoted by W_f.

On presentation of instance x, since there can be at most $|x|d$ classes scored, scoring (a dot product) takes $O(|x|d)$ with a straight forward index implementation. Thus, if a randomly picked feature of a randomly picked instance has relatively low d, scoring is efficient. d is in the order of 10s in our experiments. We will see that updates that we cover require number of operations in one or a few multiples of $|x|d$. In our experiments, during scoring, only the highest 25 weighted connections for each active feature are used for scoring.

To evaluate classification quality, we will mainly use the standard accuracy measure R_1 (one minus zero-one error) and a more relaxed R_5, defined as follows [18]. We sort the scored classes $\tilde{C}_x$ in decreasing order to obtain a ranking. Let k_x be the rank of the highest ranked true category for instance x in a given ranking. Thus $k_x \in \{1, 2, 3, \cdots\}$. If no positive appears in the ranking, then $k_x = \infty$. We use R_k to denote recall at (rank) k, $R_k = E_x[k_x \leq k]$, where E_x denotes expectation over the instance distribution and $[k_x \leq k] = 1$ iff $k_x \leq k$, and 0 otherwise (Iverson bracket).

3. TWO FEATURE-FOCUS VARIANTS

In both index learning methods, the weight matrix is 0 initially. This means an empty index (no edges) in the implementation. Both methods learn nonnegative weights, motivated by efficiency considerations, as we explain. We begin by describing EMA.

3.1 Exponential Moving Average (EMA)

It was shown in [18] that for the purpose of classification, features' "votes" for various classes can simply be aggregated. Features' votes are the proportion of concepts, or class conditional probabilities, in their own stream (the stream of concepts observed whenever the feature is active). Computing simple proportions was motivated by efficiency considerations in many-class learning: features should not use complex processing, nor keep much space to track history. For efficiency, features would drop proportions under a threshold without affecting overall accuracy significantly. That work gave two proportion estimation methods, one assuming a static distribution (proportions do not change over time), and a second one appropriate for nonstationarity. The authors focused on the single label setting and used the first update in experiments with indexing. Here, we derive the properties of the latter update method, the exponential moving average updating (EMA), shown in Figure 1 in a general form handling real-valued features and multilabel problems. For simplicity and for starters, imagine features are Boolean, $x_f \in \{0, 1\}$, and $|C_x| = 1$ (the single label case).

A moving average tracks the average of a time varying target value of interest, with applications in financial statistics, time series prediction, and signal analysis [6]. Observations are made at discrete time points. Let $o^{(t)}$ denote the observation at time t. The exponential (or exponentially weighted) moving average, is a convex combination of the last average and current observation, in its simplest form given by $w^{(t)} = (1 - \beta)w^{(t-1)} + \beta o^{(t)}, t \geq 1$, where $w^{(t)}$ is the (moving) average at time t, and β is a mixing rate, $0 < \beta \leq 1$. Equivalently, we may express $w^{(t)}$ as: $w^{(t)} = \beta \sum_{0 \leq k \leq t} (1 - \beta)^k o^{(t-k)}$, where $\beta o^{(0)}$ is the initial value.

```
EMA_Update($f$, $x_f$, $C_x$, $\beta$)
 /* All connections are decayed, then
   the connection to true class(es) boosted */
 $\forall c, w_{f,c} \leftarrow (1 - x_f^2\beta) w_{f,c}$ /* $0 < x_f \leq 1$ */
 $\forall c \in C_x, w_{f,c} \leftarrow w_{f,c} + x_f\beta$
 If $w_{f,c} < w_{min}$, then /* drop small weights */
     $w_{f,c} \leftarrow 0$
```

Figure 1: An *exponential moving average* version of feature updating. β is a learning rate or a "boost" amount, $0 < \beta \leq 1$, and x_f is the "activity" level of active feature f, $0 < x_f \leq 1$. The end effect is that the weight of the connection of feature f to every positive class is strengthened. Other connections are weakened. See also Equation 1.

Therefore, the latest observation value has the highest impact on the moving average, and more generally the observation at time $t - k$ is down weighted by $(1 - \beta)^k$, hence the name.

We can interpret the EMA of Figure 1 as computing a moving average W_f of a vector valued target for each feature: Each feature in an update "observes" a Boolean vector T of concept occurrence observations of size $|C|$, most entries being zero, one or a few being 1.0, indicating the positive classes. The EMA_Update performs:

$$W_f \leftarrow (1 - x_f^2\beta)W_f + x_f\beta T \tag{1}$$

All the individual moving average quantities (entries), in this case the concept proportions, are updated. However, most concept proportions (connections) are zero and have 0 value in the observation vector, so will remain zero: nothing need be done for them. For the ones with nonzero average, they are first weakened by $(1 - \beta)$, and for the observed ones (with value of 1.0 in the observed vector), a boost $x_f\beta$ is added. For efficiency, tiny weights are dropped (zeroed). The exponential moving average is attractive in our setting as it is particularly simple to implement and efficient: it does not require extra memory for keeping track of history.

Note that $(1 - \beta)w + \beta = w + (1 - w)\beta > w$, when $w < 1$, and with $0 < \beta$. Thus, weakening and boosting a weight increases the weight w as long as $w < 1$. Furthermore, the total outgoing weight of a feature remains bounded, always converging to 1.0 in the single-label setting, as the following shows.

