GreedyBoost: Boosting Algorithms in the Online Setting

Deshana Desai and Tushar Anchan

December 19, 2017

1 Introduction

Boosting is an ensemble method that uses multiple base learners that do slightly better than random guessing and combine their hypotheses to create a strong learner. We have seen that batch versions of boosting (specifically AdaBoost) are very effective and are generally used in practice too. We have also seen that the empirical error for Adaboost decreases exponentially as a function of the number of rounds of boosting.[14] With the ever increasing role of data in the modern world, batch algorithms can be inefficient if datasets do not fit in memory. Hence, online learning algorithms can be more practical as compared to batch algorithms as they are very efficient in terms of space.

An online learning algorithm processes one data point at a time and does not require any distributional assumption. Hence, it is evident as to why it can be more practical compared to batch algorithms in certain scenarios. We try to study the existing methods that make use of boosting in an online setting.

We study online versions of boosting proposed in the works by *Oza and Russell*[4], *Liestner et al.*[3], *Chen et al.*[2]. A drawback for the online algorithm is that the discriminative complexity of the classifier is limited, because the number of weak classifiers is fixed.[19] Moreover, if we combine too many weak learners, the result can be dominated by poorly performing weak learners that shouldn't have been included.[2] We propose to modify the Online Boosting algorithm suggested by *Oza and Russell*[4] to use the greedy strategy of starting with one weak learner and adding a new weak learner after each misclassification. We perform various experiments to compare the accuracy of such a model and comment on different variants of the greedy strategy used. While our algorithm doesn't have any theoretical guarantees, we try to show a weak justification for why it can be useful. We also address the question of how and when to add new base learners.

1.1 Online Learning

Online Learning algorithms process an example one at a time compared to processing the whole set of examples in the batch setting. At each time step t, the algorithm receives an example x_t and makes a prediction $\hat{y}_t$. It then receives the true label y_t and incurs a loss $L(\hat{y}_t, y_t)$, where L is a loss function. For classification problems the loss function is typically (0-1) and $L(\hat{y}_t, y_t) = |\hat{y}_t - y_t|$

for regression problems. The objective of an online algorithm is to minimize the loss $\sum_{t=1}^{T} L(\hat{y}_t, y_t)$ over T rounds[14].

Note that in the online setting, no distributional assumption is required and so in some sense there is no notion of generalization. Infact, theoretical guarantees in this setting assume worst case or adversarial scenarios. The performance of an online learning algorithm is analyzed with the help of the notion of regret or mistake models.

1.2 Batch Boosting vs Online Boosting

We start by revisiting the boosting framework for batch algorithms and see the challenges that arises for coming up with boosting algorithms in the online setting. The general online setting runs for T rounds where at the tth round the algorithm receives an instance a data point $x_t \in X$ and makes a prediction y_t. It then receives the true label and then incurs a loss. In batch boosting, the whole set of labelled examples are available to the base learner and it updates it hypothesis based on the loss it incurs on the whole training.

Now in the online setting, since the examples are available one at a time, the algorithm must run the N base learners in parallel *i.e.* the algorithm must update the N base learners right away before seeing the remaining examples. Hence, the weighting scheme for the examples in the batch boosting wouldn't work in the online setting as the weights are determined only after seeing all the examples and so, a weighting scheme is required that is suited for the online setting while still satisfying the property of increasing the weight to a misclassification and decreasing in the other case [2]. In addition, it is also not clear how to choose an appropriate number N of base learners for the online algorithm. There are theoretical results that give an upper bound for the value of N and as pointed out by *Chen et al.*[2] this maybe much larger than the appropriate, moreover they explain that if we combine too many base learners, the result may be dominated by the poor predictions of the many base learners which should not be included.

1.3 Setup

We say that a function F is online learnable if there exists online learning algorithm which given any sequence of examples (x_t, y_t) for t = 1,2,...,T generates predictions $\hat{y}_t$ such that:

$$\sum_{t=1}^{T} 1\hat{y}_t \neq y_t \leq \inf_{h \in H} \sum_{t=1}^{T} 1h(x_t) \neq y_t + R(T)[15] \tag{1}$$

where R is the Regret. The above definition is simply the online generalization of the ERM (Empirical Risk Minimization) in the batch setting. We desire an online boosting algorithm such that $\frac{1}{T}R(T) \to 0$ quickly as $N \to \infty$ and $T \to \infty$.

