Improved Multiclass AdaBoost Using Sparse Oblique Decision Trees

Magzhan Gabidolla, Arman Zharmagambetov and Miguel Á. Carreira-Perpiñán
Dept. of Computer Science and Engineering, University of California, Merced, CA, USA
Email: {mgabidolla,azharmagambetov,mcarreira-perpinan}@ucmerced.edu

***Abstract*—Boosting, one of the most effective machine learning frameworks, has attracted an enduring interest since its introduction 30 years ago. The majority of boosting methods use trees as base learner and, while much work has focused on theoretical and empirical variations of boosting, there has been surprisingly little progress on the tree learning procedure itself. To this day, each individual tree is typically axis-aligned (which is ill-suited to model correlations and results in relatively weak classifiers), and is learned using a greedy divide-and-conquer approach such as CART or C5.0, which produces suboptimal trees. We show we can improve boosted forests drastically by making each tree a much stronger classifier. We do this by using sparse oblique trees, which are far more powerful than axis-aligned ones, and by optimizing them using "tree alternating optimization" (TAO), suitably modified to handle the base learner optimization problem dictated by the boosting framework. Focusing on two versions of AdaBoost, we show that the resulting forests not only are consistently and considerably more accurate than random forests or gradient boosting, but that they use a very small number of trees and a comparable number of parameters.**

***Index Terms*—supervised learning, AdaBoost, decision trees**

I. Introduction

Ensembles of decision trees (decision forests), such as Random Forests (RF) [1] and boosted decision trees [2], [3], often achieve state-of-the-art performance and have long been widely used in many applications, such as large scale classification [4], learning to rank [5], computer vision [6], [7], and many others. We focus on boosted forests, for which over 30 years of research have created a huge literature [8]–[10], exploring many variations of the boosting framework, base learner and other topics. At present, it is fair to say that there is no clear winner among the basic boosting frameworks (say, gradient boosting vs AdaBoost) and their variations [8], [11]. Some recent, highly scalable toolkits have become widely used, such as XGBoost [5] or LightGBM [4], but their merits rely more on their highly optimized systems-level implementation rather than on algorithmic issues. Can we improve boosted forests at a more fundamental level?

To this end, we focus on what we think is a glaring deficiency in nearly all established versions of boosted forests, namely that they use trees that have limited modeling power and that are poorly optimized. Let us see this in detail. At a basic level, boosting consists of a definition of how the next tree in the ensemble should be learned. Specifically, what the weight of each training point is, and what loss function the tree should optimize. The weights are initially equal and then are updated using a formula based on the training error obtained by the last learner which emphasizes misclassified points. The loss function can take many forms (0/1 loss, exponential loss, etc.). Using this procedure, one repeatedly adds trees to the ensemble until a stopping criterion is satisfied, usually based on a validation set and a limit on the number of trees (overfitting is rarely observed but does occur with boosting). Importantly, notice that *each boosting framework defines the base learner optimization precisely*. But this tree optimization is then solved in an approximate way, using a divide-and-conquer algorithm such as CART [12] or C5.0 [13]. Although many variations exist of this topic (such as the choice of purity criterion, stopping criterion, minimum number of points in a node to be split, etc.), the tree learning consists of a greedy procedure where nodes are recursively split by locally optimizing a purity criterion (after which the node parameters are fixed henceforth). As is well known [8], such algorithms produce suboptimal trees. This is not just because they fix nodes in a greedy (and sometimes approximate) way. *CART-type algorithms largely ignore the actual optimization problem defined by boosting*: the purity criterion is different from the loss function. (An exception is tree stumps, where the optimization can actually be exact, but they are extremely weak classifiers.) Also, the points' weights are typically approximated by resampling. The reason for this suboptimal procedure is the difficulty of learning decision trees, which are non-differentiable functions. This has also limited the trees to be axis-aligned (where each node tests a single feature), because then the purity at each split can be optimized exactly (by trying all feature/threshold combinations); the purity optimization is much harder for oblique nodes. However, axis-aligned trees have trouble modeling correlated features and usually grow enormously, particularly with high-dimensional features.

As we will see, this gap in the tree learning is critical in boosting. We address it as follows. 1) We use oblique trees (having hyperplane decision nodes), which are more powerful and much smaller (in depth and number of nodes). 2) We use a sparsity penalty on the hyperplane parameters, so we can still find good partitions but using just a subset of features (hence fewer parameters per node). And 3) we use a recently proposed algorithm, TAO, that (suitably modified to handle the AdaBoost optimization problem) is very effective in training individual trees. (TAO also improves over CART with axis-aligned trees [14], but oblique trees are much better.) A reader familiar with ensemble learning may object that a better tree

optimization could result in worse boosted forests, because 1) boosting was originally motivated to work with weak learners, and 2) lack of diversity will kill any ensemble. We will show that, while TAO indeed produces individual sparse oblique trees which are much more accurate than axis-aligned CART trees, the boosting framework is still able to inject sufficient diversity so the resulting ensemble is much better than a single tree—and much better than state-of-the-art boosted forests. In addition, the resulting forests contain very few trees and a comparable number of parameters. Although previous attempts at using oblique trees in boosting exist, we believe ours is the first one that succeeds at making them clearly effective. As an additional advantage over many boosted forest frameworks for K-class classification, we can use a single K-class tree as base learner rather than K trees, which greatly reduces the number of trees and promotes parameter sharing.

In the rest of the paper, we review related work (section II), the boosting algorithms we use (section III) and the TAO algorithm including our modifications (section IV). We then evaluate our approach experimentally (section V) and analyze it from different perspectives (section VI).