LEMMA 3.1. *Let $0 < \beta \leq 1$, and assume the Boolean case.*

1. *The update $w^{(i+1)} \leftarrow (1-\beta)w^{(i)} + \beta w^*$, has a unique fixed point at w^*, i.e., $\lim_{i\to\infty} w^{(i)} = w^*$. The convergence is one-sided and geometric: $w^* - w^{(i+1)} = (1-\beta)(w^* - w^{(i)})$ (for instance, if $w^{(1)} < w^*$ then $\forall i \geq 1, w^{(i)} \leq w^*$).*
2. *For the update $w^{(i+1)} \leftarrow (1-\beta)w^{(i)} + \beta x^{(i)}$, when $\forall i, w_1 \leq x^{(i)} \leq w_2$, we have $\forall i, w_1 \leq w^{(i)} \leq w_2$.*
3. *In the EMA update method of Figure 1, when $\forall x, |C_x| = 1$ (the single-label setting) and $w_{min} = 0$, the total outgoing weight of feature f, converges to 1.0 in a one-sided manner In the multilabel setting, the outgoing weight never exceeds the maximum number of true positives per instance.*

Proof (sketch). The update is a linear function with non-negative slope. The second part follows from the properties of convex combinations. The third part follows from the first and second parts, once we realize that the total outgoing weight w_f is itself a moving average. □

Therefore, though certain connections may be zeroed in a given update when $w_{min} > 0$, the lost weight can be recaptured in future updates. There is always a push for the sum to approach 1.0. The row vector W_f corresponding to feature f always remains within the unit simplex. For the multilabel setting, since the weakening step is performed only once but there can be multiple boosts, a features' outgoing total weight may exceed 1.0. This is fine under the interpretation of the moving average: the proportions need not add up to at most 1.0, as given that a feature is active, there can be multiple positives. A feature that is always associated with the same two positive classes should obtain a weight of 1.0 to each class, its total outgoing weight should converge to 2.0.

In general w_{min} should be set as a function of the desired maximum outdegree, the learning rate, and the feature values. We set w_{min} to $\min(0.005, \beta/5)$. Note that a low β can lead to high outdegree, and slow convergence and slow adaptation to changes (slow learning). On the other hand, it allows for a more refined weight estimation, and thus potentially higher accuracy. See experiments (Section 4.2). Incorporating the degree of activity of a feature into the update affects the extent of the update. Updating proportional to activity level, or "volume", has been referred to as *volume weighting*. In addition to the moving average view for tracking possibly nonstationary concept proportions, EMA updating has a gradient interpretation. This further justifies the particular update expression given in Figure 1. Observe that, as an example, if a feature always has an activity level of $x_f = 0.5$, and is repeatedly updated for the same class c, $w_{f,c}$ would converge to 2.0, so that the product $x_f w_{f,c}$ would converge to the desired 1.0 (more scenarios are explored in [20]).

3.1.1 EMA Minimizes Squared Loss for Each Individual Feature

Observe that as the connection strength to the true class c_x approaches 1.0, the strengthening amount in the update decreases. EMA tries to correct most for values farthest from 1.0. Squared loss shares this sensitivity, and we verify that EMA updating changes the weight vector of a feature in the direction of the gradient of the objective of lowering quadratic loss. Given the current connection weights of an active feature, the quadratic loss of feature f is:

$$l = \sum_{c \notin C_x} (x_f w_{f,c})^2 + \sum_{c \in C_x} (1 - x_f w_{f,c})^2.$$

The partial derivatives are:

$$\frac{\partial l}{\partial w_{f,c_j}} = \sum_{c \notin C_x} \frac{\partial((x_f w_{f,c})^2)}{\partial w_{f,c_j}} + \sum_{c \in C_x} \frac{\partial((1 - x_f w_{f,c})^2)}{\partial w_{f,c_j}},$$

and so for $c_j \notin C_x$, $\frac{\partial l}{\partial w_{f,c_j}} = 2x_f^2 w_{f,c_j}$ and similarly for $c_j \in C_x$, $\frac{\partial l}{\partial w_{f,c_j}} = 2(1 - x_f w_{f,c_j})(-x_f) = 2x_f^2 w_{f,c_j} - 2x_f$. To lower the loss, a small step along the negative gradient vector is taken. With a step size of $\beta/2$, $W_f \leftarrow W_f - \frac{\beta}{2}\nabla l$, we obtain the EMA update given in Equation 1. We note that we may naturally want to minimize quadratic loss over the class scores (not individual feature's votes, but their aggregation). In this case, the updates can lead to negative weights, which can make the control of outdegree difficult. We do not explore that avenue here.

```
/* EMA Updating with Margin */
Margin_EMA_Updating($x$, $\delta_m$, $\beta$ )
  $\forall f \in x, \forall c, w_{f,c} \leftarrow (1 - x_f^2\beta)w_{f,c}$ /* weaken all */
  $\tilde{C}_x \leftarrow x^T W$ /* re-score classes */
  $s_{c'} \leftarrow \max_{c \notin C_x} s_c$ /* compute highest negative score*/
  $\forall c \in C_x$, if $s_c - s_{c'} < \delta_m$, then
      $w_{f,c} \leftarrow w_{f,c} + x_f\beta$
  $\forall c$, If $w_{f,c} < w_{min}$, then /* drop small weights */
      $w_{f,c} \leftarrow 0$
```

Figure 2: EMA updating when a margin threshold is used. First weaken all connections, then for every true category with (new) margin below the desired threshold, update. $s_{c'}$ is the score of a highest scoring negative category: $s_{c'} = \max_{c \in C - C_x} s_c$.

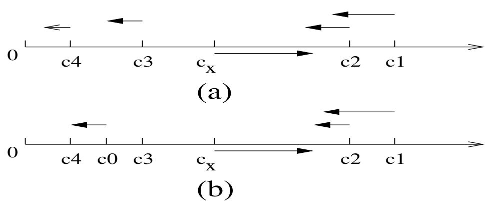

Figure 3: An example scenario showing how the scores are changed after an update, where c_x is the positive class, others are negative. (a) In EMA, all negative concepts' connections are demoted, resulting in lowering of all their scores (the arrows). (b) In OOZ, only the "offending" negatives may be demoted: the weight amount taken off of their connections is channeled to the positive concepts' connections. c_0, the "dummy" concept, is always offending, always reduced first.

