A Family of Online Boosting Algorithms

Boris Babenko
University of California, San Diego
bbabenko@cs.ucsd.edu

Ming-Hsuan Yang
University of California, Merced
mhyang@ucmerced.edu

Serge Belongie
University of California, San Diego
sjb@cs.ucsd.edu

Abstract

Boosting has become a powerful and useful tool in the machine learning and computer vision communities in recent years, and many interesting boosting algorithms have been developed to solve various challenging problems. In particular, Friedman proposed a flexible framework called gradient boosting, which has been used to derive boosting procedures for regression, multiple instance learning, semi-supervised learning, etc. Recently some attention has been given to online boosting (where the examples become available one at a time). In this paper we develop a boosting framework that can be used to derive online boosting algorithms for various cost functions. Within this framework, we derive online boosting algorithms for Logistic Regression, Least Squares Regression, and Multiple Instance Learning. We present promising results on a wide range of data sets.

1. Introduction

Online learning is a paradigm in which a learning algorithm is presented with one example at a time. This setting is more challenging than learning in batch mode, but carries with it many benefits. For example, some applications may have a continuous stream of data (e.g., visual tracking). In other scenarios, the entire training data set cannot be fit into the memory of a machine, either because the amount of data is enormous, or because the machine has tight memory constraints. Since memory operations are more computationally expensive than CPU operations, using less memory can also lead to faster execution. Lastly, learning a model in an online fashion can be more efficient in many cases because the model can always be updated if more data becomes available, rather than having to retrain a model from scratch.

Much work has been done on analyzing online learning algorithms, as well as taking popular batch learning algorithms and creating online versions (e.g., online SVM [7], incremental decision trees [28], etc.). In this paper we focus on the boosting family of learning algorithms. In [23] Oza presents an online variant of the popular AdaBoost [11] algorithm, and proves that this variant converges to the same solution as the original batch algorithm, given an infinite number of examples. Since then, some improvements have been made to his work, and some interesting applications in computer vision have been developed [14, 21]. However, Oza's online algorithm can only be applied to classification problems, and many other flavors of batch boosting exist that solve a variety of interesting problems. Although a couple other flavors of boosting have been adapted to the online scenario, such as Multiple Instance Learning [3] and Semi Supervised Learning [15], it is difficult to see the relationship between these algorithms or how to extend these techniques to other problems. In this paper we propose a single framework that allows such extensions to be made, and present three different online boosting algorithms derived using this framework (although many more are possible). We present a wide range of experimental results for our algorithms and show that they converge to give similar performance as standard batch boosting methods, and outperform traditional online learning methods.

The paper is structured as follows: in Section (2) we review the relevant boosting literature; in Section (3) we present our framework; in Section (4) we use our framework to develop three online boosting algorithms, and present empirical results in Section (5); finally, in Section (6) we discuss other related work.

2. Boosting as Greedy Optimization

The term "boosting" refers to the process of taking a "weak" learning algorithm (classification or regression) and boosting its performance by training many classifiers and combining them in some way [27]. This is particularly useful in cases where it is difficult to design a high accuracy classifier, but easy to come up with simple decision rules that perform slightly better than random guessing. Generally, the model for the final "strong" classifier is a weighted voting or linear combination of the weak classifiers. Formally, suppose we are given a training data set $\{x_i, y_i\}_1^N$ where $x \in \mathcal{X}$ and $y \in \mathcal{Y}$ (e.g., $\mathcal{X} = \mathbf{R}^d$, $\mathcal{Y} = \{+1, -1\}$). We seek to learn a function $H : \mathcal{X} \rightarrow \mathcal{Y}$ by combining multiple weak learning functions h, which come from some hypothesis class $\mathcal{H}$ and are parameterized by some vector a. The form of the strong model is

$H(x) = \sum_{m=1}^{M} \alpha_m h(x; a_m)$, where α_m is a scalar weight. In recent years, many boosting algorithms have been developed to solve various problems. To develop a boosting algorithm, one must first define a loss function $\mathcal{L}(H|\{x_i, y_i\}_1^N)$, which will generally depend on the application. For classification, perhaps the most famous boosting algorithm is AdaBoost [11], which aims to minimize the exponential loss: $\mathcal{L} = \sum_i \exp\{-y_i H(x_i)\}$. Ideally, we would solve for all the parameters of this model simultaneously. However, such optimization would not be practical. For this reason most boosting algorithms search for optimal parameters in a greedy fashion, adding one weak learner at a time. Let $H_{m-1} = \sum_{t=1}^{m-1} \alpha_t h(\cdot; a_t)$ be the strong model made up of the first $(m-1)$ weak learners. We now want to update this model by adding one more weak learner with the goal of minimizing the loss of the resulting strong model:

$$a_m = \underset{a}{\operatorname{argmin}}\, \mathcal{L}(H_{m-1} + \alpha_m h(\cdot; a)|\{x_i, y_i\}_1^N) \tag{1}$$

