Multiple Scale Boosting[1]

Charles A. Pehlivanian

Abstract

We introduce an integer resolution measure on the classical Gradient Boosting iteration step, allowing for specification of a modified recursive, multi-scale algorithm. The resulting classifier relies on the Multiple Cluster Detection objective in the context of Consecutive Partition solvers which allows the classifier update to optimally identify regions for further focus in subsequent iterations. In addition, the use of synthetic loss functions is derived based on specification of the update function rather than the loss. In this way a real-value parameterized family of loss functions is developed, generalizing well-known cases, aiding in loss engineering tasks.

1 Consecutive partition solvers

Let $n \in \mathbb{N}$ be positive and set $\mathcal{V} = \{1, \ldots, n\}$. Let $X = \{x_1, \ldots, x_n\}$, $Y = \{y_1, \ldots, y_n\}$ be finite real sequences with $y_i > 0$, for all i. In spatial scan statistics ([1], [12]) applications, for example, the tuples (x_i, y_i) correspond to realized occurrence and estimated baseline attributes, respectively, associated with a spatial location indexed by i. Denote by $\mathcal{D} = \mathcal{D}_{X,Y}$ the set of tuples $\{(x_i, y_i)\}_{i=1}^n$ associated with X, Y. $\mathcal{D}$ is assumed to have an order induced by a *priority* function on the product $X \times Y$.

Definition 1. *A priority function is a function $g \colon \mathbb{R} \times \mathbb{R}^+ \to \mathbb{R}$ that induces an ordering on the dataset $\mathcal{D}$. We refer to $g(x, y) = \frac{x}{y}$ as the standard priority function.*

The ordering imposed on $\mathcal{D}$ is not antisymmetric, so is not a total ordering. A subset S of $\mathcal{V}$ can be identified with the point $(\sum_{i \in S} x_i, \sum_{i \in S} y_i)$ in $\mathbf{R}^2$, called the *partition point* of S. Set $p_\emptyset = (0, 0)$. In this way the sequences X, Y is associated with a point set in $\mathbf{R}^2$ by considering the union of all partition points. The notion of score function comes from the spatial scan statistics literature.

Definition 2. *A score function is a C^2 smooth $f(x, y) \colon \mathbb{R} \times \mathbb{R}^+ \to \mathbb{R}$, nondecreasing in x, with continuous extension to the origin in any wedge $\mathcal{W}(\mu_1, \mu_2) = \left\{(x, y) : y > 0, \mu_1 \leq \frac{x}{y} \leq \mu_2\right\}$, for $-\infty < \mu_1 \leq \mu_2 < \infty$, with the extension $\hat{f}$ satisfying $\hat{y}(0, 0) = 0$. If f is of the form $f(x, y) = x^\alpha y^{-\beta}$ for some $\alpha, \beta > 0$, then f is a rational score function.*

The original definition does not require the C^2 smoothness assumption above. Rational score functions will play a central role in this paper. The regularity condition on $\mathcal{W}$ in wedges guarantees a continuous extension to the origin on any positive cone in the upper half-plane, for rational score functions the constraint corresponds to the constraint $\alpha > \beta$. A score function f induces a real-valued set function on $2^\mathcal{V}$ by defining $F(S) = f(\sum_{i \in S} x_i, \sum_{i \in S} y_i)$, for $S \subseteq \mathcal{V}$. For a given f we call the F so defined as the *discrete scoring* or *discrete score function* of f. Finally let $\mathcal{C}(X, Y)$ be the convex hull in $\mathbf{R}^2$ of all partition points of the pair (X, Y).

Unless otherwise stated, the sets X, Y will be assumed to be indexed in (standard) priority order, i.e., $g(x_1, y_1) \leq \cdots \leq g(x_n, y_n)$, where g is the (standard) priority function.

