2021 IEEE International Conference on Big Data and Smart Computing (BigComp) | 978-1-7281-8924-6/20/$31.00 ©2021 IEEE | DOI: 10.1109/BigComp51126.2021.00023

MP-Boost: Minipatch Boosting via Adaptive Feature and Observation Sampling

Mohammad Taha Toghani and Genevera I. Allen

***Abstract*—Boosting methods are among the best general-purpose and off-the-shelf machine learning approaches, gaining widespread popularity. In this paper, we seek to develop a boosting method that yields comparable accuracy to popular AdaBoost and gradient boosting methods, yet is faster computationally and whose solution is more interpretable. We achieve this by developing MP-Boost, an algorithm loosely based on AdaBoost that learns by adaptively selecting small subsets of instances and features, or what we term *minipatches* (MP), at each iteration. By sequentially learning on tiny subsets of the data, our approach is computationally faster than classic boosting algorithms. MP-Boost upweights important features and challenging instances, hence adaptively selects the most relevant minipatches for learning. The learned probability distributions aid in interpretation of our method. We empirically demonstrate the interpretability and comparative accuracy of our algorithm on a variety of binary classification tasks.**

***Index Terms*—Minipatch Learning, AdaBoost, Adaptive Sampling, Internal Validation.**

1. Introduction

Boosting algorithms adaptively learn a series of weak learners that often yield state-of-the-art predictive accuracy for certain machine learning tasks. But computationally, they can be slow for huge datasets. Furthermore, most boosting algorithms such as the popular AdaBoost [1] and gradient boosting (GB) [2] are considered as ”black-box” models which often lack transparency and interpretability [3]. In this paper, our goal is to develop an AdaBoost-based algorithm that is computationally faster than standard boosting models and also yields more interpretable solutions.

We propose to achieve boosting-like ”slow learning” by adaptively sampling tiny subsets of both instances and features simultaneously, something we refer to as a *minipatch learning*. The computational advantages of the ensemble of uniformly random minipatches have been investigated in [4]; however, uniform sampling can be modified with an adaptive procedure to achieve boosting as the importance of different observations and features varies. Furthermore, we can use adaptive sampling to interpret the final model; for example, the SIRUS algorithm [3] suggests how to leverage the frequency of splits in random forest [5] trees to generate an interpretable model with fewer splits. Our method, MP-Boost, will incorporate the advantages of adaptivity in order to learn distributions on the observations and features.

Several studies have suggested sampling schemes for AdaBoost. Works along this line usually target adaptive subsampling of either features or observations. Methods like Laminating [6] employ score-based feature selection techniques where either access all instances or their random subsets. In contrast, algorithms like FilterBoost [7] suggest adaptive oracle subsampling of observations, although it does not subsample features, and the number of selected samples grows per iteration. A series of algorithms, such as stochastic gradient boosting [8], have been proposed to reduce the computational complexity of GB. For example, XGBoost [9] and its variations employ novel structures for decision trees as well as multiple training tricks that extensively optimize the runtime of GB. More recently, the minimal variance sampling algorithm [10] suggests sampling of instances proportional to their gradient values; however, it lacks feature subsampling. Note that all of the existing boosting techniques that employ subsampling, do so for computational reasons. In contrast, our method, MP-Boost which is inspired by AdaBoost, will employ subsampling of features and observations as the boosting mechanism itself; the computational advantages are an added bonus to our approach.

The remainder of the paper is organized as follows. In Section 2, we present the main algorithm. In Section 3, we investigate the advantages of adaptive subsampling, interpretability of MP-Boost, and provide numerical experiments. We end with concluding remarks in Section 4.