The boosting algorithm thus rests on our ability to solve the above equation. For certain loss functions (e.g., exponential loss) we can solve this equation in closed form. However, in other interesting cases, there are no closed form solutions. Some methods have been proposed to solve this equation in a more generalized manner. A particularly useful framework for doing so was proposed by Friedman [12] (a similar framework was described in [22]), and has been used to develop batch boosting algorithms for a variety of problems such as regression [31, 12], multiple instance learning [29], semi-supervised learning [16], etc. Friedman suggests picking parameter a_m by solving $a_m = \arg\min_a \frac{1}{2} \sum_i \left(g_{m-1}(x_i) - h(x_i; a)\right)^2$, where $g_{m-1}(x) = -\frac{\partial \mathcal{L}}{\partial H(x)}|_{H_{m-1}(x)}$. If we think of the high dimensional space where every axis is the value of $H(x_i)$ (what is referred to as the "function space"), and the loss as a function of that space, $g_{m-1}(x)$ points in the direction of greatest change. We would like to move in this direction to further minimize loss, but we can only move in this space by adding another weak function $h(\cdot; a_m)$ onto H. This gives us a limited number of directions to move in. For this reason Friedman suggests finding an h that is as close as possible to g_{m-1}. Note that to compute g_{m-1} we need to look at the entire training data set; Friedman's approach therefore results in a batch learning algorithm.

The goal of this paper is to formulate a framework that is as flexible as Friedman's approach, but works in an online setting, where examples are acquired one at a time, and are not stored.

3. Online Boosting

3.1. Stochastic Gradient Descent

Before we begin to discuss online boosting, we briefly review stochastic gradient descent as it forms the basis of our approach. Suppose we would like to minimize the loss of some function f given some data points $\mathcal{L}(f(\cdot; a)|\{x_i, y_i\}_1^N)$ by finding the optimal parameter vector a. Perhaps the simplest way to achieve this would be using gradient descent updates: $a^{(t+1)} = a^{(t)} - \eta_t \frac{\partial \mathcal{L}(f(\cdot;a)|\{x_i,y_i\}_1^N)}{\partial a}|_{a^{(t)}}$, where t specifies the iteration. This is a batch learning procedure since the loss is defined over the entire data set. A common online variant of gradient descent is called stochastic gradient descent [4]. Here the assumption is that the loss over the entire training data can be expressed as a sum of the loss for each point. That is, $\mathcal{L}(f(\cdot; a)|\{x_i, y_i\}_1^N) = \sum_i \mathcal{L}(f(\cdot; a)|x_i, y_i)$. The update rule is then[1]: $a^{(i+1)} = a^{(i)} - \eta_i \frac{\partial \mathcal{L}(f(\cdot;a)|x_{i+1},y_{i+1})}{\partial a}|_{a^{(i)}}$. Notice that instead of computing the gradient of entire loss, here we compute the gradient with respect to just one data point. This update is repeated for every point in the data set, sometimes with several passed through the data. Many online algorithms such as the perceptron [25], backpropagation [26], and least means squares [30] are based on the idea of stochastic gradient descent.

We note that in almost all cases, the loss functions used in boosting can be split into a sum of terms, each of which depends on only one example. Thus, for the remainder of the paper we will assume this is the case, and will use the following shorthand notation: $\mathcal{L}(H|\{x_i, y_i\}_1^N) = \sum_i \mathcal{L}_i(H)$, where $\mathcal{L}_i(H) = \mathcal{L}(H|x_i, y_i)$.

3.2. Stochastic Boosting

Recall that in boosting, when training the m^{th} weak learner, our goal is to minimize (1). To make the problem simpler, suppose the weight α_m is absorbed into the parameter vector a_m, and that the function $h(\cdot; a)$ can return real values. Furthermore, assume that h is differentiable with respect to the parameter vector a. This allows us to use stochastic gradient descent to directly minimize $\mathcal{L}(H_{m-1} + h(\cdot; a_m))$ in an iterative manner, leading to the batch boosting procedure we call Batch Stochastic Boosting (BSB), shown in Algorithm 1. Note that for simpler notation we will not show the superscript above a_m indicating the iteration of each update.

The BSB algorithm updates each weak learner with all data points. We can change the flow of the algorithm by instead updating *all* weak learners for *each* data point (*i.e.* flipping the order of the two **for** loops). This results in the Online Stochastic Boosting algorithm (OSB) shown in the right column.

The flow of this algorithm is similar to that of Oza's Online AdaBoost [23], although the actual algorithms differ. Notice that if the data does not fit in memory, in the batch

[1] In both versions of the gradient descent procedure, η_i is a learning rate parameter. Most commonly this is set either to some small constant, or to a decaying rate such as $\eta_i = c/(i+1)$, where c is a constant. We have found empirically that the latter produces better results, and have used this setting in all algorithms we present.