For a given score function f, $T \leq n$ we are interested in *maximal partitions* for F, i.e., solutions to the program

$$\mathcal{P}^* = \underset{\mathcal{P}=\{S_1 \ldots, S_T\}}{\operatorname{argmax}} \sum_{j=1}^{T} F(S_j) = \underset{\mathcal{P}=\{S_1, \ldots, S_T\}}{\operatorname{argmax}} \sum_{j=1}^{T} f\Big(\sum_{i \in S_j} x_i, \sum_{i \in S_j} y_i\Big) \tag{1}$$

[1] C++ software package available at https://github.com/pehlivanian/MultiBoost.git

which for a rational score function becomes

$$\mathcal{P}^* = \underset{\mathcal{P}=\{\S_1 \ldots, \S_T\}}{\operatorname{argmax}} \sum_{j=1}^{T} \frac{(\sum_{i\in S_j} x_i)^\alpha}{(\sum_{i\in S_j} y_i)^\beta} \tag{2}$$

Definition 3. *A consecutive subset of $\mathcal{V}$ is a subset of the form $\{j, j+1, \ldots, k\}$ for some $1 \leq j \leq k \leq n$. Consecutive subsets of the form $\{1, \ldots, j\}$, $\{k, \ldots, n\}$ are called ascending consecutive, descending consecutive, respectively. A consecutive partition $\mathcal{P} = \{S_1, \ldots, S_t\}$ is a partition of $\mathcal{V}$ such that each S_i is a consecutive subset.*

Letting $\mathcal{S}_c$ be the set of consecutive partitions of $\mathcal{V}$, it is easy to see that $|\mathcal{S}_c| = \frac{n(n+1)}{2}$.

Definition 4. *The score function f satisfies the Consecutive Partitions Property (CPP) if the solution*

$$\mathcal{P}^* = \underset{\mathcal{P}=\{S_1,\ldots,S_T\}}{\operatorname{argmax}} \sum_{j=1}^{T} f(\sum_{i\in S_j} x_i, \sum_{i\in S_j} y_i)$$

is a consecutive partition, for all X, Y. f satisfies the Weak Consecutive Partitions Property (WCPP) if the solution

$$\mathcal{P}^* = \underset{\substack{|\mathcal{P}|=\{S_1,\ldots,S_R\} \\ |R|\leq T}}{\operatorname{argmax}} \sum_{j=1}^{R} f(\sum_{i\in S_j} x_i, \sum_{i\in S_j} y_i)$$

is a consecutive partition, for any X, Y. f satisfies CPP($\mathbf{R}^+$), WCPP($\mathbf{R}^+$) if it satisfies CPP, WCPP, respectively, for $X \subseteq \mathbf{R}^+$, Y.

If there is a correspondence between f and its discrete scoring function F via (X, Y), we may say that F satisfies CPP, etc., if f does. It was shown in [19] that if f satisfies some simple properties, the solution to 1 is realized at a consecutive partition.

Theorem 1. *Let $f(x, y)$ be a convex, subadditive score function. Then the scoring funcion F of f satisfies CPP. If f is convex, then F satisfies $WCPP$.*

We can also easily characterize the partition scan statistics corresponding to the rational score functions:

Corollary 1. *Let F be discrete scoring function for $f(x, y) = x^\alpha y^{-\beta}$, for constants $\alpha, \beta > 0$, and priority function $g(x, y) = \frac{x}{y}$.*

- *If $\alpha - \beta = 1$, and α is even then F satisfies CPP. If $\alpha - \beta \geq 1$, and α is even then F satisfies $WCPP$.*
- *If $\alpha - \beta = 1$ then F satisfies $CPP(\mathbf{R}^+)$. If $\alpha - \beta \geq 1$ then F satisfies $WCPP(\mathbf{R}^+)$.*

Proof. We note that the Hessian of f has principal minors

$$M_1 = \alpha(\alpha - 1)x^{\alpha-2}y^{-\beta},$$
$$M_2 = \alpha\beta(\alpha - \beta - 1)x^{2(\alpha-1)}y^{-2(\beta+1)},$$

so that for $x \in \mathbf{R}^+$, f is convex iff $\alpha - \beta \geq 1$. f is subadditive iff $\alpha - \beta \leq 1$. For $x \in \mathbf{R}$ the additional condition that α is even is required. The result then follows from Theorem (1) above. □

In addition a dynamic programming approach provides an order $\mathcal{O}(n^2 t)$ solution to (2), with storage requirements proportional to nt, whenever CPP or $WCPP$ is satisfied, is outlined in [19] and will be given in a later section.