3.1.2 Improving Accuracy of EMA via Margins

If we always invoke EMA, the scores that each feature computes best approximate the (moving average of) class conditional probabilities (given the feature is active). However, in many tasks, we care about simple accuracy. In this context, we want the positive class to be ranked highest most often, and we may not mind uncalibrated (inconsistent) probabilities. For instance, at the expense of added implementation complexity, the scores can be mapped to probabilities afterwards [20]. In [18], it is argued and shown in experiments that mistake driven updating via use of a margin can significantly improve accuracy. For example, mistake driven updating can down weigh the effects of redundant features. Given an instance x, after scoring, the current margin is defined as:

$$\delta = \min_{c \in C_x} s_c - \max_{c' \notin C_x} s_{c'} \tag{2}$$

If the margin δ falls below a threshold $\delta_m > 0$, we invoke the EMA version given in Figure 2. In this version, we first weaken all (active) connections, then re-score concepts, so that if an additional positive class falls below the margin, it's updated (the re-scoring is not necessary in the single-label setting).

3.2 Online Optimization of Slack Variables

In every update, EMA weakens the weights of all negative classes, while for improving ranking and thus accuracy, we need only lower those negative classes that scored higher than the positive(s). Lowering others' weights may unnecessarily break what has been learned before (*e.g.*, can lead to misclassifying previously correctly classified instances) It has been shown, in particular in the setting of support vector machine learning, that optimizing hinge loss, with proper regularization, yields superior generalization performance in terms of 0/1 accuracy compared to some other approaches. Here, we explore an online technique in which a hinge loss is improved in each update. The update becomes more complex than the EMA, and in particular requires information about the scores of (a subset of) the negative categories. Furthermore, to allow for controlling the outdegree, the update is constrained further, as we describe. This aspect helps avoid overfitting as well. We call the algorithm OOZ ("ooze") (for Online Optimization of z (*i.e.*, slack) variables). The high level difference between EMA and OOZ is shown in Figure 3. For the OOZ technique, we will only develop the single label setting, although we will briefly mention ideas for extending to multiple labels. Finally, there are a number of variations possible that will become apparent as we describe the approach. We are able to explore only a few here. OOZ is shown in Figure 4.

Slacks. The slack (or "helper") variable for instance i, denoted z_x, is defined as:

$$z_x = \min_{z \geq 0}(s_{c_x} - \max_{c \neq c_x} s_c + z) \geq \delta_m,$$

for a choice of a margin threshold $\delta_m, \delta_m > 0$. We motivated the use of a margin threshold for improved accuracy in [18]. The slack variable is just high enough for the inequality to hold. The value of the slack variable is the same as the hinge loss on a given instance. Ideally, we want every slack variable to stay at the minimum value of 0, and in general, we seek to minimize the sum of the values over the training set, equivalent to minimizing the hinge loss over the training set: Objective: minimize $\sum_{x \in S} z_x$, subject to certain constraints that include: $\forall i, z_x \geq 0$, $\forall i, j, w_{i,j} \geq 0$, and $\forall i, \sum_{j \geq 0} w_{i,j} \leq 1$. OOZ is an online update that attempts to optimize the above linear formulation.

In the online setting, on a training instance x, we compute the current value of its corresponding slack variable, which amounts to scoring concepts, or computing $\tilde{C}_x = x^T W$. If the slack is positive[1], we update (invoke OOZ). To lower the value, we can either increase the value of the positive class, and/or lower the value of the negative classes that have scored too high, in particular, the highest scoring negative class(es). We need to make sure a single update is not so extensive as to "forget" what has been learned before (a general constraint in online learning, *e.g.*, [12]), Or, another way put, we need to make sure the step taken along the gradient is not too big, but substantial enough to lead to learning at an adequate rate. Equally important, a new feature should not commit all its weight to the first concept in which it is invoked to update. The learning rate (or the step amount) controls this.

Sources of Weight. We want to constrain the total weight that any feature uses, in order to constrain the feature's out-degree, so the increase in the weight of the true positive class must come from some limited source. There are several choices. One candidate is the negative classes, in particular those that exceed the threshold point (the "offenders"). Shifting weights from these negative

[1]In case of multiple positive classes, the highest scoring class defines the slack, and gets all the boost in the update. If all have 0 score (initially), the boost is spread over all the positive classes.

```
OOZ_Update($x$, $C_x$, $c_x$, $\tilde{C}_x$, $\delta$, $\beta$) {
  $\beta \leftarrow$ min($\delta, \beta$) /* possibly lower learning rate */
  $\forall f \in x, \beta'_f \leftarrow$ Deduct_from_Free_Source($f$, $\beta x_f$, $c_x$)
   /* Recompute margin and remaining rate: */
    $\delta_s \leftarrow \sum_{f \in x}(\beta x_f - \beta'_f)x_f$. /* the increase in $s_{c_x}$ */
    $\beta \leftarrow \beta - \delta_s$. /* update desired improvement */
    $\delta \leftarrow \delta - \delta_s$. /* update margin */
  If ($\beta = 0$ or $\delta \geq \delta_m$ or $\tilde{C}_x = \emptyset$) return.
  $D \leftarrow$Compute_Offenders_and_Target_Deductions( $s_{c_x} - \delta$, $\beta$, $\tilde{C}_x$)
    $R \leftarrow D$. /* R is remainders, initialized to D */
    $\forall f \in x$, Shift_from_Negatives($f$, $x_f$, $\beta'_f$, $D$, $R$, $c_x$)
}

/* Take at most $r$ from free source. Returns what's left. */
Deduct_from_Free_Source($f$, $r$, $c_x$) {
  $boost \leftarrow \min(w_{f,0}, r)$ /* is any left in $w_{f,0}$? */
  $w_{f,c_x} \leftarrow w_{f,c_x} + boost$
  return ($r - boost$)
}

/* Compute Target Deduction Values */
Compute_Offenders_and_Target_Deductions($s_{min}$, $\beta$, $\tilde{C}_x$) {
  $Y \leftarrow \emptyset, D \leftarrow \emptyset$. /* 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 $Y$. /* s is a sorted array */
  $cum \leftarrow 0, n \leftarrow 1, i \leftarrow 0$.
  loop while ($cum \leq \beta$) {
    $\Delta \leftarrow \min(\beta - cum, s[i+1] - s[i])$
    $\forall c, s_c > s[i+1], D(c) \leftarrow D(c) + \frac{\Delta}{n}$
    $cum \leftarrow cum + \Delta$
    $n \leftarrow n + 1$
    $i \leftarrow i + 1$
  }
  return $D$
}

/* Shift Weights From Negatives, Respecting Target Deductions */
Shift_from_Negatives($f$, $x_f$, $r$, $D$, $R$, $c_x$) {
  $boost \leftarrow 0$.
  /* 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*/
    $shift \leftarrow \min(R(c), w_{f,c}, x_f D(c), r)$
    $boost \leftarrow boost + shift$
    $w_{f,c} \leftarrow w_{f,c} - shift$
    $R(c) \leftarrow R(c) - shift$
    $r \leftarrow r - shift$
   }
  $w_{f,c_x} \leftarrow w_{f,c_x} + boost$
}
```