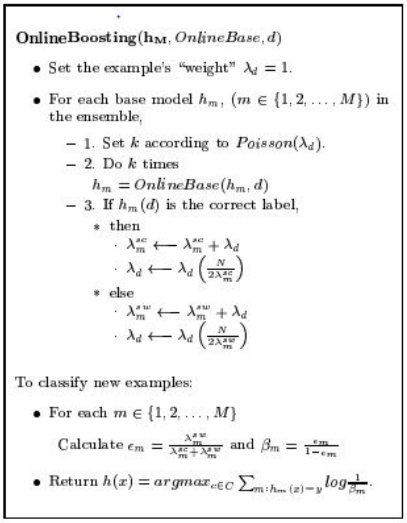

Figure 1: Online Adaboost [4]

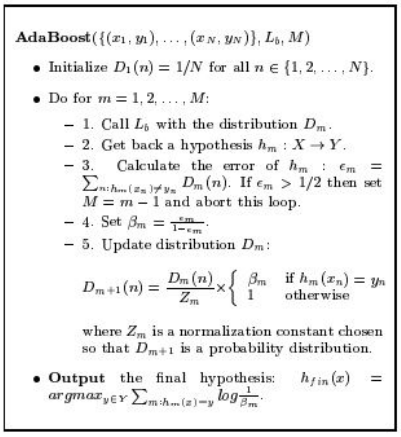

Figure 2: Adaboost Algorithm[4]

2 Online Boosting Algorithms

2.1 OzaBoost

Boosting in the online setting is believed to first appear in the work by *Oza and Russell*[12]. They develop an online version of the well-known batch boosting AdaBoost algorithm. Since then online boosting has given successful results in object detection and visual tracking problems.

They use a poisson sampling process to approximate the weighting scheme of AdaBoost and show that their algorithm will converge to the performance of AdaBoost as the number of examples approaches infinity[4]. Just as in AdaBoost[14], their weighting scheme updates the weights of a misclassified example by giving it half the total weight(poisson value) in the next stage and the other half to the correctly classified example. More specifically, OzaBoost assigns a weight λ_d to example (x) equal to $\lambda_d N/(2\lambda_m^{sw})$ if it is incorrectly classified or $\lambda_d N/(2\lambda_m^{sc})$ if it is correctly classified for the next base learner where the λ_m^{sw} and the λ_m^{sc} are the number of incorrect and correct classifications by the mth base learner. Two take away points from their work was that the base learners to be used should be online themselves *i.e.* an incremental learning algorithm and that the number of base learners must be specified before training. We will denote the algorithm as OzaBoost.

2.2 Online SmoothBoost

The weak learning assumption of this boosting algorithm is as follows: *Assumption 1: There exists an online base learner which can achieve advantage* $2\gamma > 0$ *for any sequence of examples and weights satisfying the condition* $|w| \geq c/\gamma^2$

and $w_t \in [0,1] \forall t$ and some constant c. The advantage in this case is defined as:

$$\sum_{t=1}^{T} \frac{w_t h(x(t)) y_t}{|w|} \tag{2}$$

Suppose Assumption 1 holds and let WL be such an online base learner with advantage 2γ. There are N copes of base learners as our N base learners. The weights are chosen similar to SmoothBoost *Servedio, 2003*[5] as follows:

$$w_t^{i+1} = min(1-\gamma)^{z_t^{(i)}/2}, 1 \tag{3}$$

where $z_t^{(i)} = z_t^{(i-1)} + w_t h_t^{(i)}(x(t)) y_t - \theta$ with $\theta = \gamma/(2+\gamma)$.

We also refer to the algorithm which gives uniform weight to learners with the weight of each example i updated by each base learner as: $w_i = 1/(1+\exp^{y_i * F_i})$ where $F_i = \text{Sign}(\sum_{j=1}^{i} h_j(x_i))$ as "Smooth Boost".

2.3 OCPBoost and EXPBoost

OCPBoost propose a more reasonable weak learning assumption and adapt the offline SmoothBoost algorithm. Their assumption requires that the base learner perform well only with respect to a certain distribution. Their weighting scheme is the same as the one in SmoothBoost.They also provide theoretical guarantees for the choosing an appropriate number of base learners and choose it based on those results. To mitigate the problem of redundant weak learners dominating the prediction, they give different voting weights to different weak learners and determine those weights using Online Convex Programming[2] and Prediction with Expert Advice[2]. We call the one that used online convex programming as the OCPBoost algorithm and the one that uses prediction with expert advice as the EXPBoost algorithm.

2.4 OGBoost: Online Gradient Boost

OGBoost[3] is an online version of GradientBoost which uses functional gradient decent to determine the optimal example weights and greedily minimizes the loss function of interest. The algorithm tries to reduce the effect of the presence of noisy samples. They keep a fixed set of base learners and perform boosting on the "selectors" among these base learners. The m^{th} selector maintains a set of K base learners, thus maintaining a total of MxK base learners. In the first round, the first example is passed to the first Selector which chooses its best performing base learner among its set of K base learners as the representative for that selector. This continues for each of the M selectors resulting in a set of M representatives which are then ensembled. The optimization step is performed by updating the weight estimate w_n according to the negative derivative of the loss function $-l'(y_n F_m(x_n)$ where $F_m(x_n)$ refers to the ensemble of the chosen base learners for that round. The next $(n+1)^{th}$ selector with K base learners uses this w_n to train on that point (i.e. as sample weight) and to compute their error ($e_m^k = e_m^k + w_n * Sign(f_m^k(x_n)) \neq y_n$). The best base learner among these K is chosen and it's hypothesis is added to the existing set.