1.1 Applications to gradient boosting

Given a set of quadratic polynomials $p_i(z) = \frac{1}{2}h_i z^2 + g_i z + c_i$, with $h_i > 0$ for all $i \in \mathcal{V}$, the minimum values attained are $-\frac{g_i^2}{2h_i} + c_i$ at the values $z_i^* = -\frac{g_i}{h_i}$. Define the subset polynomial, for and $S \subseteq \mathcal{V}$

$$\begin{aligned} p_S(z) = \sum_{i \in S} p_i(z) &= \frac{1}{2}\left(\sum_{i \in S} h_i\right) z^2 + \left(\sum_{i \in S} g_i\right) z + \left(\sum_{i \in S} c_i\right) \\ &= \frac{1}{2} H_S z^2 + G_S z + C_S \end{aligned}$$

Given a set $\{g_i\}$, $\{h_i\}$ which defines the polynomials above, with all $h_i > 0$, and $T \leq n$, we seek a partition $\mathcal{P}^* = \{S_1, \ldots, S_T\}$ of $\mathcal{V}$ and a set of points $\{z_1^*, \ldots, x_T^*\}$ that minimizes the sum of the subset polynomials p_{S_j} evaluated at z_j. Therefore

$$\begin{aligned} \mathcal{P}^* = \underset{\substack{\mathcal{P}=\{S_1 \ldots, S_T\} \\ \{z_1^*, \ldots z_j^*\}}}{\operatorname{argmin}} \sum_{j=1}^{T} p_{S_j}\left(z_j^*\right) &= \underset{\mathcal{P}=\{S_1 \ldots, S_T\}}{\operatorname{argmin}} \sum_{j=1}^{T} \left(\frac{1}{2} H_S (z_j^*)^2 + G_S z_j^* + C_S\right) \\ &= \underset{\mathcal{P}=\{S_1 \ldots, S_T\}}{\operatorname{argmax}} \sum_{j=1}^{T} \frac{G_j^2}{H_j} \\ &= \underset{\mathcal{P}=\{S_1 \ldots, S_T\}}{\operatorname{argmax}} F\left(S\right), \end{aligned}$$

where F is the discrete scoring of the function $f\left(x, y\right) = \frac{x^2}{y}$ on $\mathcal{C}\left(X, Y\right)$, taking $\left(X, Y\right) = \left(\{g_i\}_i, \{y_i\}_i\right)$, and $\{z_1^*, \ldots, z_n^*\} = \left\{-\frac{G_1}{H_1}, \ldots, -\frac{G_T}{H_T}\right\}$ once $\{S_1, \ldots, S_T\}$ are determined.

By the results of the previous section, F attains its maximum on a consecutive partition under the priority function $g\left(x, y\right) = \frac{x}{y}$. We take this approach to modify the split-finding step in the Gradient Boosting algorithm ([9], [10], [17]). The original gradient boosting algorithm used only first-order information from the loss function, no Hessian computations were used. The more recent libraries and treatments ([20], [7], [8], [13]) include second-order Hessian matrix information (almost always nonzero except for the diagonal). It is this algorithm that we will refer to as gradient boosting.

Let $\left(X, y\right) = \left(\{x_i\}_{i=1}^n, \{y_i\}_{i=1}^n\right)$, $x_i \in \mathbf{R}^l$, $y_i \in \mathbf{R}$ be a dataset of features, labels on which we wish to fit a classifier for out of sample prediction. The iterative boosting step at step t attempts to find a classifier update f_t which minimizes the loss expression

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) \tag{3}$$

where $l(y_i, \hat{y}_i)$ is an arbitrary convex loss function, $\hat{y}_i^{(t-1)}$ is the result of the step-t classifier applied to x_i, and $\Omega(f_t)$ is an l^2-regularization term given by $\Omega\left(f\right) = \sum_{w \in F(f)} \lambda w^2 + \gamma |F\left(f\right)|$ for $F\left(f\right)$ the set of distinct leaf values for the classifier f. This is the iterative step of a greedy algorithm, often f_t is assumed to come from a family of classifiers under suitable parameterization. In the classical gradient boosting approach, the loss in equation (3) is approximated by a quadratic polynomial, and a tree-based classifier (decision tree, random forest, etc.) is used for the classifier f_t. Denoting by $\{(S_j, w_j)\}_{j=1}^t$ the tuples of leaf level sets (sets of constant leaf value for f_t) and leaf values w_j, equation (3) becomes