Figure 4: Pseudo-code for OOZ and its subroutines. OOZ shifts weights from concept 0 (the dummy concept) and the offending negatives, to the positive. First, each feature subtracts from its free source. New margin is computed. If desired margin is not met, the set of offending concepts and how much to deduct from each is computed and given to the features to further shift weights appropriately.

classes yields a kind of weight conservation or more accurately weight bounding. However, for instance initially, there may not be a positively scoring negative class at all. The solution is to assume each feature also has access to a "free" but limited source of weight, $w_{f,0}$. A good way to think about it is that each feature is connected to a dummy always negative class, concept 0, and the first time the feature is seen during training, the connection has weight $w_{f,0} = 1$. Thus in every update of a given feature f, we can make the following distinctions regarding the source of weight to channel to the connection to the positive class w_{f,c_x}:

1. The free weight source (if connected, *i.e.*, if $w_{f,0} > 0$)
2. Concepts that scored too high (if connected)
3. Any negative concept (if connected)

Intuitively, it is best to first take from the free source, when available. This will have lowest impact on what has been learned before as the dummy concept is never a true positive. After that, if there still remains room for updating, the feature should deduct weights from those concepts that scored too high, if the feature is connected to them. And finally, the feature could take from any other negative concepts it's connected to. OOZ currently implements the first two options. We need to determine how much each feature may shift, and how much to remove from each type of negative concept.

Extent of Change and Improvement. As seen in Figure 4, each feature's connections is updated proportional to its activity level x_f. Given the overall desired improvement step β, we can verify that the change $\Delta_{f,c}$ in any connection of feature f is at most βx_f once OOZ is done: $\Delta_{f,c} \leq \beta x_f$. Thus, when vector x is l_2 normalized, the improvement in score of the positive class, $\Delta_{s_{c_x}}$, will be $\sum_{f \in x} x_f(\Delta_{f,c_x}) \leq \beta \sum_{f \in x} x_f^2 = \beta$. We can verify that the slack value may improve by at most 2β. Furthermore, again with every connection change $\Delta_{f,c} \leq \beta x_f$, and with l_2 normalization, it follows that the change in ΔW in weight matrix W in l_2 norm is also at most β: $||\Delta W||_2 = \sqrt{\sum_{f \in x}(\Delta_{f,c})^2} \leq \sqrt{\sum_{f \in x}(\beta x_f)^2} = \beta$. Therefore, OOZ as given obeys a norm constraint in changing the weight matrix.

Taking from the Free Source. As the free source is "free", one question is why should the weight amount taken from it be constrained in an update, at most $x_f \beta$ as given in OOZ? While taking a larger amount may have little impact on the relative ordering of other classes on previous instances (in the beginning, there are no previous instances to worry about), the intuition is that we don't want to over commit to a single update and instance. We don't want to give all the weight available to a feature to predict a single class. We want to see the feature active in several updates to best allocate its weight distribution. On a related consideration, we want to limit the influence of infrequent or newly seen features when predicting (see Section 3.2.1 below on further budgeting).

Weight Shifting. After taking weights from the free source (possibly 0), OOZ computes the remaining overall β left in the update, as well as β'_f for each feature f. If $\beta = 0$ or margin is met, the update is over. Otherwise, in the spirit of changing as little as possible while getting the most slack reduction, OOZ computes the *offending* (negative) concepts, those scoring above $s_{c_x} - \delta_m$. It also computes the target (desired) deduction amounts for each. An example explains the process best. Let c_1, c_2 and c_3 be negative classes with scores $s_{c_1} = 0.4, s_{c_2} = 0.3$ and $s_{c_3} = 2.1$, and let $\beta = 0.2$. The target deduction amounts are $D(c_1) = 0.15$, $D(c_2) = 0.05$, and $D(c_3) = 0$: as first the score of c_1 needs to be lowered to tie c_2 (0.1), then both are lowered, for 0.05 each, at which point β is exhausted. This manner lowers the slack the quickest. Observe that some offending classes may not be assigned a deduction. Next each active feature is sequentially processed, and deducts the minimum of $x_f D(c), w_{f,c}, R(c)$, and r from $w_{f,c}$, wherein $R(c)$ is initially $D(c)$, but goes down as features deduct weights from their connection to c. In these deductions, some connections can be zeroed, limiting the feature degrees.