Algorithm 1 Batch Stochastic Boosting (BSB)

Input: Dataset $\{x_i, y_i\}_{i=1}^N$

1: Initialize $a_1, a_2 ... a_M$ randomly
2: **for** $m = 1$ to M **do**
3: **for** $i = 1$ to N **do**
4: Update $a_m \leftarrow a_m - \eta_i \frac{\partial}{\partial a} \mathcal{L}_i(H_{m-1} + h(\cdot; a))|_{a_m}$
5: **end for**
6: **end for**

Algorithm 2 Online Stochastic Boosting (OSB)

Input: Dataset $\{x_i, y_i\}_{i=1}^N$, available one at a time

1: Initialize $a_1, a_2 ... a_M$ randomly
2: **for** $i = 1$ to N **do**
3: **for** $m = 1$ to M **do**
4: Update $a_m \leftarrow a_m - \eta_i \frac{\partial}{\partial a} \mathcal{L}_i(H_{m-1} + h(\cdot; a))|_{a_m}$
5: **end for**
6: **end for**

case we are forced to load each example multiple times, where as in the online case we can load each example just once. Although the time complexity of both online and batch algorithms are the same, the computational time differs significantly. Since memory operations are more expensive than CPU operations, using fewer loads from memory can lead to significant gain in run time. Furthermore, the OSB algorithm is able to learn in an online manner, making it suitable for interactive applications.

3.3. Discussion

We give some intuition of the update steps in the batch and online boosting algorithms. First, let's write out the update rules for the online and batch versions of the algorithms:

BATCH
$a_m^{(i+1)} \leftarrow a_m^{(i)} - \eta_i \frac{\partial}{\partial a} \mathcal{L}_i \Big(\sum_{t=1}^{m-1} h(\cdot; a_t^{(N)}) + h(\cdot; a) \Big)|_{a_m^{(i)}}$
ONLINE
$a_m^{(i+1)} \leftarrow a_m^{(i)} - \eta_i \frac{\partial}{\partial a} \mathcal{L}_i \Big(\sum_{t=1}^{m-1} h(\cdot; a_t^{(i+1)}) + h(\cdot; a) \Big)|_{a_m^{(i)}}$

Note that the only difference is that when updating the m^{th} parameter, in the batch version all previous weak learners are in their final N^{th} state $a_t^{(N)}$, while in the online version they are in the $(i+1)^{th}$ state. Another observation is that for a_1 the update rule is identical since the summation term drops out. Suppose that after seeing k examples, a_1 converges (that is, $\frac{\partial}{\partial a} \mathcal{L}_i(h(\cdot; a))|_{a_1^{(k)}} = 0$). This means that for a_2 the update rule in the online version becomes identical to the batch version after k examples. It does not imply that a_2 will converge to the same vector in both the online and batch versions, because the starting conditions are different. However, it gives us the intuition that both the online and batch algorithms are trying to optimize the same thing. We note that even if the loss function is convex with respect to the final classifier H, this does not imply that there is a unique solution in terms of weak parameters $(a_1, a_2, .. a_M)$ because we can change the order of the weak learners and get an identical H. This makes it challenging to derive a formal proof of convergence without making additional strict assumptions. Therefore, rather than pursuing theoretical justification, in this paper we provide strong empirical evidence, which shows that our online algorithms converge to good performance quickly.

4. Applying Online Stochastic Boosting

In this section we derive online boosting procedures for different problems using our framework. In all cases we will simply show the update rule for each problem, which can then be plugged into Step 4 of Algorithm 1 or 2.

4.1. Logistic Regression Boosting

Our first example is with binary classification. For this task, the binomial log likelihood is a commonly used loss function:

$$\mathcal{L}(H|\{x_i, y_i\}_1^N) = - \sum_{i=1}^{N} y_i \log \sigma(H) + (1 - y_i) \log(1 - \sigma(H))$$

where $\sigma(\cdot)$ is the sigmoid function. To fill in Step 4 of the algorithms, we first need to solve for the following derivative:

$$\frac{\partial \mathcal{L}_i(H_{m-1} + h(\cdot; a))}{\partial a} = \Big(\sigma(H_{m-1}(x_i) + h(x_i; a_m)) - y_i \Big) \frac{\partial h(x_i; a)}{\partial a}$$

This leads to the following update rule:

UPDATE RULE
$a_m \leftarrow a_m - \eta_i \Big(\sigma(H_{m-1}(x_i) + h(x_i; a_m)) - y_i \Big) \frac{\partial h(x_i; a)}{\partial a} |_{a_m}$

4.2. Least Squares Regression Boosting

We now convert the least squares regression boosting algorithm [12] into the online setting. The loss function is:

$$\mathcal{L}(H|\{x_i, y_i\}_1^N) = \sum_{i=1}^{N} \frac{1}{2} \Big(y_i - H(x_i) \Big)^2$$

Again, we take the derivative with respect to a:

$$\frac{\partial \mathcal{L}_i(H_{m-1} + h(\cdot; a))}{\partial a} = (H_{m-1}(x_i) + h(x_i; a) - y_i) \frac{\partial h(x_i; a)}{\partial a}$$

We can now write down the update rule to fill in both algorithms.