II. Related work

Since the introduction of the first practical boosting algorithm (AdaBoost) [15], much has been written to explain its theory and many more boosting algorithms were developed. The book by [2] provides a comprehensive overview of the theory of boosting. Statistical treatments of the subject can be found in [8], [16]. We briefly review some of the notable boosting algorithms and decision tree learning algorithms.

a) Boosting algorithms: The first AdaBoost algorithm was designed only for binary classification, and since then many multiclass extensions have been developed. The most straightforward of these is AdaBoost.M1 [2]. From the perspective of stagewise additive modeling, [17] proposed SAMME, which has a close resemblance to AdaBoost.M1, and we will describe those in detail later. AdaBoost.MH [18] is a multiclass, multilabel boosting algorithm designed to minimize the Hamming loss. AdaBoost.MR [18] solves the ranking problem, and under certain assumptions it simplifies to AdaBoost.M2. Most of the implementations of AdaBoost.MH and AdaBoost.MR are based on one-vs-all or one-vs-one binary reductions. LogitBoost [19] is derived from minimizing the logistic loss using stagewise additive modeling, and at each boosting step it uses K regression base learners (where K is the number of classes). [20] give a game-theoretic perspective of multiclass boosting and introduce AdaBoost.MM, where base learners just need to minimize a cost matrix. [21] propose a new formulation of multiclass boosting using margin enforcing loss functions and an optimal set of codewords, and present two generic boosting procedures: CD-MCBoost and GD-MCBoost. Finally, gradient boosting [3] fits approximate functional gradient descent in stagewise additive modeling, and with several efficient variations and optimized implementations [4], [5], [22], they are now widely used as state-of-the-art off-the-shelf classifiers.

b) Optimizing a single tree: Although boosting can use many types of base learner, axis-aligned trees are by far the most common choice, because of their relatively good performance when ensembled, and their speed of training (e.g. in the widely used XGBoost and LightGBM toolkits). We briefly overview the large literature about decision tree learning. The most widely used algorithms are based on greedy top-down induction, which is suboptimal, as mentioned in the introduction. By recursively splitting a node to minimize some impurity measure (e.g. Gini index or entropy), these algorithms partition the input space to have nodes as pure as possible (receiving instances of the same class). This usually results in large trees that significantly overfit, so nodes are pruned based on some cost-complexity measure. Widespread decision tree algorithms such as CART [12], ID3 [23] or C4.5 and C5.0 [13] are based on slight variations of this basic procedure. Note that none of these algorithms use a well-defined objective function, rather they apply an (essentially arbitrary) impurity criterion to split nodes. When used for boosting, the procedure is the same, possibly with small modifications (e.g. in the impurity measure); the training points' weights are often simulated by resampling [2], [18], [19]. Most of the research literature and software packages on decision trees, and especially those used in boosting, consider only axis-aligned partitions of the input space. Oblique tree optimization was considered early on [12], [24], and some works have proposed boosting oblique trees [25], [26], but the accuracy of either single or boosted oblique trees has not been found to improve consistently over that of axis-aligned trees, while the number of parameters is much bigger. This situation has changed with the recently proposed TAO algorithm, described later, which is able to do a much better, global (rather than greedy) optimization of trees of the axis-aligned, oblique, or other types [14], [27], [28]. TAO has also been successfully combined with bagging [29], [30], gradient boosting [31] and others [32], [33].

III. Decision tree boosting: AdaBoost

As just reviewed, there exists a huge variety of boosting frameworks and, while we believe that all of them will benefit from better optimized oblique trees, we can only evaluate a few in this paper. We focus on the oldest and widely adopted framework, AdaBoost, of which we consider two of its most popular variations: AdaBoost.M1 and SAMME. As our experiments will show, we are able to improve them so they outperform the most recent and efficient implementations of boosting, in particular XGBoost.

The pseudocode of SAMME is presented in Algorithm 1. In both AdaBoost.M1 and SAMME, each boosting step involves training a base learner (a tree in our case) to minimize the weighted misclassification loss (I is an indicator function):

$$E = \sum_{n=1}^{N} w_n \, I(y_n \neq \mathbf{T}_t(\mathbf{x}_n)) \tag{1}$$

where $\mathbf{T}_t(\cdot)$ is the prediction of the current (tth) tree and w_n is the weight per input instance (initially equal to $w_n = \frac{1}{N}$

Algorithm 1 SAMME pseudocode using TAO trees. The pseudocode for AdaBoost.M1 has minor modifications.

input: training set $\{(\mathbf{x}_n, y_n)\}_{n=1}^N$ where $y_n \in \{1, \dots, K\}$;
base learner $\mathbf{T}$; number of boosting steps T;
shrinkage factor η;
initial weights (per instance): $\{w_n = \frac{1}{N}\}_{n=1}^N$;
for $t = 1$ **to** T **do**
Train a TAO tree ($\mathbf{T}_t$) on the training set with the current weights and loss function (2) (see Algorithm 2);
Obtain predictions: $\{\hat{y}_n\}_{n=1}^N \leftarrow \mathbf{T}_t(\{\mathbf{x}_n\}_{n=1}^N)$;
Compute weighted misclassification loss E in eq. (1);
if $E \geq 1 - \frac{1}{K}$ **then**
set $T = t - 1$; exit loop;
end if
Compute $\alpha_t = \eta \cdot (\log \frac{1-E}{E} + \log(K-1))$;
Set $w_n \leftarrow w_n \cdot \exp(\alpha_t \cdot I(y_n \neq \hat{y}_n))$;
renormalize $w_1, \dots, w_N$ to sum to 1;
end for
return $F(\mathbf{x}) = \arg\max_k \Sigma_{t=1}^T \alpha_t \cdot I(\mathbf{T}_t(\mathbf{x}) = k)$;

for $n = 1, \dots, N$). We provide the details of training the tree base learner in the next section. For boosting to proceed, the training loss E (eq. (1)) must be lower than some threshold, which differs in these two versions: in AdaBoost.M1 the threshold is $\frac{1}{2}$, and in SAMME it is $1 - \frac{1}{K}$ (where K is the number of classes). If this weak learning condition is not met, the boosting procedure will terminate. In the multiclass case ($K > 2$), AdaBoost.M1 has more stringent requirement on base learners than SAMME, but with fairly strong base learners this should not be an issue [2]. Once the base learner is trained, both algorithms estimate the weight α_t (see Algorithm 1) of the current base learner. AdaBoost.M1 differs from SAMME here in that there is no $\log(K-1)$ term in the sum. The final step in a single boosting iteration is to change the distribution of the instance weights for the next base learner. Both algorithms increase the weights of the misclassified instances and decrease the weights of the correctly classified instances. The predictions from all base learners are combined through a weighted majority vote to produce the final prediction.