$$\mathcal{L}^{(t)} = \sum_{j=1}^{T} [(\sum_{i \in S_j} g_i) w_j + \frac{1}{2}(\sum_{i \in S_j} h_i + \lambda) w_j^2] + \gamma t \tag{4}$$

where $g_i = \frac{\partial l(y_i, \hat{y}^{(t-1)})}{\partial \hat{y}^{(t-1)}}$, $h_i = \frac{\partial^2 l(y_i, \hat{y}^{(t-1)})}{\partial^2 \hat{y}^{(t-1)}}$. Obtaining the optimal leaf sets and values is equivalent to solving

$$\mathcal{P}^* = \underset{\mathcal{P}=\{S_1 \ldots, S_T\}}{\operatorname{argmax}} \sum_{j=1}^{T} \frac{(\sum_{i \in S_j} g_i)^2}{\sum_{i \in S_j} h_i + \lambda} + \gamma t \tag{5}$$

and setting $w_j = -\frac{\sum_{i \in S_j} g_i}{\sum_{i \in S_j} h_i + \lambda}$ (for details see [7]). The heart of our iterative step finds the optimal $\{(S_j, w_j)\}_{j=1}^{t}$ via the *CPP* framework, and uses these to fit a tree-based classifier f_t at each step on the restricted set of leaf values and level sets. Bayes consistency requires the form of the final classifier after N iterations to be

$$f(y) = \sum_{i=1}^{N} f_i(y) \tag{6}$$

There are many popular implementations of gradient boosting and modifcations available - XGBoost [7], LightGBM [13], CatBoost [20] to name a few. The general approach there is to fit a tree-based classifier to (3) directly and control classifier fitting by traditional pruning methods. Our approach, now described, attempts to pre-specify the target values in an optimal way.

1.2 Multiscale boosting

We tie the choice of the time t update classifier f_t in (3) to the solution of the program (5) by obtaining the leaf values w_j, $j = 1, \ldots T$ and level sets S_j which will be used as terminal leaf values and nodes for a tree-based classifier (decision tree, random forest, etc.). The set $\{(w_j, S_j)\}_{j=1}^{T}$ comes from an exact solution of (5), which we call a T *resolution* of the iterative step at time t.

The authors in [19] describe a method for choosing an optimal partition size T for the general problem, but we will not take that approach here. Instead, motivated by fast, iterative multiple-resolution algebraic or grid solvers [5] we propose a novel multi-resolution approach to the the traditional gradient boosting algorithm. Given a dataset of size n, we will successively solve scale T_i resolutions for some sequence $\{T_i\}_{i=1}^{t}$, $1 \leq T_i \leq n$, and propagate solutions across these 'layers'. The approach is similar to those used in iterative multigrid solvers for discretizations resulting from finite-difference methods or finite elements. The multigrid approach relies on one or more relaxation steps on a fine discretization of the domain, after which the residual is restricted to a coarser grid and relaxation is performed there. Once relaxation on the coarsest grid is complete, the solution is propagated 'upward' to a finer grid, and the process is repeated. The exact form of the recursion can take several forms, usually referred to as V-cycles, W-cycles, etc. see [5].

Specify $\{T_i\}$, $i = 1, \ldots, m$ with each $1 \leq T_i \leq n$. These will serve as partition sizes for each of the m iteration steps. We do not assume the sequence is monotonic nondecreasing or nonincreasing, although that works well in practice. Starting with $i = 1$, we solve the T_1 scale resolution problem for y based on a loss associated with the initial prediction $\hat{y}_0$, obtaining leaf values and level sets $\{(w_1 1, S_1), \ldots (w_1 l, S_1 l)\}$. These are used as specifications of leaf values and level sets for a tree-based (Decision Tree, Random Forest, etc.) classifier C_1. In this way C_1 is fit on a summarization or representation of the values $y - \hat{y}$ that is optimal in the sense of scoring. Denote by $\hat{y}_i$ the prediction generated by the classifier at step i. At step $i + 1$, the T_{i+1} scale resolution problem is solved, and the classifier C_{i+1} is fit. The new prediction is $\hat{y}_i + \hat{y_{i+1}}$. After iteration m produces a classifier at for the T_m scale problem, the adjusted estimate is passed upward to the T_{i-1} scale problem, and the process continues. This is the familiar V-cycle, and is described in algorithm 1 below.