UPDATE RULE

$$a_m \leftarrow a_m - \eta_i(H_{m-1}(x_i) + h(x_i; a_m) - y_i)\frac{\partial h(x_i;a)}{\partial a} \mid_{a_m}$$

4.3. Multiple Instance Boosting

The Multiple Instance Learning (MIL) problem is an extension of binary classification which was introduced in [9]. The basic idea of this paradigm is that rather than having labeled instances in the training data, one is given labeled sets of instances, or "bags". If a bag contains one or more positive instances, its label is positive, and negative otherwise. This learning paradigm has many useful applications. For example, Dietterich et al. [9] apply MIL to the problem of drug discovery, where each drug molecule can have several different 3D shapes. The properties of the molecule can be tested, but it is not known which particular shape is responsible for certain behaviors. Hence, in this case a bag is a set of feature vectors describing all possible shapes of a molecule. In [29], Viola et al. apply MIL to the problem of face detection. They argue the exact bounding box of a face is ambiguous, and instead gather several possible bounding boxes around each face to create a positive bag.

Formally, we are given a training data set $\{X_i, y_i\}_1^N$ where $X_i = (x_{i1}, x_{i2}..)$ is the i^{th} bag, and $y_i \in \{0, 1\}$ is the bag label. This problem in particular can benefit from an online approach because MIL datasets tends to be very large as each bag consists of many (potentially hundreds) of high dimensional vectors. We will use the loss function proposed by Viola et al. in [29][2]. First let us define the probability of a bag being positive as $p_i(H) = 1 - \prod_j \Big(1 - \sigma(H(x_{ij}))\Big)$, where again $\sigma(\cdot)$ is the sigmoid function. The above equation is based on the Noisy-OR formula. The loss function is the negative log likelihood for all bags:

$$\mathcal{L}(H|\{x_i, y_i\}_1^N) = -\sum_{i=1}^{N} y_i \log p_i(H) + (1 - y_i)\log(1 - p_i(H))$$

In the online case we get one bag at a time. Note that although the loss function cannot be split into independent terms per each instance, it can be split into independent terms per each bag. We will use $p_i = p_i(H_{m-1} + h(\cdot; a_m))$ and $p_{ij} = \sigma\Big(H_{m-1}(x_{ij}) + h(x_{ij}; a_m)\Big)$ to simplify notation. Taking the necessary derivative:

$$\frac{\partial \mathcal{L}_i(H_{m-1} + h(\cdot; a))}{\partial a} = \frac{p_i - y_i}{p_i} \sum_j p_{ij} \frac{\partial h(x_{ij}; a)}{\partial a}$$

This leads to the following update rule:

UPDATE RULE

$$a_m \leftarrow a_m - \eta_i \frac{p_i - y_i}{p_i} \sum_j p_{ij} \frac{\partial h(x_{ij};a)}{\partial a} \mid_{a_m}$$

Note that the resulting algorithm is different than the one presented in [3] because here each weak learner is trained with weighted samples.

4.4. Weak Learners

As we mentioned before, we would like to choose weak learners that can return real values (in other words, regression models) and are differentiable with respect to their parameters. In this paper we use the following weak models $h(x; a) = \beta_0 + \beta_1 \tanh(x^\top \bar{\beta}_2)$, where $a = (\beta_0, \beta_1, \bar{\beta}_2)$. We experimented with other similar models replacing the hyperbolic tangent with the sigmoid function, arc tangent, etc. The various alternatives gave similar performance, and we chose the hyperbolic tangent because it has a simple derivative.

5. Experiments

In this section we show results for the three algorithms we derived in previous sections. Our primary objective is to show that the online and batch versions of stochastic boosting converge to similar results given enough training data. However, since both of these are novel, we also include results of a standard batch boosting algorithm. Finally, to compare absolute performance we include results for two other online algorithms: stochastic gradient descent used to train the model in Section 4.4[3], and a more classic linear model trained with stochastic gradient descent (for classification this is the perceptron [25], for regression this is least means squares (LMS) [30], and for MIL this is Logistic Regression MIL [24]). For all experiments we split the data

[2] We use the Noisy-OR version as it was reported with better performance than the other one in [29].

[3] Both OSB and BSB reduce to this if $M = 1$.

Name	# of examples/bags	# of features
MNIST17	4374	784
MNIST49	4979	784
Forest CoverType	56264	54
Abalone	4177	7
Head Pose	5007	256
Kinematics	8192	8
CBIR Fox	1320/200	230
CBIR Elephant	1391/200	230
CBIR Tiger	1220/200	230
MNIST MIL2	20000/10000	784
MNIST MIL4	20000/5000	784
MNIST MIL6	19998/3333	784

Table 1. **Data Sets**

randomly into 70/30% training and testing sets, and show averaged results of 10 trials. For all boosting algorithms we set the total number of weak classifiers, M, to 100. In all cases, *the online algorithms are trained with one example at a time*.