2. MP-Boost

Let the data be $(\mathbf{X}, \mathbf{y}) \in \left(\mathbb{R}^{N\times M}, \mathbb{R}^{N}\right)$ for N instances and M features. We focus on binary classification tasks where for each sample $\mathbf{x}_i \in \mathbb{R}^M$, we observe a label $y_i \in \{-1, +1\}$. The goal is learning a classifier $y_i = \operatorname{sgn}\left(F(\mathbf{x}_i)\right)$. We propose a boosting method by training weak learners on tiny subsets of instances and features. We call this tiny subset a *minipatch*, termed based on the use of ”patches” in image processing, and minibatches as small subsamples of observations commonly used in machine learning. Our approach is to take an ensemble of minipatches, or minipatch learning where each minipatch

Mohammad Taha Toghani is with the Department of Electrical and Computer Engineering, Rice University, Houston, TX 77005 USA.(email:mttoghani@rice.edu). Genevera I. Allen is with the Departments of Electrical and Computer Engineering, Statistics, and Computer Science, Rice University, Houston, TX 77005 USA, and Jan and Dan Duncan Neurological Research Institute, Baylor College of Medicine, Houston, TX 77030 USA (email:gallen@rice.edu). *The authors acknowledge support from NSF DMS-1554821, NSF NeuroNex-1707400, and NIH 1R01GM140468.*

is sampled adaptively. Formally, we define a minipatch by $(\mathbf{X}_{\mathcal{R},\mathcal{C}}, \mathbf{y}_{\mathcal{R}})$, where $\mathcal{R}$ is a subset of examples with size n, and $\mathcal{C}$ is a subset of features with size m. By minipatch learning, our algorithm will have major computational advantages for large N and/or M datasets.

We define $\mathcal{H}$ to be the *class of weak learners*, where each $h \in \mathcal{H}$ is a function $h : \mathbb{R}^m \to \{-1, +1\}$. Our algorithm is generic to the type of weak learners, which can be either simple or expressive, although we select decision trees as default weak learners for MP-Boost. We consider both depth-k trees as well as saturated trees that are split until each terminal leaf consists of samples from the same class.

The core of our algorithm uses adaptive sampling of observations to achieve the adaptive slow learning properties of the AdaBoost algorithm. Similar to [10] for GB, MP-Boost subsamples observations according to an adaptive probability distribution. Let $\boldsymbol{p}$ be the probability distribution on observations (i.e., $\sum_{i=1}^{N} p_i = 1$) and initially set $\boldsymbol{p}$ to be uniform ($\mathbf{U}_{[N]}$). We define $\text{Sample}(N, n, \boldsymbol{p})$ as sampling a subset of $[N]$ of size n according to the probability distribution $\boldsymbol{p}$ without replacement.

Let $F : \mathbb{R}^M \to \mathbb{R}$ be the *ensemble function*. Our algorithm selects a minipatch, trains a proper weak learner on it, and computes the summation of weak learners, $F(\mathbf{x}_i)^{(t)} = \sum_{k=1}^{t} h^{(k)} \left((\mathbf{x}_i)_{\mathcal{C}^{(k)}} \right)$. Misclassified samples are more difficult to be learned, so we need to increase their probabilities to be sampled more frequently. Let $\mathcal{L} : \mathbb{R} \times \mathbb{R} \to \mathbb{R}^+$ be a function that measures the similarity between the ensemble outputs and labels, i.e., positive yF yields smaller $\mathcal{L}(y, F)$ and vice versa. MP-Boost assigns a probability proportional to $\mathcal{L}(y_i, F(\mathbf{x}_i)$ to the i^{th} observation. The exponential or logistic losses [1], [11] are suitable options for $\mathcal{L}$.

Full-batch boosting algorithms reweight all of the observations and train a new weak learner on their weighted average in each iteration [1], [11]. In contrast, stochastic algorithms use each sample's frequency to take the effect of its weight into account [7], [10]. We update the probability of the observations similar to FilterBoost [7]. However, FilterBoost increases n during its progress and uses negative sampling, and is thus slower than ours.