2.5 Constructing the Final Ensemble

In this study, we use datasets where the sequence of examples are assumed to be drawn i.i.d from a joint distribution. Let S = $((x_1, y_1), ..., (x_m, y_m))$ be the training set consisting of m examples drawn i.i.d from D. The goal for a supervised learner is to learn a model that minimizes the generalization error $E_{(x,y)\ D}[l(y, h_S(x))]$. If the training examples in S are input sequentially to a given online learning algorithm, the algorithm constructs a sequence of models $h_t : X \to \hat{Y}, t = 1, 2, ..., T + 1$ and returns predictions based on them. The performance of an online learner is measured by it's cumulative loss $L_T = \sum_{t=1}^{T} l(y_t, \hat{y}_t)$, relative to the cumulative loss of the best model in a comparator class H.

To test our model on a test set as in the case of batch setting, there are several possible ways to construct a final model $h_S : X \Rightarrow \hat{Y}$ based on the models h_t for $t \in [1, T]$ produced at different rounds t by the online algorithm as given in [11].

1. We can run the online algorithm for a single pass or for multiple passes over the training set and return the final model H_{T+1}. However, this may not always be the best approach.

2. We can also return the model among $h_1, ..., h_T$ that survived the longest (the 'pocket' method [Gallant, 1986])[16].

3. We can return the best model from $h_1, ..., h_T$ based on its empirical performance on a validation or test set [Littlestone, 1989][17].

4. we can randomly choose one model from the set $h_1, ..., h_T$.

5. we can average over the models in some way and return the resultant model [Helmhotz & Warmuth, 1995][18].

6. We can run the algorithm for a single pass over a subset of the data seen until that round and hold-out the rest of the data as a test set for each round. However, this may not be a good measure as the size of the test set keeps decreasing.

We choose the hypotheses constructed by final model H_T as is done in the literature [4], [3], [2].

3 Our Modification

3.1 Motivation

A drawback of online boosting algorithms is that the discriminative complexity of the learner is limited, because the number of base learners is fixed.[19] We have also seen that it is not clear how to choose an appropriate value of base learners N for boosting in the online setting even though upper bounds have been proposed for the choice of N [2] [15]. Moreover, *Chen et al.*[2] comments that choosing too many base learners could lead to predictions dominated by redundant base learners that did not learn well whereas including too few base learners could make the ensembling process itself redundant, as the goal of

boosting is to improve upon the performance of the base learners. However, in the batch setting setting, as we keeping adding a new hypothesis in each round and stop once the new weak hypothesis reaches a threshold error, this is not an issue. In addition, for the case of OzaBoost, *Charles Mersh*[12] empirically show that as the number of base learners increases, the accuracy of the OzaBoost algorithm starts decreasing. Hence, it seems appropriate to address the question of how to choose a good value for N.

Adding a new base learner to the existing set seems more intuitive in the online case where no assumptions can be made about the size or complexity of the dataset. In a sense, the modified algorithm adjusts its set of base learners in an online manner (i.e. by processing input in a serialized fashion without having any knowledge of the dataset available from the start).

We propose a greedy strategy to tackle this, by starting with a small number of base learners and adding a new base learner only when it is appropriate to. We also try to address the issue of deciding when would be an appropriate instance to add a new base learner and how should it be added to ensure that the resulting combined hypothesis performs comparable to the OzaBoost algorithm.

3.2 Algorithm

We modify the OzaBoost algorithm to greedily add a new hypothesis h to add to the existing set instead of choosing a fixed number N as the number of base learners. We start with one base learner in this algorithm and greedily keep adding new learners whenever an example is misclassified with a $margin = y_i \sum_{i=1}^{N} h_i(x)\alpha_i$ less than δ where the $\alpha_i's$ are normalized. Let $m(x_t)$ refer to the margin of an example (x_t). We use the same weighting scheme as in the OzaBoost algorithm. We define a confidence in the prediction of an example as $c(x_t) = m(x_t)/\sum_i \alpha_i h_i(x_t)$ as proposed in [13]. It is easy to see that an example is misclassified when $m(x, y) < 0$ and correctly classified when $m(x_t) > 0$. We add a new base learner to the set after every miss-classification to increase the capacity of prediction for the existing boosted set of base learners.

Consider an example for which the final weighted prediction of the base learners be incorrect. This implies that $\text{Sign}(y_i \sum_{i=1}^{N} h_i(x_t)\alpha_i) < 0$ where $\alpha_i's$ are normalized. Let $y_i \sum_{i=1}^{N} h_i(x_t)\alpha_i$ be greater than $-\delta$ (for example if $\delta = 0.5$ then the $y_i \sum_{i=1}^{N} h_i(x)\alpha_i = -0.4$ is greater than -0.5). Then we can say that with not too large a margin, the example was incorrectly classified.