5.1. Binary Classification

In this section we experiment with the algorithm derived in Section 4.1 with three different data sets. In all cases we report the equal error rate (EER), which is obtained by choosing the appropriate confidence threshold. For the AdaBoost algorithm we use stump weak classifiers [4].

MNIST optical digit recognition
For the first experiment, we took the commonly used MNIST optical digit recognition benchmark (available at http://yann.lecun.com/exdb/mnist/) and extracted a binary data set: 0-4 vs 5-9. We used the raw pixel values as the features. The results are shown in Figure 3(A).

Smile detection
For this experiment we used the dataset presented in [19], which contains videos of 97 subjects displaying various emotions. The last 10 frames from each video (where the subjects make their most expressive motion) were extracted, and downsampled to 20×20 pixels with histogram equalization (*cf*. Figure 1). Each sample is represented with pixel values as features. We labeled the images as 'smiling' versus 'non-smiling', 783 and 6,768 images, respectively. Results are shown in Figure 3(B).

Figure 1. **Smile detection**: An example of the face images used in the smile detection experiment in Section (5.1)

Gender recognition
For our last classification experiment we used the data set from [17], which contains 13,233 images of celebrities. We labeled these as 'female' and 'male' and performed gender recognition. As before, the images were downsampled to 20×20 pixels with histogram equalization, and represented with pixel values (*cf*. Figure 2). Note that this dataset is very challenging as the face images are not well aligned (as a result of fully automatic face detection), and display large changes in illumination, expression, pose as well as occlusions.

Figure 2. **Gender recognition**: An example of the face images used in the gender recognition experiment in Section (5.1)

[4] One could argue that this is an unfair comparison because our weak classifiers have access to all features, while stump classifiers have access to only one. However, in experiments that are not included in this paper we found that AdaBoost with the perceptron as the weak classifier achieves almost identical performance as it does with stumps. Using the same weak model as we use for OSB and BSB gave poor performance with AdaBoost.

5.2. Regression

We now experiment with the algorithm derived in Section 4.2 with three different data sets. In this case the batch boosting algorithm we compare to is Least Squares Regression Boosting algorithm from [12], with a regression stump as a weak classifier.

Abalone age prediction
Our first experiment is with the commonly used Abalone data set from the UCI repository (available at http://mlearn.ics.uci.edu/MLRepository.html). This data set contains various physical measurements and the target variable is the age of the abalone. Results are shown in Figure 5(A).

Kinematic control of an 8-link robotic arm
This dataset is a simulation of an 8 link all revolute robotic arm. The input space contains the joint positions and angles, and the target variable is the distance from the end-effector to a target. There are several version of this dataset available in the DELVE repository (http://www.cs.toronto.edu/∼delve/), and we used the non-linear version, with medium amount of noise. Results are shown in Figure 5(B).

Figure 4. **Head Pose Estimation**: An example of the headpose images used in the experiment in Section (5.2)

Head pose estimation
We applied the algorithms to the task of head pose estimation. For this experiment we collected data using a camera and a 3D Inertia Measurement Unit (IMU) for six subjects. Our goal is to predict the yaw angle of the person's head (we discard pitch and roll information since yaw is the most important and most difficult to predict). We used a standard face tracker [20] to crop out square images containing the face, and aligned these with the IMU measurements. This

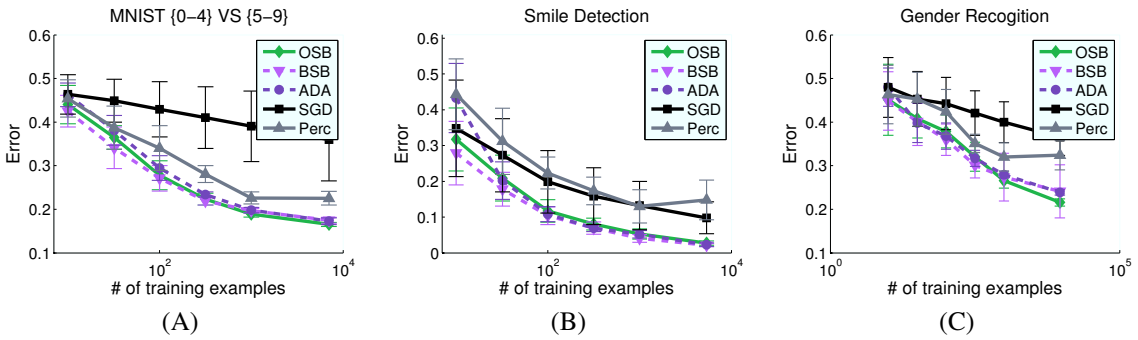

Figure 3. **Binary Classification**: **(A-B)** MNIST 47 VS 19 and MNIST 0-4 VS 5-9 digit recognition. We plot the equal error rate against the size of the training data set. **(C)** Same as the previous figure, but with the Viola-Jones face dataset. Batch algorithms represented with dashed lines, online algorithms with solid lines. See text for details and supplementary material for larger images.

procedure resulted in 5,007 face images, with corresponding ground truth yaw measurements. All face images were downsampled to of 16×16 pixels with histogram equalization, and the pixel values were used as features (*cf*. Figure 4). We plan to make this data publicly available in the near future. Results are shown in Figure 5(C).