IV. Optimizing a single tree with TAO

Since each boosting step involves optimizing a single tree, we now show how to apply the TAO algorithm, suitably modified, to train it within the boosting framework. We consider a binary decision tree for K-class classification having a predetermined structure, usually a complete binary tree of depth Δ (although any other structure is possible, e.g. random or obtained from a CART tree). Each decision (or internal) node i has a thresholding function $f_i(\mathbf{x}; \boldsymbol{\theta}_i)$ with parameters $\boldsymbol{\theta}_i$ which sends instance $\mathbf{x}$ to the corresponding child (either left or right). Throughout this paper, we consider trees with oblique splits only, i.e., the decision function is linear, $f_i(\mathbf{x}; \boldsymbol{\theta}_i) = \mathbf{w}_i^T \mathbf{x} + w_{i0}$, so it sends $\mathbf{x}$ to the right child if $f_i(\mathbf{x}; \boldsymbol{\theta}_i) \geq 0$ and to the left otherwise. Each leaf is a constant predictor which outputs the same class label regardless of the input $\mathbf{x}$. The prediction of a tree (denoted by $\mathbf{T}(\mathbf{x}; \boldsymbol{\Theta})$) for instance $\mathbf{x}$ is obtained by routing $\mathbf{x}$ from the root to exactly one leaf and applying its prediction. Here, $\boldsymbol{\Theta}$ denotes the set of parameters of all tree nodes. Next, consider the following optimization problem at each boosting step:

$$\min_{\boldsymbol{\Theta}} E(\boldsymbol{\Theta}) = \sum_{n=1}^N w_n \, L(y_n, \mathbf{T}(\mathbf{x}_n; \boldsymbol{\Theta})) + \lambda \sum_{\text{nodes } i} \phi(\boldsymbol{\theta}_i) \quad (2)$$

where ϕ is a regularization term (with hyperparameter $\lambda \geq 0$) that penalizes the parameters $\boldsymbol{\theta}_i$ of each node. In this paper, we use a sparsity penalty $\phi(\boldsymbol{\theta}_i) = \|\boldsymbol{\theta}_i\|_1$. L is a classification loss for data point $\mathbf{x}_n$ with ground-truth label y_n. Notice that in the case of 0/1 loss, L is identical to eq. (1) (with w_n as a weight per data point coming from the AdaBoost algorithm) but having an extra regularization term. Therefore, TAO directly attempts to minimize the boosting objective function. Eq. (2) differs from the original TAO formulation due to the AdaBoost weights and thus it requires a new derivation of the TAO reduced problem, which we describe below (that is, we do not resample the training set according to the weights).

TAO optimizes globally all node parameters rather than greedily. It works in a way that is similar to how most other machine learning models are trained: one first fixes the model architecture (the tree structure, e.g. a complete binary tree of given depth) and iteratively optimizes its parameters by monotonically decreasing $E(\boldsymbol{\Theta})$. With most models this is done via gradient-based methods, but this is not possible with decision trees, which are nondifferentiable. Instead, TAO applies alternating optimization (over the parameters at each node) in a certain way, and this guarantees that after each iteration $E(\boldsymbol{\Theta})$ decreases or stays constant. Besides, with an ℓ_1 penalty over the nodes' parameters, nodes are automatically pruned when all its weights become 0. Hence, it is convenient to use a deep enough complete tree structure and let TAO prune it as necessary.

TAO is based on two theorems which we describe briefly (refer to [27], [28] for details) and adapt for the boosting framework. Define the *reduced set* $\mathcal{R}_i \subset \{1, \dots, N\}$ of node i (decision node or leaf) as the training instances that reach i given the current tree parameters.

a) Separability condition: Consider any pair of nodes i and j that are not descendants of each other. Since the tree makes hard decisions, the reduced sets of i and j are disjoint. Furthermore, we assume that the parameters are not shared across nodes: $\boldsymbol{\theta}_i \cap \boldsymbol{\theta}_j = \varnothing$ for all i, j such that $i \neq j$. Then E *separates* over the parameters of nodes i and j: $E(\boldsymbol{\Theta}) = E_i(\boldsymbol{\theta}_i) + E_j(\boldsymbol{\theta}_j) + E_{\text{rest}}(\boldsymbol{\Theta}_{\text{rest}})$, where $\boldsymbol{\Theta}_{\text{rest}}$ is the parameters of all other nodes that do not include i and j.

b) Reduced problem: Given the separability condition, we can optimize a non-descendant set of nodes independently. Consider first optimizing eq. (2) over a leaf node i. This is equivalent to training i's parameters $\boldsymbol{\theta}_i$ on its reduced set, i.e., solving the original weighted classification problem on its reduced set. With a constant-label leaf, this is solved exactly by the *weighted* majority vote.

Optimizing over a decision node is more complicated. Assume the parameter values of all nodes except i are fixed.

Algorithm 2 Training a base classifier using our modified TAO algorithm.

input: training set $\{(\mathbf{x}_n, y_n)\}_{n=1}^N$;
  initial tree $\mathbf{T}(\cdot; \boldsymbol{\Theta})$ of depth Δ;
  AdaBoost weights $\{w_n\}_{n=1}^N$;
repeat
  for depth $d = 0$ **to** Δ **do**
    for $i \in$ nodes of $\mathbf{T}$ at depth d **do**
      if i is a leaf **then**
        $y_i \leftarrow$ weighted majority vote on reduced set $\mathcal{R}_i$;
      else
        $\boldsymbol{\theta}_i \leftarrow$ minimizer of weighted binary classification problem (4) with current weights $\{w_n\}_{n=1}^N$;
      end if
    end for
  end for
until max # iterations or E decrease ¡ set tolerance
postprocess $\mathbf{T}$: prune dead subtrees of $\mathbf{T}$;
return $\mathbf{T}$