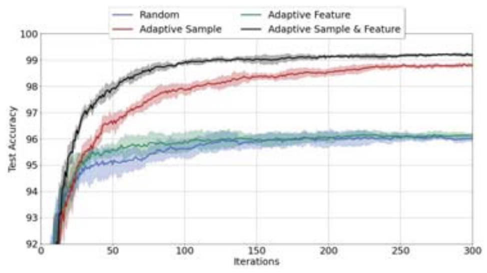
Another major goal is to increase the interpretability of boosting methods. To accomplish this, we also propose to adaptively select features that are effective for learning. Similar to $\boldsymbol{p}$, let $\boldsymbol{q}$ be the probability distribution on features. Our algorithm requires a criterion to compute the importance of the selected m features based on the structure of $\mathcal{H}$. There exist several choices for computing features importance based on $\mathcal{H}$. Some of these inspection techniques are model agnostic, hence proper for MP-Boost to incorporate weak learners from different classes. For example, the permutation importance method [5], [12] shuffling each feature infers its importance according to the difference in the prediction score.

Nevertheless, specific metrics like *impurity reduction score* [13] are defined for decision trees. We utilize this quantity to define our probability distribution over the features. Let $\mathcal{I}^h$ denote the normalized feature importance vector for a weak learner h, wherein each entry determines the relative importance of the corresponding feature compared to other features in the minipatch. In each iteration, $\boldsymbol{q}$ is updated through computing the weighted average of $\boldsymbol{q}$ and $\mathcal{I}$ according to a momentum μ. The hyperparameter $\mu \in (0, 1)$ determines the ratio of exploration vs. exploitation. MP-Boost only modifies the probability of features inside the minipatch, in each iteration.

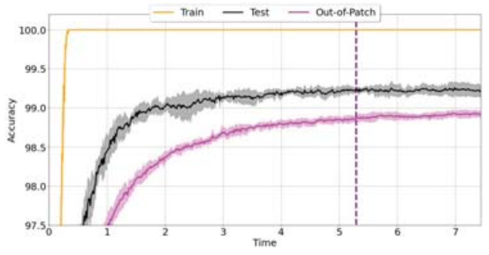
Algorithm 1 MP-Boost

```
MP-Boost $(\mathbf{X}, \mathbf{y}, n, m, \mu)$
Initialization $(t = 0)$:
$\boldsymbol{p}^{(1)} = \mathbf{U}_{[N]}$    // observation probabilities
$\boldsymbol{q}^{(1)} = \mathbf{U}_{[M]}$    // feature probabilities
$F^{(1)}(\mathbf{x}_i) = 0, \quad \forall i \in [N]$    // ensemble output
$G^{(1)}(\mathbf{x}_i) = 0, \quad \forall i \in [N]$    // out-of-patch output
while $\text{Stopping} - \text{Criterion}(\text{oop}^{(t)})$ not met do $t \leftarrow t + 1$
  1) Sample a minipatch:
     a) $\mathcal{R}^{(t)} = \text{Sample}(N, n, \boldsymbol{p}^{(t)})$    // select $n$ instances
     b) $\mathcal{C}^{(t)} = \text{Sample}(M, m, \boldsymbol{q}^{(t)})$    // select $m$ features
     c) $\left(\mathbf{X}^{(t)}, \mathbf{y}^{(t)}\right) = \left(\mathbf{X}_{\mathcal{R}^{(t)},\mathcal{C}^{(t)}}, \mathbf{y}_{\mathcal{R}^{(t)}}\right)$    // minipatch
  2) Train a weak learner on the minipatch:
     a) $h^{(t)} \in \mathcal{H}$: weak learner trained on $\mathbf{X}^{(t)}, \mathbf{y}^{(t)}$
  3) Update outputs:
     a) $F^{(t)}(\mathbf{x}_i) = F^{(t-1)}(\mathbf{x}_i) + h^{(t)} \left( (\mathbf{x}_i)_{\mathcal{C}^{(t)}} \right), \quad \forall i \in [N]$
  4) Update probability distributions:
     a) $p_i^{(t+1)} = \frac{\mathcal{L}(y_i, F^{(t)}(\mathbf{x}_i))}{\sum_{k=1}^{N} \mathcal{L}(y_k, F^{(t)}(\mathbf{x}_k))}, \quad \forall i \in [N]$
     b) $q_j^{(t+1)} = (1 - \mu) q_j^{(t)} + \mu r \mathcal{I}_j^{h^{(t)}}, \quad j \in \mathcal{C}^{(t)}$
        where, $r = \sum_{j \in \mathcal{C}^{(t)}} q_j^{(t)}$
  5) Out-of-Patch Accuracy:
     a) $G^{(t)}(\mathbf{x}_i) = G^{(t-1)}(\mathbf{x}_i) + h^{(t)} \left( (\mathbf{x}_i)_{\mathcal{C}^{(t)}} \right), \quad \forall i \notin \mathcal{R}^{(t)}$
     b) $\text{oop}^{(t)} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}_{\{\text{sgn}(G^{(t)}(\mathbf{x}_i)) = y_i\}}$
end while
Return $\text{sgn}(F^{(T)}), \boldsymbol{p}^{(T)}, \boldsymbol{q}^{(T)}$
```