5.3. Multiple Instance Learning

Finally, we experiment with the algorithm presented in Section 4.3. Recall that for online multiple instance learning, we receive one bag at a time, rather than one instance. Furthermore, in all experiments we present the equal error rate on bags.

Content Based Image Retrieval

The task of identifying the main object in a given image can be framed as a multiple instance learning problem. The image is segmented into several coherent regions, and we can assume that one of these regions is the object of interest, but it is not known which segment "explains" the bag label. For this experiment we used a standard CBIR data set that has been mentioned in other MIL work such as [1]. This data set was generated using the blobworld system [6]. Each image is represented as a bag of feature vectors, where each vector is a high dimensional representation of a segment in that image. The images come from the Corel catalog, and 3 categories are used in the experiment: fox, elephant and tiger. The results are shown in Figure 6(A-C). We found that this data set is rather small, and thus observe a high variability in error, as well as poor performance with the fox category (although our results are similar to previously reported performance). This motivates our next MIL experiment.

MNIST MIL data

Since most of the commonly used MIL data sets are fairly small, we created several MIL datasets out of the MNIST data. We arbitrarily chose the digit '4' to be positive, and all other digits negative. We then generated positive and negative bags of different sizes (positive bags containing one '4' and negative bags containing only the other digits). The results are shown in Figure 6(D-F). We see that with a larger amount of data, the results are more stable and converge to a similar error rate.

5.4. Discussion

In all of the experiments we observe that with enough training data, the online stochastic boosting algorithms converge to similar performance as the batch stochastic boosting algorithms, as well as the standard batch boosting algorithms; they also outperform both stochastic gradient descent and the traditional linear models. Furthermore, the rate of convergence is in most cases similar for all algorithms.

The exact performance depends on the data set, and in some cases better absolute performance could be achieved with elaborate parameter selection and better feature representations.

Although we kept almost all parameters fixed for all experiments (changing only the learning rate η), our algorithms performed well on all of the diverse data sets. We again note that our algorithms are able to learn with one example at a time, and are straightforward to implement.

6. Related Work

In this section we discuss and compare our algorithms with related work in the literature. Friedman's stochastic gradient boosting algorithm [13] certainly has a similar title to ours, but is substantially different. As we reviewed in Section 2, Friedman's approach is to optimize the overall loss function $\mathcal{L}$ with gradient descent in *function space*. In [13] he extends this idea to use stochastic gradient descent instead. Each step in this descent results in one weak classifier. Hence, if this algorithm were used in an online

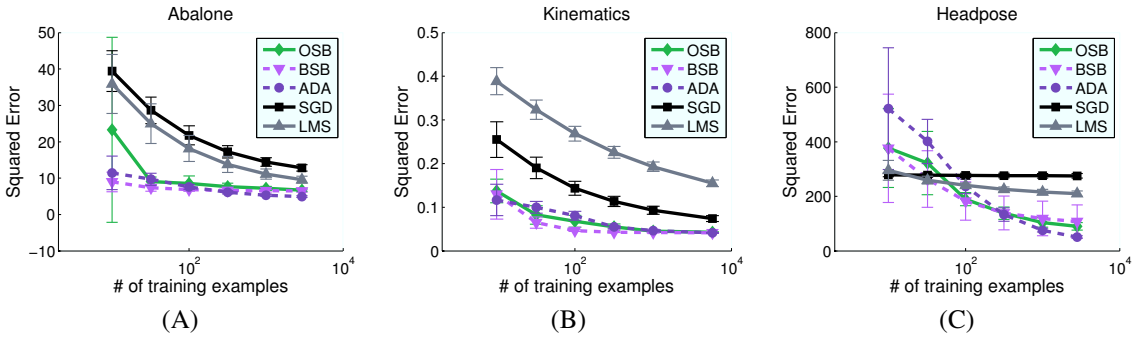

Figure 5. **Regression**: Average squared error versus the number of training examples for three data sets: **(A)** Abalone; **(B)** Kinematics; **(C)** Head pose estimation. Batch algorithms represented with dashed lines, online algorithms with solid lines. See text for details and supplementary material for larger images.

setting, it would result in a impractically large strong classifier (there would be 1 weak learner for every example seen). Friedman chose to experiment with the stochastic variation because of its resistance to noise, rather than to make the algorithm learn online. On the other hand, our approach is to use stochastic gradient descent in *weak parameter space* to optimize $\mathcal{L}(H_{m-1} + h(\cdot; a))$ for each weak learner. This allows us to develop boosting algorithm that learn from one example at a time.