With our current implementation, focusing on sources 1 and 2, a feature may still have some allowed change left. For instance, one feature may only be connected to offending c_1 and anther to c_2. Multiple calls to further deduct can help reduce the allowed change (or revisiting the same instance in multiple passes). However, we can verify that OOZ as implemented makes positive progress (strictly lowers the slack in an update): if no feature is connected to a free source, then there must be offending classes with positive deduction target amounts, and at least one feature connected to one.

3.2.1 Further Budgeting

One budget constraint is that the free source for every feature begins at 1.0 and can only decrease over time, thus total outgoing weight $w_f \leq 1$ throughout training. On finite sets, and subsequent passes on the same training set, infrequent features can develop strong connections by taking weight from the free source in every pass, leading to overfitting. Imagine every instance having a unique feature. To avoid this, we can impose stricter budget constraints. After pass k, for some choice of $k \geq 1$, we can remove the free source: setting $\forall f, w_{f,0} \leftarrow 0$, effectively limiting the remaining optimization to weight shifting. We will see that this option can improve accuracy on some data sets.

3.3 Discussion

OOZ takes time in $\tilde{O}(|x|d)$, involving several passes over the connections of active features, and some sorting and hashing for efficiency. As long as the average outdegree of features d remain relatively low, OOZ will be fast. Since the learning rate β may be lowered in an update (when desired margin is almost met, or after the use of free source), it is possible that the weight increments and deductions may become very low in some updates. Explicit weight removal under a minimum threshold, and their transfer to the connection of the true positive, may help efficiency.

For the multilabel setting, one approach is to introduce multiple slack variables per instance, one for each positive class, and attempt to minimize the slack sum over all instances as before. Just as in the case of EMA, it is intuitive that features that tend to be in instances with multiple labels should be able to develop total weights greater than 1.0 (*e.g.*, when compared to features that only occur in single-label instances). An ideas to handle this is allow access to k free sources (each initially at 1.0) when an instance has $k \geq 1$ positive classes in an update. We leave exploring this avenue to future work.

4. EXPERIMENTS

Figure 5 displays our data sets. We use the same data sets[2] and

[2]Except for Advertisements, which is a Yahoo! internal data set.

Data Sets	$\lvert S\rvert$	$\lvert F\rvert$	$\lvert C\rvert$	$E_x\lvert F_x\rvert$	$E_x\lvert C_x\rvert$
Reuters-21578	$9.4k$	$33k$	10	80.9	1
20 Newsgroups	$20k$	$60k$	20	80	1
Industry	$9.6k$	$69k$	104	120	1
Reuters RCV1	$23k$	$47k$	414	76	2.08
Web	70k	685k	14k	210	1
Jane Austen	749k	299k	17.4k	15.1	1

Figure 5: Data sets: $|S|$ is number of instances, $|F|$ is the number of features, $|C|$ is the total number of classes, $E_x|x_f|$ is the average (expected) number of unique active features per instance, and $E_x|C_x|$ is the average number of positive classes per instance.

the same preprocessing and experimental set up as in [18]. Web refers to web page classification into Yahoo! directory of topics. The others are benchmark categorization [15, 22]: Reuters are news articles, and news groups is news postings [14]. The last data set, Austen, is a word prediction task [24], the concatenation of 6 online novels of Jane Austen (obtained from project Gutenberg (http://www.gutenberg.org/). Here, each word is its own class, and there is no class hierarchy. Each word's surrounding neighborhood of words, 3 on one side, 3 on the other, and their conjunctions constituted the features (about 15 many). We include this prediction task to further underscore the potential of scalable learning.

All instances are l_2 (cosine) normalized (unless otherwise mentioned), using standard features (unigrams, bigrams,..), obtained from publicly available sources (*e.g.*, [15, 22]). To simplify evaluation for the case of the taxonomy, we used the lowest true category(ies) in the hierarchy the instance was categorized under. In many practical applications such as personalization, topics at the top level are too generic to be useful. All but one data set (Reuters RCV1) is single-label per instance.

In the comparisons, we report on the average performance over ten trials. In each trial a random 10% of the data is held out. The exception is the newsgroup data set where we use the ten 80-20 train-test splits provided by Rennie *et. al.* [22].

In our comparisons, we report on some of the results of [18] as well. On the first 3 small sets, we report their results when comparing to the one-versus-rest technique (*e.g.*, [23]) employing binary classifier learning: perceptrons, committee of 10 perceptrons, and linear SVMs (a fast algorithm/implementation [11]) using at least two regularization parameters, $C = 1$ and $C = 10$.[3] On the two larger ones, the binary classifiers are deployed in a top-down method (*e.g.*, [16]). The original feature-focus explored in [18] is similar to EMA updating, except that it assumes stationarity, and an effective rate of 1.0 was used. See [18] for details. In addition to the comparison with previous results, we also compare with other methods that stem from recent advances in efficient online learning. Multi-class Multi-label Perceptron (MMP) [2] and Passive Aggressive (PA) [1] are both families of prototype-based classification algorithms, wherein each category has its own associated prototype weight vector. MMP updates all the prototypes in learning, while PA only updates two and thus can be faster. To compare with strong baseline methods, we intentionally selected

[3]Often these two suffice for text, and we have further experimented with others such as $C = 100, C = 20$, and so on, on the smaller data sets, and have observed no improvement in accuracy

the variant of each that demonstrated the best result in the original work. Specifically, we used the uniform update scheme and the $IsErr$ ranking loss for MMP. PA-II was chosen for its capability to better handle label noise, with the introduction of slack variables and aggressive parameter C. For OOZ and EMA, during classification, the top 25 connected classes of every feature are scored. For EMA, w_{min} was set to 0.005. For each algorithm we tested, we experimented with a few parameter settings (margin threshold and learning rate) to obtain good accuracy results. Experiments were run on an Intel quad-core Q6600 (2.4GHz) CPU with 4GB of memory.