We can observe that as an example is consistently misclassified by all the base learners in the set, the weight of the example increases and subsequently, the margin $m(x_t)$ becomes a large negative number. As this happens, it becomes less and less likely that the example will be correctly classified by the base learner.

Now, we posit that it is more optimal for the algorithm to "give up" on those examples and concentrate its efforts more on those examples whose margin is a small negative number as suggested in *Freund*[13]. The parameter δ refers to the value of the margin which we consider large enough to "give up" on the example. If the margin of an example $m(x_t) < \delta$ and $m(x_t) < 0$ (since the example is misclassified), we deem the example to be close to the distribution that the set of base learners have learned and try to correctly classify this point by adding a new learner to the set which is trained multiple times on this example.

Algorithm 1 Greedy Ozaboost Update

procedure InitializeNewLearner
    Input: Streaming examples $(x1, y1), ...(xM, yM)$
    Input: Online base learner L
    Initialize: $W_m^1 = 1$ for m in $[M]$
    Initialize: $N = 1$, error $= 0$ and $\alpha_1^1 = 1$
    Initialize new base learner h_t^i
    for each point (x, y) in *previouspoints* **do**
        Fit h_t^i on (x, y)
        Calculate $\lambda_m^{sw} = \lambda_m^{sw} + 1_{h_t^i(x) \neq y}$
        Calculate $\lambda_m^{sc} = \lambda_m^{sc} + 1_{h_t^i(x) = y}$
    for i in range($Poisson(Weight)$) **do**
        Fit h_t^i on misclassified example (x, y)
        Calculate $\lambda_m^{sw} = \lambda_m^{sw} + Poisson(Weight) * 1_{h_t^i(x) \neq y}$
        Calculate $\lambda_m^{sc} = \lambda_m^{sc} + Poisson(Weight) * 1_{h_t^i(x) = y}$
    Calculate Error $e_t = \lambda_m^{sw} / (\lambda_m^{sw} + \lambda_m^{sc})$
    Add new h_t^i to existing set of base learners
procedure Main
    for Each point (x, y) in *previouspoints* **do**
        Calculate $margin = y_i \sum_{i=1}^{N} h_i(x)\alpha_i$ of each point (x, y)
        if $margin < \delta$ **then**
            Call InitializeNewLearner(example $= (x, y), weight$)

Now we address the task of how to add a new learner to the existing set. We suggest the following ways:

1. Add a base learner WL_i trained once on all the previous examples encountered by the set of learners. This has the pitfall of having to store the examples encountered till the current misclassification (X_t, y_t). An interesting variant of this solution would be to only store γ past examples where $\gamma << T$ (N being the size of the dataset). We have explored the results of this approach in the Experiments section.

2. Add a base learner WL_i trained on a past examples (X_t, y_t) with a probability $p_t = \frac{\sum_{i=1}^{N} \alpha_i}{\sum_{i=1}^{N} h_i(x)\alpha_i}$. Thus if the margin of classification of an example is small, the probability of adding the example to the training set of the base learner WL_i would be high.

3. Add a base learner WL_i a discount factor γ for examples that were encountered further back in time. Thus $h_i = \gamma x(t) + \gamma^2 x(t-1) + \gamma/^3 x(t-2)$.. and so on.

4. The above methods can be said to be a priming of the base learner with either the entire previous subset of examples or a subset of the examples. A different approach could be to make a copy of the hypothesis learned by an existing base learner and sufficiently train it on the misclassified examples to cause a large enough perturbation of the learned hypothesis.

Note that we want to add a base classifier so that $y_i(\sum_{i=1}^{N} h_i(x_t)\alpha_i + h_{i+1}(x_t)\alpha_{i+1}) > 0$ for the misclassified example (x_t, y_t). We know that $y_i(y_i \sum_{i=1}^{N} h_i(x_t)\alpha_i > -\delta and y_i \sum_{i=1}^{N} h_i(x_t)\alpha_i < 0$

Thus we need $h_{i+1}(x)\alpha_{i+1} > \delta$. To find such a base classifier, we need $y_i h_{i+1} > 0$ where h_{i+1} is the predicted value by the new hypothesis. Let $\sum_{i=1}^{N} h_i(x)\alpha_i = -\delta + \epsilon$ where $\epsilon > 0$. Then we get $\alpha_{i+1} >= \delta - \epsilon$ to satisfy our condition that $y_i(\sum_{i=1}^{N} h_i(x)\alpha_i + h_{i+1}(x)\alpha_{i+1}) > 0$. However, the α_i are calculated by $\log(1 - error)/error$ and thus need not necessarily satisfy above condition. However, if they do, then this point p will be correctly classified.