A more recent work by Bradley and Schapire [5] proposes a boosting algorithm with a filter flavor, where each weak classifier is trained with new data. This approach is well suited for large datasets, but it cannot learn from one example at a time, as is necessary in many real-time applications.

The statistics community has also applied the ideas of stochastic and convex optimization to boosting (e.g., [32, 18]). Here, however, the set of weak learners is assumed to be finite and relatively small. These methods then use convex optimization to find the optimal set of weak learner weights α_i. In our framework there is a continuous space of weak learners, and these types of approaches would not work.

Finally, since our approach requires the weak classifiers to be differentiable, there is an interesting similarity to neural networks (where each weak classifier is a node). In particular, Ash [2] proposed an algorithm that adds one node to the network at a time, which bears some similarity to boosting in general. We note, however, that although back-propagation [26] is based on stochastic gradient descent, Ash's method requires an iteration of back-propagation for each new node, which means that each data example is accessed multiple times.

7. Conclusions and Future Work

We have presented a framework that can be used to derive online boosting algorithms for various interesting loss functions, and derived three such algorithms. Our algorithms are able to learn with just one example at a time, and unlike other work [2, 5, 13], access each data point only once. Our framework can be used to easily derive boosting algorithms for many other problems (*i.e*. ranking [10], transfer learning [8], *etc*.).

We performed a wide range of experiments, and have presented empirical evidence that our online algorithms converge to give similar performance as standard batch boosting algorithms. One requirement of our approach is that we assume that the weak learners must be differentiable with respect to their parameters, but we believe this still leaves a wide range of choices for the designers of an algorithm.

There are two possible avenues for future work. First, it would be interesting to prove convergence bounds for some of the algorithms we have developed, although such bounds are likely to exist only under certain necessary conditions. Second, we would like to take advantage of the flexibility of our framework, and apply it to other challenging real world problems, such as visual tracking.

Acknowledgments

This research was supported in part by NSF CAREER Grant #0448615, NSF IGERT Grant DGE- 0333451, and ONR MURI Grant #N00014-08-1-0638. Part of this work was done while B.B. and M.H.Y. were at Honda Research Institute, USA.

References

[1] S. Andrews, I. Tsochantaridis, and T. Hofmann. Support vector machines for multiple-instance learning. In *NIPS*, pages 561–568. 2002.

[2] T. Ash. Dynamic Node Creation in Backpropagation Networks. *Connection Science*, 1(4):365–375, 1989.

[3] B. Babenko, M.-H. Yang, and S. Belongie. Visual Tracking with Online Multiple Instance Learning. In *CVPR*, 2009.

[4] L. Bottou. On-line learning and stochastic approximations. In D. Saad, editor, *On-Line Learning in Neural Networks*, pages 9–42. Cambridge University Press, 1999.