Table 4 gives the results. We observe that OOZ improves on accuracy across the data sets, while EMA performs better than (original) FF in some cases. This gain in accuracy is achieved at cost of some loss in efficiency. In general, the original FF quickly converges (within very few passes), but the accuracy achieved may not be the best (*e.g.*, Figure 6). The standard deviations are relatively low (third to second decimal point), and representatives are given in subsequent figures and tables. OOZ tends to outperform PA-II, which in turn outperforms MMP. OOZ also scores significantly higher $R1$ and $R5$ than the one-versus-rest and top-down binary classifier based methods using linear SVM or perceptron methods in several data sets. As explained in [18], binary classifier training is often more appropriate for ranking instances with respect to a fixed class, versus our problem of scoring classes given a single instance.

We are particularly interested in comparing the accuracy with the original FF algorithm. We boldfaced the cells in Table 4 where OOZ/EMA win over FF in at least 9 out of the 10 trials (95% confidence from the sign test). Particularly in the large datasets, we observe substantial gain in classification accuracy of OOZ compared to FF. For instance, there is relatively 14% improvement in $R1$ in the web dataset and 5% in the RCV1 dataset.

Finally, in terms of efficiency, we observe the benefits of indexing. Top-down method using SVMs took more than a day and it was stopped on the Web data [18]. Similarly the PA and MMP methods take hours on Web, and we stopped them. We also see that the space used by the index is often substantially less than other methods.[4]

4.1 A Plot of Convergence

Figure 6 shows classification accuracy of the three indexing algorithms OOZ, EMA and FF as a function of the number of passes in the RCV1 dataset. FF reaches its peak of performance with R1 approaching 0.79 at pass 2 and 3, but slowly declines afterwards exhibiting a possible sign of overfitting. On the other hand, the accuracy of EMA and OOZ levels off once they reach the peaks. OOZ converges slower than EMA, but achieves 2% higher classification accuracy. Similar learning patterns are also observed in other datasets, though we omit the figures for space constraints.

[4]On the smaller datasets, the space consumption of the SVMs or perceptron committees (when one-versus-rest was used) were estimated optimistically by reporting the size of a perceptron implemented via a sparse space-efficient method [18] (the plus signs denote estimates). Otherwise, if all features are explicitly represented, space consumption would simply be $|F||C|$, *i.e.*, the product of the total number of features in the problem and the number of classes.

Classifier	R_1	R_5	T_{tr}	$\bar{e}$	$\lvert\mathcal{I}\rvert$
Reuters-21578 (10 classes)					
OOZ ($\beta: 0.01, \delta_m: 0.2, p: 8$)	**0.912**	**0.999**	1.6s	5	87k
EMA ($\beta: 0.1, \delta_m: 1.0, p: 5$)	**0.892**	0.996	2.5s	5	87k
FF ($\delta_m: 0.5, p: 1$)	0.884	0.997	0.7s	5	73k
MMP	0.879	0.995	1s	7	116k
PA-II ($C: 1$)	0.900	0.994	0.7s	8	62k
SVM ($C: 1$)	0.906	0.998	11s	10	74k+
Newsgroup (20 classes)					
OOZ ($\beta: 0.02, \delta_m: 0.4, p: 15$)	0.862	0.986	72s	14	226k
EMA ($\beta: 0.6, \delta_m: 7.0, p: 30$)	**0.867**	0.988	85s	13	205k
FF ($\delta_m: 0.5, p: 1$)	0.865	0.987	3.7s	10	171k
MMP	0.803	0.958	6.2s	17	386k
PA-II ($C: 1$)	0.840	0.966	5.7s	18	163k
SVM ($C: 1$)	0.852	0.975	92s	20	189k+
Industry (104 classes)					
OOZ ($\beta: 0.01, \delta_m: 0.2, p: 30$)	**0.903**	**0.952**	180s	21	230k
EMA ($\beta: 0.1, \delta_m: 1.0, p: 30$)	0.873	0.933	150s	20	199k
FF ($\delta_m: 0.5, p: 3$)	0.886	0.949	16s	16	196k
MMP	0.776	0.893	135s	93	1.6M
PA-II ($C: 1, p: 10$)	0.862	0.930	230s	87	266k
SVM ($C: 10$)	0.872	0.933	235s	104	330k+
Reuters RCV1 (414 classes)					
OOZ ($\beta: 0.02, \delta_m: 0.1, p: 8$)	**0.855**	**0.975**	72s	21	163k
EMA ($\beta: 0.02, \delta_m: 0.05, p: 20$)	**0.815**	**0.958**	400s	23	285k
FF ($\delta_m: 0.1, p: 4$)	0.787	0.952	24s	13	220k
MMP	0.840	0.979	143s	375	383k
PA-II ($C: 1$)	0.847	0.966	67s	228	281k
Perceptron	0.621	0.815	70s	38	760k
Committee	0.769	0.918	750s	36	760+
SVM ($C: 1$)	0.783	0.939	520s	36	4M
Web (14k classes)					
OOZ ($\beta: 0.05, \delta_m: 0.05, p: 4$)	**0.402**	**0.601**	31m	23	1.9M
EMA ($\beta: 0.1, \delta_m: 0.1, p: 20$)	0.335	0.496	2h	22	1.7M
FF ($\delta_m: 0.0, p: 2$)	0.352	0.576	128s	8	1.5M
Perceptron	0.098	0.224	1h+	250	14M
Committee	0.207	0.335	12h+	190	14M+
Jane Austen (17k classes)					
OOZ ($\beta: 0.1, \delta_m: 0.1, p: 2$)	0.275	0.477	192s	22	1.6M
EMA ($\beta: 0.1, \delta_m: 0.1, p: 20$)	**0.284**	**0.485**	6m	22	1.6M
FF ($\delta_m: 0.0, p: 1$)	0.272	0.480	40s	8	1.5M

Table 1: Performance of different the learners on six datasets over 10 trials. T_{tr} is the training time (s=seconds, m=minutes and h=hours), $\bar{e}$ is the average number of edges touched per feature of a test instance (during classification), $|\mathcal{I}|$ denotes the number of (nonzero) weights in the classifier ($k = 10^3$, $M = 10^6$). Parameters used for each classifier are given in parentheses after the corresponding method's name.