3.3 Justifications

To see why this strategy can be useful, we try to argue that there is some chance for the algorithm to make fewer mistakes or miss-classifications on adding a new base learner whenever it makes a mistake than it would have made had we continued as before. For simplicity, we consider the problem of binary classification with a 0-1 loss and the margin δ to be 0. Let's say you have added several base learners appropriately and now have N of them for the tth round and the algorithm miss-classifies this example. A mistake in the tth round would mean

$$y_t.\text{sign}(\sum_{i=1}^{N} h_i(x_t)\alpha_i) < 0 \tag{4}$$

which is equivalent to $\text{sign}(\sum_{i=1}^{N} h_i(x_t)y_t\alpha_i) < 0$. This can further be represented as

$$\sum_{i:y_t h_i(x_t)=1} \alpha_i - \sum_{i:y_t h_i(x_t)=-1} \alpha_i < 0 \tag{5}$$

or that the total weights of the base learners that classified the example wrong is greater than the total weights of the ones that classified the example correctly. Now, consider the $t+1$th round where we add a new base learner and run the algorithm. Observe that the example is classified based on the sign of

$$\left(\sum_{i:y_{t+1} h_i(x_{t+1})=1} \alpha_i - \sum_{i:y_{t+1} h_i(x_{t+1})=-1} \alpha_i \right) + \alpha_{N+1} y_{t+1} h_{N+1}(x_{t+1}) \tag{6}$$

Since we make no assumption for the distribution of the data we can't figure out exactly how many base learners and which one of them classifies the example correctly. Note that the expression in the (.) would determine if an algorithm(lets call it A) that started with N base learners would classify the $t+1$th example correctly or not. Consider the following cases:

1. A made a mistake at that round. Depending upon how good the new base learner added is, it is possible for it to overpower the sum of the miss-classification weights if all the other base learners are performing somewhat similar to each other in terms of accuracy thereby, preventing the miss-classification. We claim that it will be the case with high probability if we randomize the sequence in which the example is passed on the base learners at each round and if the newly added base learner is somewhat better than the base learners that are slightly better than random guessing.

2. *A* didn't make a mistake at that round. Like earlier, it is again possible for the new base learner added to overpower the sum of the correctly classified weights and cause the the algorithm to make a mistake this time. Here we claim that we keep adding new base learners each time we encounter this scenario and eventually we either resolve it in some round or that we end up using the maximum number of base learners that is appropriate. The reason it can be resolved is due to a randomized sequence of training the base learners there is a high probability that the base learners perform similar to each other(empirically shown in [7]) in terms of accuracy and so the probability that more than half the base learners miss-classify an example keeps decreasing as the number of rounds increases. However, if it is still not resolved and we reach the max number of base learners we can have, note that the algorithm from here on will work similar to OzaBoost and hence should converge given large number of examples.

Hence, with this strategy there seems to be a good chance that the algorithm uses as few base learners as possible to reduce it's error as much as possible.

4 Experiments

In this section, we describe how we ran our experiments on several datasets and compare the empirical results of our algorithm with the mentioned boosting algorithms in the online setting.

4.1 Datasets

For the purpose of our study we used the *iris*, *mushroooms*, *breast cancer* and *ionosphere* datasets from the UCI ML Repository at `https://archive.ics.uci.edu/ml/datasets.html`. We convert the heart dataset to Binary Classification (with labels +1 and -1) to refer to the presence or absence of the heart disease. It is mentioned in the data sheet that experiments with the Cleveland database have concentrated on simply attempting to distinguish presence (values 1,2,3,4) from absence (value 0) so the dataset is not highly unbalanced. We kept aside 20% of the data for test and trained on the remaining 80% data subset. We ran each algorithm ten times with each number of training examples to account for the randomness in the ensemble algorithms while also shuffling the order of the datapoints. We report the average error made by the classifier on each of the test sets. The base learners used for the online boosting must themselves be online algorithms. We thus implemented and used Random Decision Stumps, Perceptrons and Naive Bayes as online base learners. Note that in each of the experiments we have trained the online classifiers only for 1 epoch (single pass).

4.2 Results

We tested our algorithm using 5 binary classification datasets as explained above. First, we compare the results of different online algorithms on each of these datasets as shown in the Tables 1.1, 1.2 and 1.3. The results in this table are computed over an average of 20 randomized runs on the training set.

We chose the number of base learners for every online algorithm (except GreedyBoost which starts with only one base learner) as 100 as done in Chen et al. [2] For Greedyboost, we can tune the parameter δ using progressive validation which is a standard online validation technique, where each training example is used for testing before it is used for updating the model (Blum et al., 1999)[20]. However, we have kept the δ constant at 0.8 for the purposes of our experiment.

Datasets	Ozaboost	OGBoost	OCPboost	OSboost	SmoothBoost	ExpBoost	GreedyBoost
Heart	0.5383	0.5016	0.5408	0.5441	0.5075	0.5675	**0.57**
Ionosphere	0.7571	0.7714	0.8242	0.8407	0.7121	0.82	**0.8714**
Breast Cancer	0.6859	0.7622	0.7662	0.7456	0.8013	0.6442	**0.8666**
Mushrooms	0.8477	0.8801	0.887	0.887	**0.887**	0.8818	0.8860

Table 1.1 -The above table displays results for *Perceptron* used as the base classifier.