Using the separability condition, we can rewrite eq. (2) for decision node i equivalently as a function of $\boldsymbol{\theta}_i$:

$$\min_{\boldsymbol{\theta}_i} E_i(\boldsymbol{\theta}_i) = \min_{\boldsymbol{\theta}_i} \sum_{n \in \mathcal{R}_i} w_n \, l(f_i(\mathbf{x}_n; \boldsymbol{\theta}_i)) + \lambda \phi_i(\boldsymbol{\theta}_i) \qquad (3)$$

where $\mathcal{R}_i$ is i's reduced set. Since $f_i \in \{\text{right,left}\}$ can only have two possible values, we define $l(\cdot)$ as the loss incurred by choosing the right or left subtree. Hence, we can rewrite eq. (3) as the following equivalent optimization problem:

$$\min_{\boldsymbol{\theta}_i} \sum_{n \in \mathcal{R}_i} w_n \, L(\overline{y}_n, f_i(\mathbf{x}_n; \boldsymbol{\theta}_i)) + \lambda \phi_i(\boldsymbol{\theta}_i) \qquad (4)$$

where L is the same 0/1 classification loss mentioned before and $\overline{y}_n \in \{\text{right,left}\}$ is a "pseudolabel" indicating the child which gives a lower value of E for instance $\mathbf{x}_n$ under the current tree. Problem (4) above is a *weighted* 0/1 loss binary classification problem, unlike the traditional 0/1 loss reduced problem in the original TAO paper. Optimizing eq. (4) is NP-hard [34], [35], but we can approximate it with a convex surrogate loss such as the logistic loss. A number of efficient solvers exist which can handle this problem. In our experiments, we use LIBLINEAR [36], which can also handle weights per data point.

TAO pseudocode is in Algorithm 2. Our TAO algorithm processes the tree nodes in breadth-first order, and repeatedly trains a binary classifier (decision node) and a K-class classifier (leaf). All the nodes at the same depth are trained independently and in parallel due to the separability condition. A single TAO iteration consists of one pass through all the nodes in the tree. The main hyperparameters of a TAO tree are the depth Δ and the sparsity factor λ.

V. Experiments: overall performance of the Boosted TAO trees

We perform extensive comparison across standard machine learning benchmarks to show effectiveness of our proposed method. We tried to pick a diverse group of datasets (see Table I) from various domains (e.g. computer vision, NLP), with different sample sizes (N), number of classes (K), feature types (dense, sparse, etc.) and dimensions (D). We compare against multiple forest-based methods: Random Forests (RF), gradient boosting (XGBoost implementation) and conventional AdaBoost (SAMME version). We denote AdaBoost.M1 as "M1-CART" and SAMME as "S-CART". We also compare with published results of a few forest variations proposed in the literature: Alternating Decision Forest (ADF) [37], shallow Neural Decision Forest (sNDF) [38] and refined Random Forests (rRF) [39]. Finally, we give the result of training a single CART tree [12] for reference. As for the boosted TAO trees, "S-TAO" uses SAMME as a boosting algorithm and sparse oblique TAO trees as base learners, whereas "M1-TAO" employs AdaBoost.M1 boosting algorithm.

TABLE I
Datasets used: number of train and test instances (N_{train}, N_{test}), number of features D, number of classes K.

Dataset	N_{train}	N_{test}	D	K
Letter	16 000	4 000	16	26
MNIST	60 000	10 000	784	10
Char74k	66 707	7 400	64	62
R8	5 485	2 189	400	8
RCV1	15 564	518 571	47 236	53

A. Experiment details

Experiments were performed on Intel(R) Xeon(R) CPU E5-2699 v3 @ 2.30GHz. For implementations that support parallel processing, we set the number of threads to 8. For the experiments that we perform, we repeat them 5 times, and report the mean result (and standard deviation for the test error).

a) Implementation: We implemented TAO in C++. As an initial tree, we take an oblique complete tree of depth Δ with random weights. For solving a reduced problem in a decision (internal) node, we use an ℓ_1-regularized logistic regression solver in LIBLINEAR (v2.30 with support for instance weights) [36]. We parallelize the training of nodes at a given depth. We also implemented a Python interface for TAO. We implemented both AdaBoost.M1 and SAMME in Python.

For S-CART and RF, we use the Python implementation in scikit-learn [40]. For M1-CART we implement the boosting algorithm ourselves, but for the CART base learner we use scikit-learn. The hyperparameters that we consider during cross validation are `learning_rate`, `n_estimators` and `max_depth`. We use XGBoost (v0.81, Python package) and tune `eta`, `max_depth` and `num_boost_round`.

b) Reproducibility: ℓ_1-regularized logistic regression in LIBLINEAR has randomized behavior, but for the fixed random seed it is deterministic. However, in TAO, when we train nodes in parallel, we cannot control the order in which the *rand()* function in LIBLINEAR is called, and, as a consequence, our TAO implementation is not deterministic. The difference one obtains on the test error from this random behavior is not significant. Unless otherwise stated, the number of TAO iterations I throughout the experiments is 20 and the

learning rate η is 0.1. There is no random behavior in both AdaBoost.M1 and SAMME.

c) Datasets: Letter dataset is available in the UCI Machine Learning Repository [41]. We obtain the Char74k dataset from [37]. R8 consists of top 8 classes of the Reuters-21578 text categorization dataset. We train the Doc2Vec model available from the `gensim` package [42] on all the document collections and obtain 400-dimensional word-embedding features. RCV1 dataset is obtained from the LIBSVM multiclass data collection[1].

d) Estimation of model size: We calculate the number of parameters of the whole tree ensemble as the sum of the number of parameters of each individual trees. We estimate the number of parameters of a single tree as the sum of the number of parameters of all the nodes in the tree. In axis-aligned trees, the number of parameters of an internal node is 2, and in oblique trees, this equals the number of nonzero weights. In both types of trees, the number of parameters in a constant leaf is 1. We estimate inference FLOPS for a single tree as the number of parameters an input instance encounters in the root-to-leaf path, and for the whole ensemble we just take a sum of those. We calculate the inference FLOPS for each instance in the training set, and report the average result.

B. Results

Tables II and III show the results (sorted by decreasing test error). First of all, we compare a single TAO tree ($T = 1$ in the tables) with the goal to demonstrate the better optimization done by TAO. The results on MNIST and Letter convincingly show that the accuracy of a TAO tree far beats that of a CART tree. Moreover, it is remarkable that the performance of a single TAO tree is often comparable to that of some baseline forests such as RF. This can be clearly seen in R8 and RCV1 datasets. These results show that TAO is able to find good approximate optima of the decision tree optimization problem which makes the resulting tree a strong enough learner.