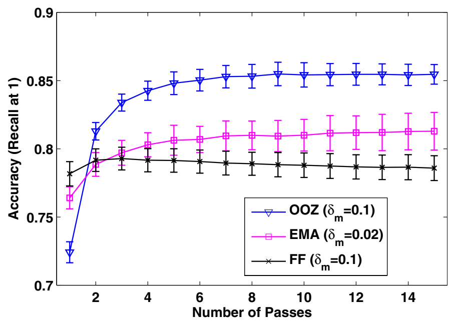

Figure 6: RCV1: Accuracy (R1) of OOZ, EMA and FF against the number of passes.

4.2 The Impact of Learning Rate

We investigate the impact of the learning rate on prediction performance of EMA and OOZ. Figure 7 shows the learning curve of EMA on the Industry dataset with β set to 0.02, 0.05, 0.1 and 0.2 (margin of 0.3). As we increase β, $R1$ converges faster, and training time is faster (not shown). However, the excessively aggressive learning rate $\beta = 0.2$ leads to inferior performance. Setting β to 0.1 in this dataset, on the other hand, yields both faster convergence and best $R1$ score. We observed similar learning patterns for OOZ on the RCV1 dataset (Figure 8, $\beta = \{0.01, 0.02, 0.05, 0.1\}$) (margin set to 0.1). Setting β to 0.05 in this dataset, $R1$ peaks at 0.855 in the fourth pass while the aggressive ($\beta = 0.1$) and slow ($\beta = 0.01$) learning rates result in worse accuracy.

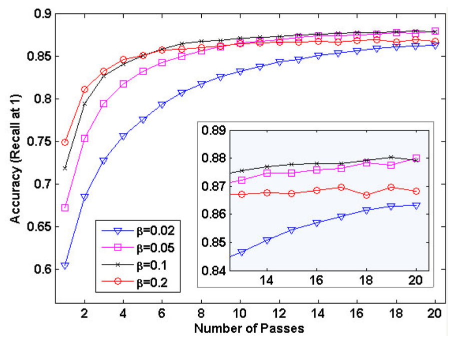

Figure 7: Accuracy as a function of the number of passes with different learning rate β for EMA in the Industry dataset. The tails of the curves are shown in the inset.

4.3 Shifting Weights and the Removal of Free Weight Source

Here, we investigate the effects, on the OOZ algorithm, of shifting weights from negative classes, lowering the learning rate, and as an extension, imposing a stricter budget than 1.0, *i.e.*, removing the free source after a certain pass (setting $w_{f,0}$ to 0) to avoid potential overfitting. As illustrated in Table 4.3, weight shifting is important, improving R_1 by 7.5% and R_5 by 5.2%. If the free weight source is removed after two passes, R_1 and R_5 can be further improved by 2-3%. The option to lower the learning rate (when desired margin is smaller) is also important for OOZ, without which the accuracy degraded by 9%.

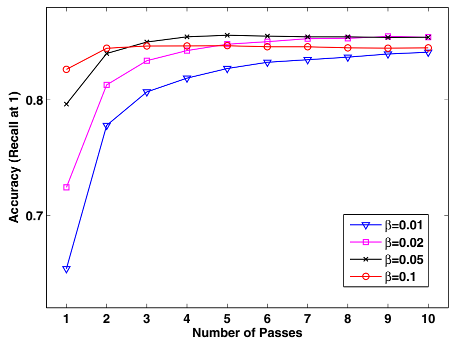

Figure 8: Accuracy as a function of the number of passes with different learning rate β for OOZ in the RCV1 dataset.

Option	R_1	R_5
OOZ w/o WS	0.774 [.010]	0.910 [.009]
OOZ w/o SR	0.832 [.008]	0.957 [.004]
OOZ w/o LLR	0.761 [.008]	0.943 [.002]
OOZ (pass 2)	0.855 [.009]	0.975 [.004]

Table 2: Effects of weight shifting (WS), weight source removal (SR), lowering learning rate (LLR) of OOZ in the RCV1 dataset. Standard deviation is shown in square brackets.

5. RELATED WORK AND DISCUSSION

The work in [19, 18] motivated many-class learning problems and provided evidence that very efficient linear learning methods, via "concept indexing", exist. Our work further explores the approach and expands and improves the methods.

There are a variety of multiclass techniques, but past research has not focused on large-scale many-class learning, and in particular the problem of efficient classification. The multiclass perceptron and related algorithms (*e.g.*, [4, 2, 1]) have similar ranking goals. These methods are best viewed as learning "prototypes" (weight-vectors), one for each class [5]. On an update, certain prototypes' weights for the active features get promoted or demoted. We will refer to them as *prototype methods*, in contrast to our *predictor-based* or feature-focus methods. In terms of the weight matrix, where features are rows and concepts are columns, indexing or feature-focus approaches work on the rows, while prototype methods operate on the columns (Figure 9). L1 regularization in particular yields prototypes with fewer nonzero weights compared to L2 regularization (*e.g.*, see LASSO in [10]). For the setting of many-class learning, it is possible that scalable discriminative methods other than predictor-based ones exist as well. It is conceivable that some kind of prototype regularization may render prototype methods efficient for large-scale many-class learning, but here are some difficulties that we see: Computing prototypes while simultaneously preserving efficiency may be more challenging as constraining the indegree of classes may not guarantee small feature outdegrees (important for efficient classification and updates) and different classes can require widely different and large indegrees for good accuracy, as some classes are more typical or generic than others [18]. Furthermore, updates that involve prototype weight normalization or sparsity control require extra data structures (*e.g.*,

[5]One method studied theoretically normalizes the whole matrix.