Datasets	Ozaboost	OGBoost	OCPboost	OSboost	SmoothBoost	ExpBoost	GreedyBoost
Heart	0.7466	0.6566	0.7641	0.7991	0.6658	0.7733	**0.8166**
Breast Cancer	0.885	0.8868	0.7697	**0.9157**	0.889	0.8776	0.8947
Mushrooms	0.84	**0.8867**	0.8678	0.8248	0.8810	0.8227	0.8523
Iris	**0.7766**	0.4333	0.6888	0.7133	0.48	0.7733	0.76

Table 1.2 - The above table displays results for *Naive Bayes* used as the base classifier.

Datasets	Ozaboost	OGBoost	OCPboost	OSboost	SmoothBoost	ExpBoost	GreedyBoost
Heart	0.795	0.7316	0.7333	0.8033	0.735	0.7383	**0.8166**
Ionosphere	0.7685	0.76	0.5371	0.8514	0.7571	0.6171	**0.8485**
Breast Cancer	**0.9324**	0.9087	0.6543	0.921	0.9052	0.8192	0.9123
Mushrooms	0.9307	0.6888	0.7919	0.7571	0.6491	0.6859	**0.9353**
Iris	0.77	0.4333	0.8	0.8	0.8	0.796	**0.8**

Table 1.3 - The above table displays results for *Decision Stumps* used as the base classifier.

For the Decision Stumps and Perceptron the base learners, our modification to the Ozaboost algorithm gives an improved performance across all datasets. For the Naive Bayes algorithm, the performance remains competitive. The performance difference is more pronounced for the Perceptron base learner used with GreedyBoost compared to the other algorithms. While the difference is lesser for the other base learners coupled with Greedyboost, it can be clearly claimed that it performs atleast as well and in some cases better than the other online algorithms. the difference between the GreedyBoost and other online algorithms is more pronounced for small datasets since the number of base learners chosen by GreedyBoost are much lesser than the maximum number allotted as 100. For the larger datasets, GreedyBoost generates the maximum number of base learners and shows similar performance to the other online algorithms. It is encouraging to note that adding base learners dynamically does not deteriorate performance even when the total number of base learners are large.

While having a better performance, GreedyBoost also runs faster in compar-

ison to OzaBoost since the number of operations required to update N number of learners at each round is larger than in the case where each of these N learners are added at different rounds. A major pitfall of our method is that the algorithm needs to store all the previous points in memory. This could be catastrophic for extremely large datasets. Online learning is especially used for cases where the data is too large to hold in memory. We have discussed some suggestions above to overcome this disadvantage. We implement the first method suggested i.e. storing only a small percentage of the previously seen data points denoted by P in memory and training the new base learner on them. In figure 3. we vary the number of points stored and report the results. Specifically, we vary P from 10% data (50 examples), 20% of the data (100 examples) and 45% of the data (200 examples) for the maximum number of samples. We compare this with the case where we store all the previous data points and train the new learner on each of them. Note that P refers to the most recent previously encountered points. It is interesting to note that the performance remains roughly similar for all of the above cases suggesting that this may be a useful solution to overcome our problem.

To take a closer look at what happens under the hood, we track the number of cumulative mistakes made upto a certain example i as the ensemblers in Ozaboost and Greedyboost receive each example. As shown in Figure 2, adding a new learner at a round i after an example i is misclassified not only possibly improves the classification in that round, it also prevents points from being misclassified in the future i.e. makes consistently lesser mistakes than the OzaBoost ensemble.

Finally, we note from Figure 4 that starting with a large number of base learners in OzaBoost leads to a slower convergence to the best performance. Intuitively, if there are a large number of poorly performing base learners, it can take longer to train them and correctly distribute the weight in proportion to the strength of their hypothesis. Thus, it makes sense to start with a very small number of base learners and add learners in an "online" fashion. Of course, if we fail to add base learners, the boosting algorithm becomes redundant and cannot transform the ensemble of base learners into a strong learner.

In Figure 2, we compare the performance of the different online boosting algorithms using Decision Stumps as the base learners for the WI breast cancer dataset. We plot the performance of the algorithms as a function of the number of base learners. We note that the OSBoost algorithm is the best at adapting to large increases in the number of base learners. On the other hand, OzaBoost achieves maximum accuracy when M = 15 and then consistently deteriorates as the number of base learners are increased. This underlines the importance of the choice of the number of base learners used on the accuracy of the algorithm. Greedy OzaBoost helps selecting the number of base learners which perform well for the given dataset and increases computational speed by performing only a single pass over data without using more base learners than required (as opposed to OzaBoost where we must randomly guess N, generally chosen as 100). While OGBoost, EXPBoost, OSBoost are not significantly facing any reduction in the accuracy, they quickly achieve best performance with a small value of N and the accuracy stops improving after a point. By dynamically choosing the

value of N, one could achieve good performance without a trade-off in accuracy and decrease computational speed.