Let us now consider ensemble of trees ($T > 1$). Our results for baselines (RF, conventional AdaBoost and XGBoost) generally coincide with previous works [37], [39]. Out of these three methods, XGBoost seems to show the best performance, although it depends on the dataset and sometimes, carefully tuned RF can show similar results (e.g. Char74). It is worth to mention that XGBoost (and most of other gradient boosting methods) add K trees at each iteration [3] generating K times more trees, and thus the resulting ensemble size can be large. Next, let us focus on boosted TAO trees. We can observe that both versions of the boosted TAO trees *consistently improve over RF, AdaBoost, XGBoost and other methods from literature showing the lowest test error in all datasets*. This is extreme in Letter and R8 where the performance gap between successor method is quite large. Moreover, the resulting forests require fewer trees and they are shallower. However, each node of a tree has $D + 1$ parameters (weight vector and bias) since we are using oblique splits. Whereas, an axis-aligned node has only two (feature index and bias). However, we enforce ℓ_1 penalty in node optimization to make node parameters sparse. Consequently, it affects to the number of parameters and table shows that the boosted TAO oblique trees often produce much compact models.

[1] http://csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass.html

TABLE II
BOOSTED TAO TREES COMPARED WITH BASELINE AND STATE-OF-THE-ART FOREST METHODS: RANDOM FORESTS (RF), XGBOOST AND SAMME (S-CART); ADDITIONALLY, WE REPORT PUBLISHED RESULTS FROM THE LITERATURE: ADF [37], SNDF [38] AND RRF [39]. WE HAVE TWO VERSIONS OF THE BOOSTED TAO TREES: "S-TAO" USES SAMME AND "M1-TAO" USES ADABOOST.M1. WE REPORT THE TEST ERROR (%, AVG±STDEV OVER 5 REPEATS), NUMBER OF PARAMETERS, NUMBER OF TREES T AND MAX DEPTH OF THE FOREST Δ.

	Forest	E_{test} (%)	#parameters	T	Δ
MNIST	CART	12.11±0.04	6k	1	50
	TAO	5.25±0.20	24k	1	8
	RF	3.05±0.06	1M	100	46
	S-CART	2.96±0.05	6M	1k	30
	RF	2.84±0.06	10M	1k	48
	sNDF	2.80±0.12	22M	80	10
	ADF	2.71±0.10	(3.6M)	100	25
	XGBoost	2.67±0.00	0.3M	1k	8
	S-CART	2.28±0.02	13M	1k	16
	M1-TAO	2.09±0.04	0.8M	30	8
	rRF	2.05±0.02	(160k)	100	25
	XGBoost	1.94±0.00	0.6M	10k	8
	S-TAO	1.93±0.02	0.8M	30	8
	M1-TAO	1.74±0.02	2.6M	100	8
	S-TAO	1.67±0.04	2.6M	100	8
Letter	CART	13.06±0.15	3k	1	27
	TAO	9.59±0.31	10k	1	11
	XGBoost	4.30±0.00	0.4M	2.6k	10
	RF	3.77±0.06	0.4M	100	34
	ADF	3.52±0.12	(1M)	100	25
	RF	3.44±0.09	4.2M	1k	36
	XGBoost	3.35±0.00	0.8M	26k	6
	S-CART	2.83±0.15	0.7M	100	16
	rRF	2.98±0.15	(180k)	100	25
	sNDF	2.92±0.17	2.4M	70	10
	S-CART	2.58±0.09	6.7M	1k	16
	M1-TAO	1.85±0.09	0.2M	30	11
	S-TAO	1.79±0.07	0.2M	30	11
	M1-TAO	1.40±0.09	0.6M	100	11
	S-TAO	1.38±0.03	0.6M	100	11

VI. EXPERIMENTS: ANALYSIS

The success of our proposed boosting method encourages us to dive into the details of the algorithm and answer important questions, such as: "Why do stronger learners perform better than weak learners in this particular case?", "Does overfitting occur and are TAO boosted trees robust against that?", "What are the most important hyperparameters in our algorithm and how sensitive is it to hyperparameter changes?". In this section, we investigate these issues and also study practicalities that are specific to boosted TAO trees, such as weighting schemes and training time.

A. TAO vs CART trees as base learners

The left column of fig. 1 shows the effect of replacing CART trees with TAO trees as base learners in AdaBoost.M1 and SAMME. The margin of improvement of TAO trees upon CART on the test error is quite significant for the illustrated 3 datasets. It should be noted that we are selecting optimal hyperparameters for both CART and TAO trees using cross

TABLE III
SIMILAR TO TABLE II, BUT FOR R8, RCV1 AND CHARS74K DATASETS

	Forest	E_{test} (%)	#parameters	T	Δ
R8	TAO	6.64±1.04	3k	1	7
	RF	6.16±0.35	93k	100	27
	S-CART	5.85±0.07	61k	100	20
	RF	5.57±0.56	0.9M	1k	27
	XGBoost	5.34±0.00	51k	800	6
	S-CART	5.11±0.09	0.6M	1k	20
	XGBoost	4.89±0.00	83k	8k	6
	S-TAO	3.12±0.10	0.6M	100	7
	M1-TAO	3.06±0.14	0.6M	100	7
RCV1	S-CART	>24 hours runtime		1k	16
	RF	19.84±0.42	1M	100	233
	RF	18.78±0.37	10M	1k	233
	TAO	17.96±0.03	3M	1	12
	S-CART	16.81±0.38	0.4M	100	100
	XGBoost	13.97±0.00	0.3M	5.1k	30
	XGBoost	13.06±0.00	0.7M	51k	8
	M1-TAO	11.81±0.02	32M	100	9
	S-TAO	11.74±0.03	18M	100	9
Char74k	TAO	23.94±0.37	46k	1	12
	XGBoost	18.08±0.00	1.7M	6.2k	50
	S-CART	17.86±0.15	2.5M	100	60
	RF	17.33±0.14	2.5M	100	65
	S-CART	16.77±0.08	14M	1k	16
	XGBoost	16.70±0.00	6M	62k	12
	ADF	16.67±0.21	(4M)	100	25
	RF	16.61±0.14	26M	1k	65
	sNDF	16.04±0.20	58M	200	12
	rRF	15.40±0.10	(1.1M)	100	25
	S-TAO	13.14±0.19	3.6M	100	12
	M1-TAO	12.81±0.11	4.7M	100	12

validation, and we find that TAO favors more shallow trees than CART.