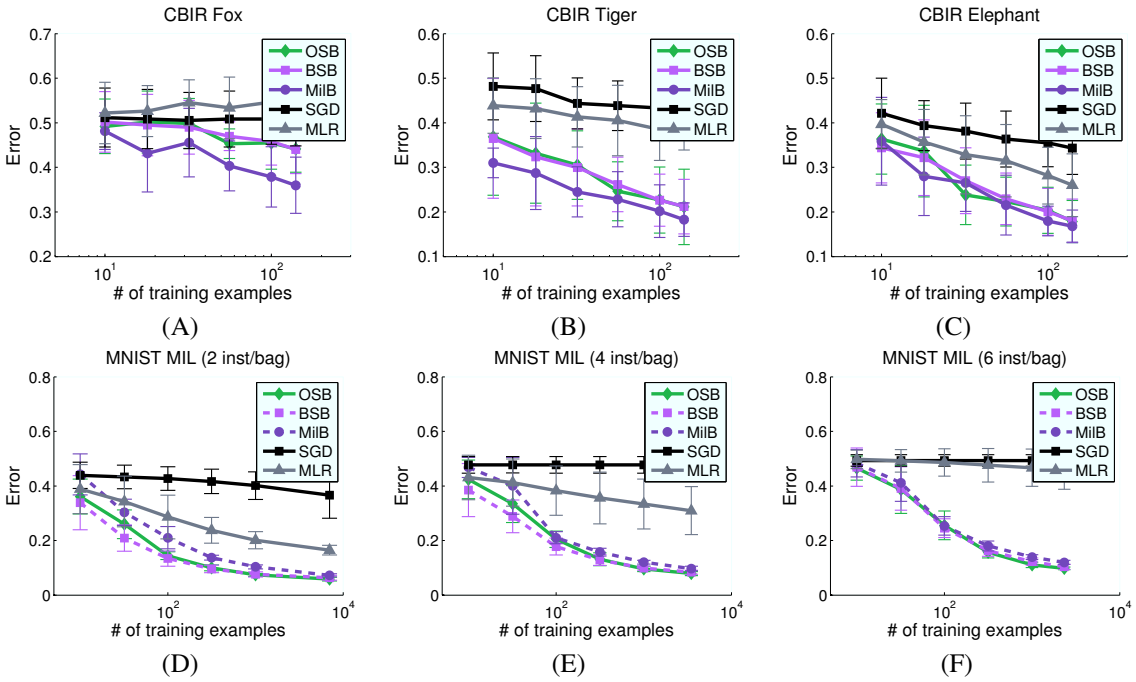

Figure 6. **Multiple Instance Learning**: **(A-C)** Content based image retrieval using blobworld image features. We plot the equal error rate against the size of the training set (note that both the error rate and data size are in terms of bags, rather than instances). **(D-F)** A similar experiment but with MNIST data in a mil format. Batch algorithms represented with dashed lines, online algorithms with solid lines. See text on how these data sets were created and supplementary material for larger images.

[5] J. Bradley and R. Schapire. FilterBoost: Regression and Classification on Large Datasets. In *NIPS*, 2007.

[6] C. Carson, M. Thomas, S. Belongie, J. Hellerstein, and J. Malik. Blobworld: A system for region-based image indexing and retrieval. *Third International Conference on Visual Information Systems*, pages 509–516, 1999.

[7] G. Cauwenberghs and T. Poggio. Incremental and Decremental Support Vector Machine Learning. In *NIPS*, pages 409–415. 2001.

[8] W. Dai, Q. Yang, G. Xue, and Y. Yu. Boosting for transfer learning. In *ICML*, pages 193–200. ACM New York, NY, USA, 2007.

[9] T. G. Dietterich, R. H. Lathrop, and T. Lozano-Perez. Solving the multiple-instance problem with axis parallel rectangles. *Artificial Intelligence*, 89(1-2):31–71, 1997.

[10] Y. Freund, R. Iyer, R. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. *The Journal of Machine Learning Research*, 4:933–969, 2003.

[11] Y. Freund and R. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[12] J. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29(5):1189–1232, 2001.

[13] J. Friedman. Stochastic gradient boosting. *Computational Statistics and Data Analysis*, 38(4):367–378, 2002.

[14] H. Grabner and H. Bischof. On-line boosting and vision. In *CVPR*, pages 260–267. 2006.

[15] H. Grabner, C. Leistner, and H. Bischof. Semi-Supervised On-line Boosting for Robust Tracking? In *ECCV*, pages 234–247, 2008.

[16] Y. Grandvalet and C. Ambroise. Semi-Supervised MarginBoost. In *NIPS*, pages 553–560. 2002.

[17] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller. Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments. *University of Massachusetts, Amherst, Technical Report 07-49*, 2007.

[18] A. Juditsky, P. Rigollet, and A. Tsybakov. Learning by mirror averaging. To appear in *Annals of Statistics*, 2008.

[19] T. Kanade, J. Cohn, and Y. Tian. Comprehensive database for facial expression analysis. *Proc. of the Fourth IEEE Int'l Conf. on Automatic Face and Gesture Recognition*, pages 46–53, 2000.

[20] J. Lim, D. Ross, R. Lin, and M.-H. Yang. Incremental learning for visual tracking. *NIPS*, 17:793–800, 2005.

[21] X. Liu and T. Yu. Gradient feature selection for online boosting. In *ICCV*. 2007.

[22] L. Mason, J. Baxter, P. Bartlett, and M. Frean. Boosting algorithms as gradient descent in function space. In *NIPS*, volume 12, pages 512–518, 2000.

[23] N. C. Oza. Online Ensemble Learning. *Ph.D. Thesis, University of California, Berkeley*, 2001.

[24] S. Ray and M. Craven. Supervised versus multiple instance learning: an empirical comparison. *ICML*, pages 697–704, 2005.

[25] F. Rosenblatt. The perceptron. *Psychical Review*, 65(6):386–408, 1958.

[26] D. Rumelhart, G. Hinton, and R. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[27] R. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.

[28] P. Utgoff, N. Berkman, and J. Clouse. Decision Tree Induction Based on Efficient Tree Restructuring. *Mach. Learn.*, 29(1):5–44, 1997.

[29] P. Viola, J. Platt, and C. Zhang. Multiple instance boosting for object detection. In *NIPS*, pages 1417–1424. 2006.

[30] B. Widrow and M. E. Hoff. Adaptive switching circuits. In *Neurocomputing: Found. of research*, pages 123–134. MIT Press, 1988.

[31] R. Zemel and T. Pitassi. A gradient-based boosting algorithm for regression problems. In *NIPS*, pages 696–702. 2001.

[32] T. Zhang. Sequential greedy approximation for certain convex optimization problems. *IEEE Trans. on Inf. Theory*, 49(3):682–691, 2003.