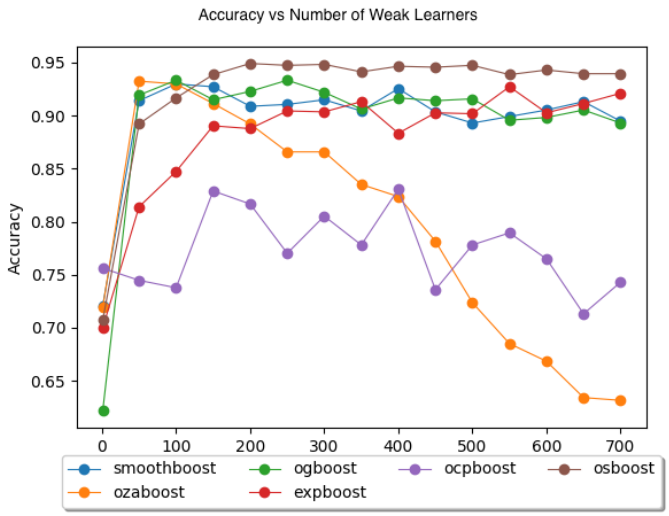

Figure 3: The plot compares the accuracy of various Online Boosting Algorithm as the number of base learners is increased. We have used *Decision Stumps* as the base learners for the *WI Breast Cancer* dataset.

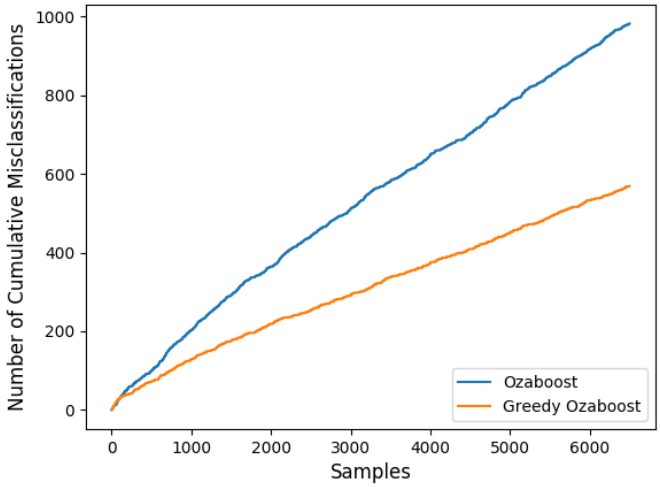

Figure 4: The plot shows a run of GreedyBoost and Ozaboost as each point is shown to the online boosting algorithm. The x-axis refers to the number of training points shown so far and the y-axis refers to the number of misclassifications made so far. Note that the misclassification values are cumulative. We have used *Decision stumps* as the base learners on the *Mushrooms* dataset.

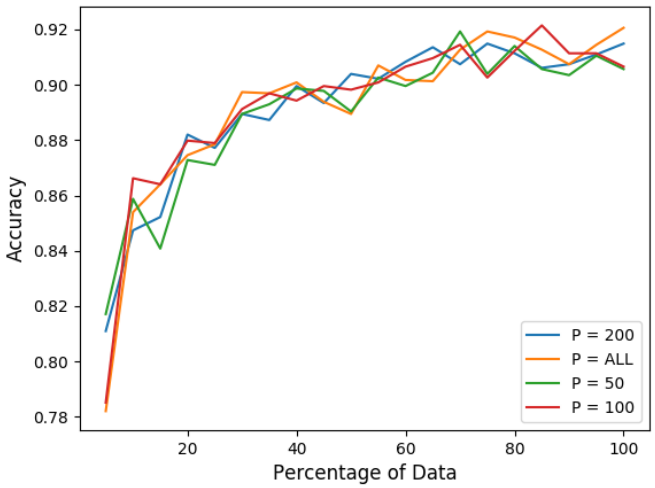

Figure 5: The plot compares the difference of the performance of Greedy Ozaboost with varying number of "past points" (denoted by m) that the new classifier learns from. We have used *Decision stumps* as the base learners on the *breast cancer* dataset.