This result clearly supports the claim that TAO finds trees closer to the optimum of the objective function of the base learner than greedy top down induction in CART. This also contradicts the common belief that boosting weak learners is as strong as a boosting in which the learner's error can be made arbitrarily small [43], [9]. If the goal is to achieve more accurate forests with AdaBoost.M1 and SAMME, then the objective function of the base learner should be optimized better.

B. Comparison of runtime and number of trees

The left column of fig. 1 also shows the results of two different tree ensembles: Random Forests and gradient boosting (XGBoost). Clearly, for the fixed T, boosted TAO trees achieve considerably better test error than all the other forest methods. This again demonstrates the benefit of better optimization of base learners in ensemble models.

One could argue that because of the iterative nature of TAO and the sequential learning of boosting, the time to train T number of TAO trees will be much slower than inducing T number of CART trees. Though TAO is in general slower than CART, it has the flexibility to control the tradeoff between training time and optimization in the number of TAO iterations I. By setting I to a lower value, we compare boosted TAO trees against different tree ensembles as a function of training time (right column of fig. 1). Even with smaller I (I = 5 for Letter and MNIST, I = 10 for R8), boosted TAO trees achieve significantly lower test error than other forest methods almost

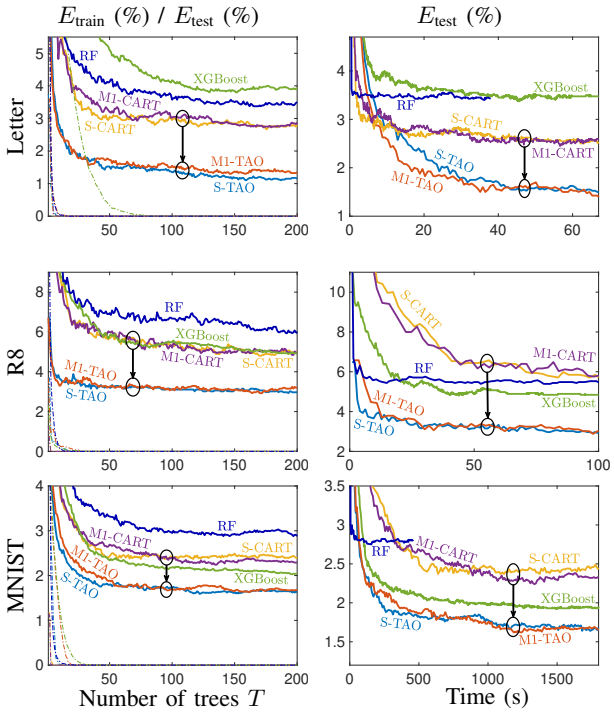

Fig. 1. Comparison between different tree ensembles as a function of the number of trees T (left) and training time (right). Solid lines—test errors, dashed lines—train errors. "M1"—AdaBoost.M1, "S"—SAMME, RF—random forest. All methods except "*-CART" use parallel training with 8 threads. In the left column, we limited the number of trees to 10^4, that is the reason RF stops early on Letter and MNIST.

from the start or in a reasonably short time. This is because TAO needs fewer boosting steps to reach better test error than the other tree ensembles.

C. Hyperparameter search

The most important hyperparameters in TAO are the number of iterations I and tree depth Δ. In fig. 2 we explore how these two hyperparameters affect the performance of S-TAO along with the other two hyperparameters of boosting: the shrinkage factor η and the number of trees T. In the suppl. mat. we provide similar exploration for M1-TAO.

The shrinkage factor η has a significant effect on the test error. Larger values of η tend to result in noisy and less accurate ensembles, while smaller values tend to require more boosting steps T to reach the minimum test error. This is in accordance with the empirical findings in statistical literature. For the default shrinkage factor we use $\eta = 0.1$.

Comparison of different I suggests that more TAO iterations result in more accurate ensembles. This is expected, because higher values of I correspond to better optimization. Comparison of different Δ indicate that deeper trees produce better models, though overly deep trees are more likely to overfit.

D. Does overfitting occur?

Overfitting is known to occur in ensembles with weak classifiers (e.g. decision stumps), but with trees, bagged and boosted forests are both often said not to overfit [1], [2]. This contradicts statistical wisdom that overfitting will occur if using a large enough model *that is optimized well*. Indeed,

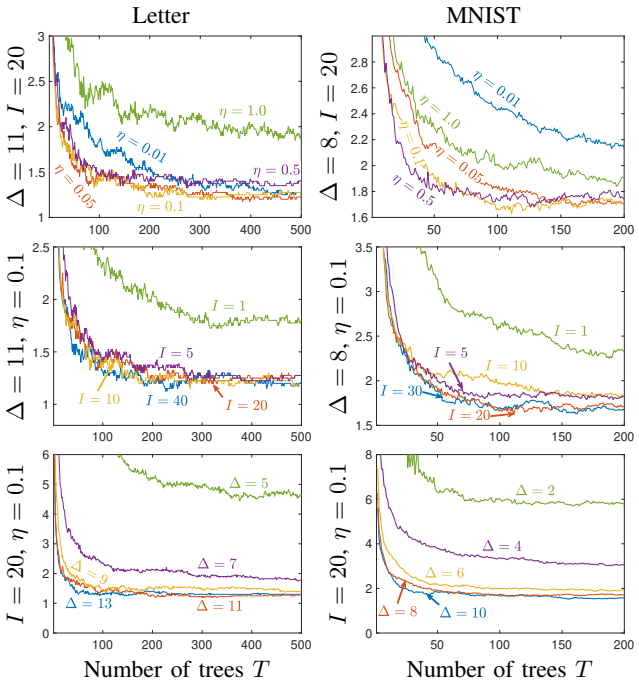

Fig. 2. Test error of S-TAO on Letter and MNIST as a function of 4 hyperparameters: tree depth Δ, number of TAO iterations I, shrinkage factor η and number of trees T. Each column fixes two and varies the other two.

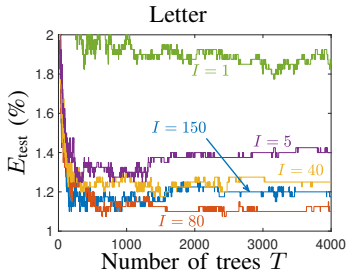

Fig. 3. S-TAO with large number of trees to detect whether overfitting happens on Letter dataset with different number of TAO iterations (I). Overfitting can be clearly observed for $I = 5$ and $I = 150$.

for the Letter dataset, [2] shows that, for a boosted forest of C4.5 trees, the test error decreases almost monotonically and slowly, reaching 3.1% with $T = 1000$ trees.