Finally, many boosting algorithms are designed to run for a fixed number of iterations [2] or use a validation criterion [7] in order to determine when to stop. Internal validation approaches often have better performance and are computationally much faster. For instance, consider the out-of-bag criterion [14] in bagging and random forest that uses internal validation properties without incurring any additional computational cost. Similar to bagging, our MP-Boost has access to out-of-patch instances, which we can use for internal validation. Therefore, for each sample $i \in [N]$, we accumulate the output of weak learners that don't observe it in their minipatch. We also quantify the out-of-patch accuracy, oop, which is a conservative estimate of the test accuracy. Hence it can assist MP-Boost in tracking the progress of the generalization (test) performance inter-



(a) effect of adaptive sampling on test data



(b) oop curve and the stopping criterion

Figure 1: **MNIST (3 vs. 8) trained by MP-Boost.** (a) Advantages of adaptive observation and feature selection. (b) Comparison of train, test, and oop curves.

nally. In a nutshell, observing the oop value, the algorithm finds where it is saturated. Stopping criterion is a heuristic algorithm that takes $\mathrm{oop}^{(t)}$ and compares it with its previous values, thus decides when to stop. In fact, the stopping algorithm follows a general rule; if the current value of oop increases with some margin, then the algorithm needs more time to improve; otherwise, the generalization performance is saturated.

We put all of this together in a summary of our MP-Boost algorithm in Algorithm 1. Notice here that selecting minipatches (step 1) reduces the computational complexity imposed per iteration, thus improves the scalability. Other variations of AdaBoost usually subsample either features or observations while ours exploits both. Therefore, in addition to the predictive model F, our algorithm learns probability distributions $\boldsymbol{p}$ and $\boldsymbol{q}$ that express the importance of observations and features, respectively. Since learning $\boldsymbol{p}, \boldsymbol{q}$ is a part of the iterative procedure, it doesn't incur an extra computational cost. In addition, MP-Boost exploits an internal validation that yields an automatic stopping criterion when the algorithm ceases to learn. Hence, steps (4) and (5) of Algorithm 1 highlight the main differences of MP-Boost with other sampling-based boosting algorithms.

3. Experiments

We begin with an illustrative case study to show how our method works and how it aids the interpretability. Next, we compare our algorithm to other popular boosting and tree-based methods, focusing on accuracy and scalability.

Illustrative Case Study: We use a series of experiments to demonstrate how our algorithm works and show how to interpret the results. We focus our investigations on an explicable binary classification task: detecting digit 3 versus 8 in MNIST [15]. This dataset includes handwritten digits as images of size 28×28. The training data is huge ($N > 10000$) and high-dimensional ($M = 784$). We use cross-validation to tune all hyperparameters, yielding $n = 500$, $m = 30$, and $\mu = 0.5$ as well as logistic loss function.