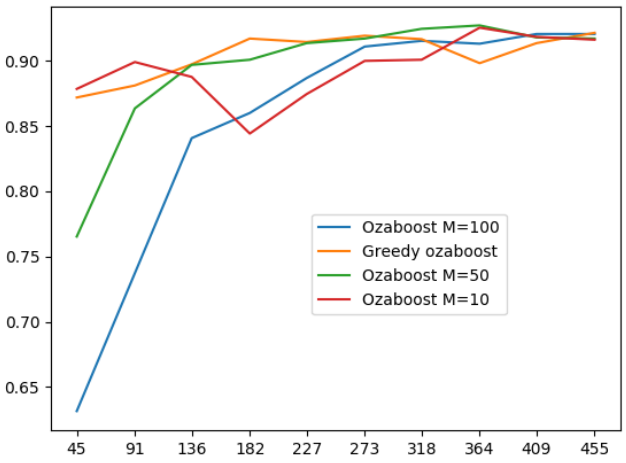

Figure 6: The plot compares the performance of our algorithm Greedy ozaboost to the vanilla ozaboost algorithm with the number of base learners used denoted by M. Note the difference in accuracy when the number of training examples have been small for different values of M. We have used *Decision stumps* as the base learners on the *breast cancer* dataset.

5 Conclusion

For this study, we compared some of the existing approaches to boosting in the online setting and tried to find out if the greedy strategy of adding a new base learner when misclassifications are encountered could be useful. We implemented this technique on 3 different base learners (Perceptrons, Naive Bayes and Decision Stumps) and found that the overall accuracy was indeed improved. A drawback of this approach, however, was the requirement of storing all the previously encountered data points. We discussed a few approaches to overcome the disadvantage and show empirical results for one of them (namely, when a subset of previously encountered data points are stored).

Although we see that implementing this technique with OzaBoost[4] performs well, we have not investigated efficient approaches to find an optimum number of classifiers for the other Online algorithms. Another avenue of research might be employing this technique to replace obsolete base learners to handle drifting data streams (where the underlying concept undergoes change). The source code used for this study can be found at:

`https://github.com/deshanadesai/online-boosting`

References

[1] K. Bache and M. Lichman. *UCI Machine Learning Repository, 2013.*

[2] S.-T. Chen, H.-T. Lin, and C.-J. Lu. *An Online Boosting Algorithm with Theoretical Justifications. ICML 2012, 2012.*

[3] C. Leistner, A. Saffari, P. Roth, and H. Bischof. *On Robustness of On-line Boosting - A Competitive Study.* In Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on, pages 1362–1369, Sept 2009.

[4] N. C. Oza and S. Russell. *Online Bagging and Boosting.* In Artificial Intelligence and Statistics 2001, pages 105–112. Morgan Kaufmann, 2001.

[5] R. A. Servedio. *Smooth Boosting and Learning with Malicious Noise.* Journal of Machine Learning Research, 4:473–489, 2003.

[6] Y. Freund and R. E. Schapire. *Experiments with a New Boosting Algorithm, 1996.*

[7] N. C. Oza. *Online Bagging and Boosting.* In Systems, Man and Cybernetics, 2005 IEEE International Conference on, volume 3, pages 2340–2345 Vol. 3, Oct 2005.

[8] B. Babenko, M.-H. Yang, and S. Belongie. *A Family of Online Boosting Algorithms.* In Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on, pages 1346–1353, Sept 2009.

[9] N. Cesa-Bianchi, Y. Freund, D. Haussler, D. P. Helmbold, R. E. Schapire, and M. K. Warmuth. *How to Use Expert Advice.* J. ACM, 44(3):427–485, May 1997.

[10] Nicolo Cesa-Bianchi, Alex Conconi, and Claudio Gentile. *On the generalization ability of on-line learning algorithms.* IEEE Transactions on Information Theory, 50(9):2050–2057, 2004.

[11] Shivani Agarwal *Online-to-Batch Conversions* Lecture 20, Statistical Learning Theory (Oct 24, 2013)

[12] Charles Mersh *Boosting in the Online Setting*

[13] Yoav Freund *An adaptive version of the boost by majority algorithm* In Proceedings of the Twelfth Annual Conference on Computational Learning Theory.

[14] Mehryar Mohri, Afshin Rostamizadeh, Ameet Talwalkar *Foundations of Machine Learning* The MIT Press, 2012. ISBN 026201825X, 9780262018258. Chapter 6 and 7.

[15] A. Beygilzimer, S. Kale, H. Luo *Optimal and Adaptive Algorithms for Online Boosting*

[16] Gallant, S. I. (1986). *Optimal linear discriminants.* In Proceedings of the Eighth International Conference on Pattern Recognition. Paris, France (pp. 849–852).

[17] Littlestone, N. (1989a). *From on-line to batch learning.* In Proceedings of the Second Workshop on Computational Learning Theory (pp. 269–284).

[18] Helmbold, D. & Warmuth, M. K. (1995). *On weak learning.* Journal of Computer and System Sciences, 50, 551–573.

[19] P. Roth, H. Grabner, D. Skočaj, H. Bischof, and A. Leonardis. *On-line conservative learning for person detection.* In Proc. Workshop on VS-PETS, 2005.

[20] Blum, Avrim, Kalai, Adam, and Langford, John. *Beating the hold-out: Bounds for k-fold and progressive cross- validation.* In Proceedings of the Twelfth Annual Con- ference on Computational Learning Theory, COLT '99, pp. 203–208, 1999.