With CART or C4.5 trees, it is not possible to optimize a desired objective function and the resulting trees are very noisy estimates of the optimal tree. With TAO we do have the ability to learn trees that are closer to the optimum of the objective function, as evidenced in section IV. Still, it is hard to see any overfitting in fig. 2 as one increases I or the model size (Δ, T).

In fig. 3 we explore this using higher values of I and T for the Letter dataset (with a relatively large depth $\Delta = 11$). This eventually makes the model size much bigger than the dataset size (i.e., the model is vastly overparameterized). Now we see that overfitting eventually occurs, for example the test error curve for $I = 150$ touches bottom at around $T = 150$ and then increases (surprisingly, this also happens for $I = 5$). However, the increase is small (from 1.1% to 1.2%) and all curves continue to oscillate slightly as T increases. This is similar to the behavior of SGD with a fixed-size model, which

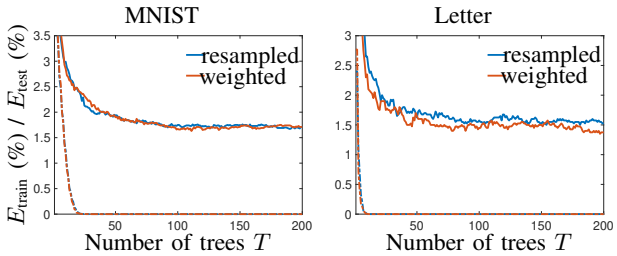

Fig. 4. Different approaches to solve weighted classification problem for trees (eq. (1)) on MNIST and Letter. Solid lines—test err., dashed lines—train err.

will converge to a neighborhood of the minimizer and oscillate around it if the step size is small but does not tend to zero. However, the model size here keeps growing monotonically.

We conclude the following. Firstly, we corroborate the benefit of a better optimization (effected by TAO): the more TAO iterations, the smaller T needs to be, and the lower the error achievable. Second, while hard to get, overfitting does occur, but in the form of oscillations around the minimum test error value, rather than a steady error increase.

E. Weighting or resampling?

Each boosting step involves training a tree to minimize *weighted* misclassification error (eq. 1) and depending on how tree optimization works, some algorithms cannot directly handle such losses. Therefore, some implementations use so called "resampling" technique as an approximation which samples N instances with replacement and with probabilities assigned for each data point. $w_1, w_2, ..., w_n$ play role of that probabilities and indeed, they can be a good approximation since each weight is interpreted as an "importance" of that instance at the current boosting step. Moreover, $\sum_n w_n = 1$. On the other hand, our oblique trees can directly handle weighted losses and thus, we do not need any "resampling". But we still investigate these two approaches applied to our boosting algorithm and present our findings in fig. 4. Results show that directly solving the optimization problem gives better performance. However, sometimes the difference might be marginal as it can be observed on MNIST.

VII. Conclusion

Decades of research on boosted trees have considered many variations of the boosting procedure and of the tree learning procedure. However, the latter has been restricted to approximate, divide-and-conquer algorithms (such as CART or C5.0) that grow an axis-aligned tree in a greedy way and are known to be quite suboptimal. And yet, even with individual trees that are quite inaccurate classifiers, the resulting boosted forest often achieves state-of-the-art classification accuracy in many tasks. We have shown that we can consistently achieve even more accurate forests, sometimes considerably so, by improving the individual tree optimization (using the TAO algorithm) and by using more powerful trees (sparse oblique). The resulting forests achieve lower test error using fewer, smaller trees than CART-based AdaBoost, random forests and gradient boosting. Given the widespread use of forests, this is a remarkable result which has immediate practical application.

While our experiments are only for (two versions of) AdaBoost, we expect that the advantages of using sparse oblique trees trained with TAO will carry over to other variations of boosting such as gradient boosting, and to other tasks such as regression or ranking, which we are working on.

Acknowledgments. Work funded in part by NSF award IIS–2007147. We thank Chih-Jen Lin (National Taiwan University) for helping us to make some custom modifications to the LIBLINEAR code.

REFERENCES

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

[2] R. E. Schapire and Y. Freund, *Boosting. Foundations and Algorithms*, ser. Adaptive Computation and Machine Learning Series. MIT Press, 2012.

[3] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001.

[4] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "LightGBM: A highly efficient gradient boosting decision tree," in *NIPS*, I. Guyon, U. v. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. MIT Press, Cambridge, MA, 2017, pp. 3146–3154.

[5] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. of the 22nd ACM SIGKDD 2016*, San Francisco, CA, Aug. 13–17 2016, pp. 785–794.

[6] P. Viola and M. J. Jones, "Robust real-time face detection," *Int. J. Computer Vision*, vol. 57, no. 2, pp. 137–154, May 2004.

[7] A. Criminisi and J. Shotton, *Decision Forests for Computer Vision and Medical Image Analysis*, ser. Advances in Computer Vision and Pattern Recognition. Springer-Verlag, 2013.

[8] T. J. Hastie, R. J. Tibshirani, and J. H. Friedman, *The Elements of Statistical Learning—Data Mining, Inference and Prediction*, 2nd ed., ser. Springer Series in Statistics. Springer-Verlag, 2009.

[9] Z.-H. Zhou, *Ensemble Methods: Foundations and Algorithms*, ser. Chapman & Hall/CRC Machine Learning and Pattern Recognition Series. CRC Publishers, 2012.

[10] L. I. Kuncheva, *Combining Pattern Classifiers: Methods and Algorithms*, 2nd ed. John Wiley & Sons, 2014.

[11] C. Bentéjac, A. Csörgő, and G. Martínez-Muñoz, "A comparative analysis of gradient boosting algorithms," *AI Review*, vol. 54, no. 3, pp. 1937–1967, Mar. 2021.

[12] L. J. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Belmont, Calif.: Wadsworth, 1984.