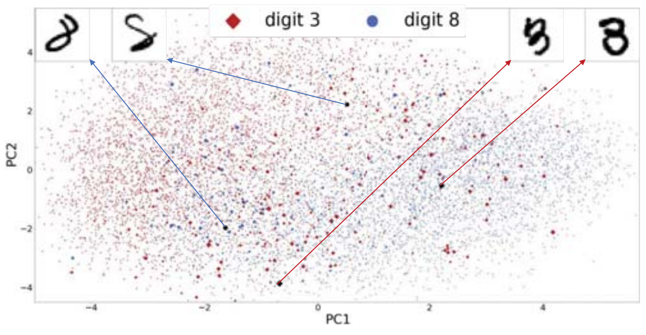
To measure the effect of adaptive observation and/or feature selection, we turn off adaptive updating for $\boldsymbol{p}$ and/or $\boldsymbol{q}$ by replacing them with uniform distributions. We train MP-Boost under the mentioned on MNIST$(3, 8)$ and repeat each experiment 5 times. Figure 1a shows the superiority of joint adaptive sampling of both instances and features in terms of performance on the test data. Further, we compare the training, out-of-patch, and test accuracy curves for MP-Boost in Figure 1b. The dashed line indicates the stopping time of our algorithm based on the oop curve. To observe the behavior of the three curves, we let MP-Boost progresses after the stopping criterion is satisfied. As shown in Figure 1b, and unlike the train curve, the trend in the out-of-patch curve is similar to the test curve. These results demonstrate the power of minipatch learning as well as the advantages of oop curve for internal validation.

Next, we illustrate how to use $\boldsymbol{p}$ and $\boldsymbol{q}$ for the interpretation of instances and features where difficult observations are upweighted in $\boldsymbol{p}$. Hence, we can use $\boldsymbol{p}$ to identify the most challenging samples, yielding a similar type of interpretation commonly employed in support vector machines [16]. As shown in Figure 2a, we project instances on a two-dimensional space using PCA with sizes proportional to $\boldsymbol{p}$. We display four instances with large $\boldsymbol{p}$ values that complied to our expectation are challenging to be classified.

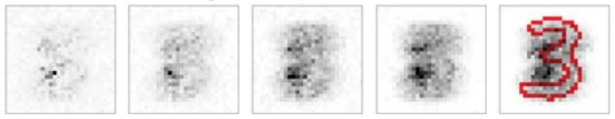
Finally, we show how to use $\boldsymbol{q}$ to find the most important features, and also illustrate how MP-Boost learns these features probability in Figure 2b. Here, the color of each pixel (feature) is proportional to its probability, with darker pixels indicating the feature has been upweighted. We expect a sparse representation for $\boldsymbol{q}$ which matches with our result. Moreover, this example clearly shows how to interpret the most relevant features; in Figure 2b, two regions are darker compared to other pixels corresponding with the complementary area for digit 3 versus 8.

Comparative Empirical Results: Here, we compare the speed and performance of our algorithm with AdaBoost, gradient boosting, and random forest on multiple binary classification tasks. To this end, we select large datasets from UCI machine learning repository [17] and MNIST [15]. As well, we generate a sparse synthetic dataset of two high-dimensional cones.

To be fair, we choose the oracle hyperparameters for every method. To this end, we pick decision trees with different maximum depths (depth $\in \{1, 2, 3, 4, 5, 6, 7\}$) or depth-saturated trees as weak learners. Note that we use scikit-learn [18] modules to implement our algorithm and all competitors so that all time-based comparisons are fair. For our method, we select from the following choices of hyperparameters: $n \in \{50, 100, 200, 500\}$, $m \in \{5, 10, 15, 20, \sqrt{M}\}$, and $\mu \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$.



(a) final probability of instances: $\boldsymbol{p}$



(b) evolution of features importance: $\boldsymbol{q}$

Figure 2: **Probability distribution of MNIST (3 vs. 8) instances and features, trained by MP-Boost.** (a) The size of each sample indicates its relative probability where those close to the boundary have higher probabilities. (b) The color of each pixel (feature) indicates the probability with respect to which the feature is selected. With the progress of MP-Boost, the probability of efficient features increases.