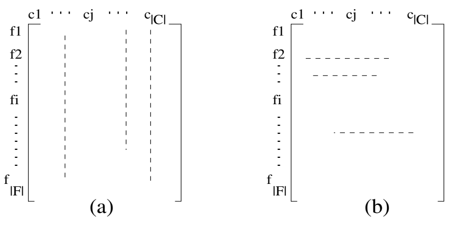

Figure 9: In learning a weight matrix, in which rows are features and columns are classes, prototype methods, given an instance, operate on the columns, for example in normalizing columns weights and regularizing. Feature-focus methods work on the rows, in normalizing or budgeting feature outdegrees given an instance.

pointers) for efficient prototype processing, in addition to the space required for the index (the feature to category mapping), complicating the approach. Note however that in developing OOZ we used similar general techniques developed previously, in for instance optimizing a hinge loss in multiclass setting (*e.g.*, [12, 1]), but subject to different constraints (feature outdegree budgets). Variations of OOZ, including batch optimization, and allowing for learning negative weights, may also prove fruitful.

Moving averages are used in time series analysis and prediction, for instance in ARMA and ARIMA models [6]. A Boolean single-feature version of EMA has also been used for the task of Unix command prediction for personalization [3, 13], to adapt to non-stationarities, wherein it was called α-updating. Temporal difference learning of state values utilized in reinforcement learning also has the form of EMA [26]. Feature-focus algorithms, as predictor-based methods, have similarities with additive models and tree-induction [10]. They may be thought to be in the family of so-called expert (opinion or forecast) aggregation and learning algorithms (*e.g.*, [21, 8, 9])). Previous work has focused on learning mixing weights for the experts, while we have focused on how the experts may efficiently update their votes.

6. SUMMARY

We developed two feature focus methods for multiclass learning, and demonstrated their potential for improved accuracy while maintaining efficiency in many-class settings. We plan to further study and advance the techniques, and explore applications. These algorithms extend the repertoire of tools available for large-scale learning and data mining.

Acknowledgments

Many thanks to the reviewers for valuable feedback and suggestions. The authors benefited from discussions with the machine learning and data mining groups at Yahoo! and researchers in the SRI computing division, including Jeffrey Davitz for suggesting the ARIMA model, and Sathiya Keerthi, John Langford, and Vijay Krishnan. This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. FA8750-07-D-0185/0004. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the Defense Advanced Research Projects Agency (DARPA), or the Air Force Research Laboratory (AFRL).

7. REFERENCES

[1] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer. Online passive-aggressive algorithms. *JMLR*, 7, 2006.

[2] K. Crammer and Y. Singer. A family of additive online algorithms for category ranking. *JMLR*, 3, 2003.

[3] B. D. Davison and H. Hirsh. Predicting sequences of user actions. In *AAAI-98/ICML'98 Workshop on Predicting the Future: AI Approaches to Time Series Analysis*, 1998.

[4] R. Duda, P. Hart, and D. Stork. *Pattern Classification*. John Wiley & Sons, 2 edition, 2001.

[5] S. Dumais and H. Chen. Hierarchical classification of web content. In *SIGIR*, 2000.

[6] B. S. Everitt. *Cambridge Dictionary of Statistics*. Cambridge University Press, 2nd edition edition, 2003.

[7] D. A. Forsyth and J. Ponce. *Computer Vision*. Prentice Hall, 2003.

[8] Y. Freund, R. Schapire, Y. Singer, and M. Warmuth. Using and combining predictors that specialize. In *STOC*, 1997.

[9] C. Genest and J. V. Zidek. Combining probability distributions: A critique and an annotated bibliography. *Statistical Science*, 1(1):114–148, 1986.

[10] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer-Verlag, 2001.

[11] S. Keerthi and D. DeCoste. A modified finite Newton method for fast solution of large scale linear SVMs. *JMLR*, 2006.

[12] J. Kivinen and M. Warmuth. Exponentiated gradient versus gradient descent for linear predictors. *Inf. and Comput.*, 1997.

[13] B. Korvemaker and R. Greiner. Predicting UNIX command lines: Adjusting to user patterns. In *AAAI/IAAI*, 2000.

[14] K. Lang. Newsweeder: Learning to filter netnews. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 331–339, 1995.

[15] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. RCV1: A new benchmark collection for text categorization research. *JMLR*, 5:361–397, 2004.

[16] T. Liu, Y. Yang, H. Wan, H. Zeng, Z. Chen, and W. Ma. Support vector machines classification with very large scale taxonomy. *SIGKDD Explorations*, 7, 2005.

[17] O. Madani. Exploring massive learning via a prediction system. In *AAAI Fall Symposium Series: Computational Approaches to Representation Change During Learning and Development*, 2007.

[18] O. Madani and M. Connor. Large-scale many-class learning. In *SIAM Conference on Data Mining (SDM)*, 2008.

[19] O. Madani, W. Greiner, D. Kempe, and M. Salavatipour. Recall systems: Efficient learning and use of category indices. In *AISTATS*, 2007.

[20] O. Madani and J. Huang. On updates that constrain the features' connections during learning. Technical report, SRI International, AI Center, 2008. In preparation.

[21] C. Mesterharm. A multi-class linear learning algorithm related to Winnow. In *NIPS*, 2000.

[22] J. Rennie, L. Shih, J. Teevan, and D. Karger. Tackling the poor assumption of Naive Bayes text classifiers. In *ICML*, 2003.

[23] R. Rifkin and A. Klautau. In defense of one-vs-all classification. *JMLR*, 5, 2004.

[24] R. Rosenfeld. Two decades of statistical language modeling: Where do we go from here? *IEEE*, 88(8), 2000.

[25] F. Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys*, 2002.

[26] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 1998.