[13] J. R. Quinlan, *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[14] A. Zharmagambetov, S. S. Hada, M. Gabidolla, and M. Á. Carreira-Perpiñán, "Non-greedy algorithms for decision tree optimization: An experimental comparison," in *IJCNN'21*, Virtual event, Jul. 18–22 2021.

[15] Y. Freund and R. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *J. Computer and System Sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[16] P. Bühlmann and T. Hothorn, "Boosting algorithms: Regularization, prediction and model fitting," *Statistical Science*, vol. 22, no. 4, pp. 477–505 (with discussion, pp. 506–522), 2007.

[17] J. Zhu, H. Zou, S. Rosset, and T. Hastie, "Multi-class AdaBoost," *Statistics and Its Interface*, vol. 2, no. 3, pp. 349–360, 2009.

[18] R. E. Schapire and Y. Singer, "Improved boosting algorithms using confidence-rated predictions," *Machine Learning*, vol. 37, Dec. 1999.

[19] J. Friedman, T. Hastie, and R. Tibshirani, "Additive logistic regression: A statistical view of boosting," *Annals of Stats*, vol. 28, no. 2, pp. 337–407, Apr. 2000.

[20] I. Mukherjee and R. Schapire, "A theory of multiclass boosting," *JMLR*, vol. 14, no. 1, pp. 437–497, Feb. 2013.

[21] M. Saberian and N. Vasconcelos, "Multiclass boosting: Margins, codewords, losses, and algorithms," *JMLR*, vol. 20, no. 137, pp. 1–68, 2019.

[22] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, "CatBoost: Unbiased boosting with categorical features," in *NEURIPS*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31. MIT Press, Cambridge, MA, 2018, pp. 6638–6648.

[23] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, no. 1, pp. 81–106, 1986.

[24] S. K. Murthy, S. Kasif, and S. Salzberg, "A system for induction of oblique decision trees," *J. AI Research*, vol. 2, pp. 1–32, 1994.

[25] C. Yu and D. B. Skillicorn, "Parallelizing boosting and bagging," Dept. of Computing and Information Science, Queens University, Tech. Rep. 2001–442, Feb. 2001.

[26] C. Henry, R. Nock, and F. Nielsen, "Real boosting *a la Carte* with an application to boosting oblique decision tree," in *Proc. of the 20th IJCAI'07*, Hyderabad, India, Jan. 6–12 2007, pp. 842–847.

[27] M. Á. Carreira-Perpiñán and P. Tavallali, "Alternating optimization of decision trees, with application to learning sparse oblique trees," in *Advances in NEURIPS*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31. MIT Press, Cambridge, MA, 2018, pp. 1211–1221.

[28] M. Á. Carreira-Perpiñán, "The Tree Alternating Optimization (TAO) algorithm: A new way to learn decision trees and tree-based models," 2021, arXiv.

[29] A. Zharmagambetov and M. Á. Carreira-Perpiñán, "Smaller, more accurate regression forests using tree alternating optimization," in *Proc. of the 37th ICML 2020*, H. Daumé III and A. Singh, Eds., Online, Jul. 13–18 2020, pp. 11 398–11 408.

[30] M. Á. Carreira-Perpiñán and A. Zharmagambetov, "Ensembles of bagged TAO trees consistently improve over random forests, AdaBoost and gradient boosting," in *Proc. of the 2020 ACM-IMS Foundations of Data Science Conference (FODS 2020)*, Seattle, WA, Oct. 19–20 2020, pp. 35–46.

[31] M. Gabidolla and M. Á. Carreira-Perpiñán, "Pushing the envelope of gradient boosting forests via globally-optimized oblique trees," in *Proc. of the 2022 IEEE Computer Society Conf. CVPR'22*, New Orleans, LA, Jun. 19–24 2022.

[32] A. Zharmagambetov, M. Gabidolla, and M. Á. Carreira-Perpiñán, "Improved boosted regression forests through non-greedy tree optimization," in *IJCNN'21*, Virtual event, Jul. 18–22 2021.

[33] ——, "Improved multiclass AdaBoost for image classification: The role of tree optimization," in *IEEE ICIP 2021*, Online, Sep. 19–22 2021, pp. 424–428.

[34] K.-U. Hoffgen, H. U. Simon, and K. S. Vanhorn, "Robust trainability of single neurons," *J. Computer and System Sciences*, vol. 50, no. 1, pp. 114–125, Feb. 1995.

[35] L. Pitt and L. G. Valiant, "Computational limitations on learning from examples," *Journal of the ACM*, vol. 35, no. 4, pp. 965–984, Oct. 1988.

[36] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "LIBLINEAR: A library for large linear classification," *JMLR*, vol. 9, pp. 1871–1874, Aug. 2008.

[37] S. Schulter, P. Wohlhart, C. Leistner, A. Saffari, P. M. Roth, and H. Bischof, "Alternating decision forests," in *Proc. of the 2013 IEEE Computer Society Conf. CVPR'13*, Portland, OR, Jun. 23–28 2013, pp. 508–515.

[38] P. Kontschieder, M. Fiterau, A. Criminisi, and S. Rota Buló, "Deep neural decision forests," in *Proc. 15th ICCV'15*, Santiago, Chile, Dec. 11–18 2015, pp. 1467–1475.

[39] S. Ren, X. Cao, Y. Wei, and J. Sun, "Global refinement of random forest," in *Proc. of the 2015 IEEE Computer Society Conf. CVPR'15*, Boston, MA, Jun. 7–12 2015, pp. 723–730.

[40] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, "Scikit-learn: Machine learning in Python," *JMLR*, vol. 12, pp. 2825–2830, Oct. 2011, available online at https://scikit-learn.org.

[41] M. Lichman, "UCI machine learning repository," http://archive.ics.uci.edu/ml, 2013.

[42] R. Řehůřek and P. Sojka, "Software framework for topic modelling with large corpora," in *Proc. LREC 2010 Workshop on New Challenges for NLP Frameworks*, Valletta, Malta, May 22 2010, pp. 45–50.

[43] R. E. Schapire, "The strength of weak learnability," *Machine Learning*, vol. 5, no. 2, pp. 197–227, Jun. 1990.

[44] S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., *Advances in NEURIPS*, vol. 31. MIT Press, Cambridge, MA, 2018.

[45] *Int. J. Conf. Neural Networks (IJCNN'21)*, Virtual event, Jul. 18–22 2021.