For each dataset, we select the best performance of MP-Boost, versus that of AdaBoost, gradient boosting, and random forest constrained to the runtime of MP-Boost. Table 1 shows the best performance of each algorithm within the training runtime of MP-Boost (fixed). Results indicate that MP-Boost achieves a better performance faster than the other three algorithms. As well, without any runtime constraint, MP-Boost is much faster with a comparable accuracy across a wide variety of datasets[1].

4. Discussion

In this work, we proposed an interpretable boosting approach using adaptive minipatch learning. We showed that our method outperforms standard boosting algorithms in a fixed runtime. Our approach will be particularly useful for huge datasets where both instances and features are large. Further, our MP-Boost algorithm can be used with datasets with sparse or noisy features. In future work, we plan on using the minipatch learning scheme in gradient-based boosting algorithms. Moreover, one could explore other updating schemes for the probability of instances and features. Theoretical analysis of MP-Boost would help to interpret the performance of our algorithm and its properties. Efficient implementations can improve the speed of current MP-Boost for different schemes.

1. Additional results: https://arxiv.org/pdf/2011.07218v1.pdf

TABLE 1: Best test accuracy (%) of MP-Boost, AdaBoost, Random Forest, and Gradient Boosting on binary classification tasks within fixed runtimes.

Dataset	**MP-Boost**	**AdaBoost**	**RF**	**GB**
Cones	$\mathbf{100.0}$	85.56	95.89	94.01
Hill-Valley	$\mathbf{99.45}$	96.28	98.76	96.69
Christine	$\mathbf{74.27}$	69.28	73.75	70.27
Jasmine	$\mathbf{80.03}$	79.53	79.47	79.47
Philippine	$\mathbf{71.01}$	69.64	70.07	69.98
SensIT Vehicle	$\mathbf{86.23}$	83.69	85.98	84.24
Higgs Boson	83.42	82.52	$\mathbf{83.57}$	81.44
MNIST (3,8)	$\mathbf{99.31}$	97.49	98.84	96.81
MNIST (O,E)	$\mathbf{98.15}$	93.78	97.74	93.23
GAS Drift	$\mathbf{99.76}$	96.54	99.64	96.54
DNA	$\mathbf{97.44}$	96.86	$\mathbf{97.44}$	96.81
Volkert	74.14	71.54	$\mathbf{77.95}$	71.9

References

[1] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of computer and system sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[2] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.

[3] C. Bénard, G. Biau, S. Da Veiga, and E. Scornet, "Interpretable random forests via rule extraction," *arXiv preprint arXiv:2004.14841*, 2020.

[4] G. Louppe and P. Geurts, "Ensembles on random patches," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 346–361, Springer, 2012.

[5] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[6] C. Dubout and F. Fleuret, "Adaptive sampling for large scale boosting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1431–1453, 2014.

[7] J. K. Bradley and R. E. Schapire, "Filterboost: Regression and classification on large datasets," in *Advances in neural information processing systems*, pp. 185–192, 2008.

[8] J. H. Friedman, "Stochastic gradient boosting," *Computational statistics & data analysis*, vol. 38, no. 4, pp. 367–378, 2002.

[9] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785–794, ACM, 2016.

[10] B. Ibragimov and G. Gusev, "Minimal variance sampling in stochastic gradient boosting," in *Advances in Neural Information Processing Systems*, pp. 15061–15071, 2019.

[11] J. Friedman, T. Hastie, R. Tibshirani, *et al.*, "Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors)," *The annals of statistics*, vol. 28, no. 2, pp. 337–407, 2000.

[12] A. Altmann, L. Toloşi, O. Sander, and T. Lengauer, "Permutation importance: a corrected feature importance measure," *Bioinformatics*, vol. 26, no. 10, pp. 1340–1347, 2010.

[13] S. Nembrini, I. R. König, and M. N. Wright, "The revival of the gini importance?," *Bioinformatics*, vol. 34, no. 21, pp. 3711–3718, 2018.

[14] L. Breiman, "Out-of-bag estimation," 1996.

[15] Y. LeCun, "The mnist database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998.

[16] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[17] D. Dua and C. Graff, "UCI machine learning repository," 2017.

[18] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.