The approach is similar in spirit to algebraic multigrid solvers. The normal Gradient Boosting approach does not rely on gradient descent applied to the classifier f_t as it is nonsmooth; only the gradient and hessian of the loss function are calculated, which are usually implemented in closed form without reliance on auto differentiation. This we are able to link the layers together by true *target propagation* (see [**?**]). There are, however, fundamental differences between the classic multigrid

approach and our proposed approach. The classic approach relies on a discretization of space-time dimensions over which the (usually) finite difference scheme is defined and the desired solution is viewed as a projection of a continuous ambient function to the discrete grid. The resulting approximation are thus amenable to frequency or Von Neumann analysis. This analysis is usually used to justify the approach, as typical Jacobi or Gauss-Seidel relaxations show specific spectral profiles which are effective on particular frequencies of residuals when applied on particular meshes. In our case there is no temporal aspect to the data, the solution is not viewed as a restriction of a continuous $\mathbf{R}^n$-defined function. The notions of low-frequency and high-frequency components of the solutions which are so effectively addressed by the classic multigrid method are analagous to low-level and high-level features in the classification case, aspects which are difficult to quantify. It is believed that the success of deep neural network models on non-tabular data is partly due to their ability to discriminate at multiple levels of resolution [11]. By intelligent choice of $T_1, \dots, T_m$ we may be able to enhance a tree-based classification scheme and vary the resolution in the same way.

Algorithm 1 Consecutive Partition Solution

function DISCOPTSOLUTION(x, y, t)
    # x: X vector
    # y: Y vector
    # t: size of partition

    # Base case ($t' = 1$)
    $(C_x, C_y) = (0, 0)$
    for $j \in \{n, n-1, \dots, 1\}$ **do**
        $(C_x, C_y) = (C_x, C_y) + (x_j, y_j)$
        $F^*(j, 1) = f(C_x, C_y)$
        $\rho(j, 1) = n + 1$

    # Iterative step
    for $t' \in \{2, \dots, t\}$ **do**
        for $j \in \{1, \dots, (n + 1 - t')\}$ **do**
            $F^*(j, t') = -\infty$
            $(C_x, C_y) = (0, 0)$
            for $k \in \{j, \dots, (n + 1 - t')\}$ **do**
                $(C_x, C_y) = (C_x, C_y) + (x_k, y_k)$
                $F^{j,k} = f(C_x, C_y) + F^*(k + 1, t' - 1)$
                if $F^{j,k} > F^*(j, t')$ **then**
                    $F^*(j, t') = F^{j,k}$
                    $\rho(j, t') = k + 1$

    # Recover the highest-scoring consecutive partitioning for each t'
    for $t' \in \{1, \dots, t\}$ **do**
        $scores[t'] = F^*(1, t')$.
        $partitions[t'] = []$
        $j = 1$;
        for $t'' \in \{t', (t' - 1), \dots, 1\}$ **do**
            $j_{next} = \rho(j, t'')$
            Append $[j,\ j_{next} - 1]$ to $partitions[t']$
            $j = j_{next}$
    return *scores*, *partitions*

Algorithm 2 Multiscale Gradient Boosting Fit

function COMPUTEGH($y, \hat{y}, l$)
    # y: labels,
    # $\hat{y}$: prediction
    # l: loss function

    for $i \in \{1, \ldots, n\}$ **do**
        $g_i = \frac{\partial l(y_i, \hat{y}^{(t-1)})}{\partial \hat{y}^{(t-1)}} (y_i, \hat{y}_i)$
        $h_i = \frac{\partial^2 l(y_i, \hat{y}^{(t-1)})}{\partial^2 \hat{y}^{(t-1)}} (y_i, \hat{y}_i)$
    return g, h

function LEAVES(g, h, P, lr, st)
    # g: first partial vector of loss,
    # h: second partial vector of loss
    # P: partition $\{S_1, \ldots, S_T\}$ of $\{1, \ldots, n\}$,
    # lr: learning rate
    # st: subset threshold

    for $j \in \{1, \ldots, T\}$ **do**
        $G_j = 0$, $H_j = 0$
        if $score\,(S_j) \geq st$ **then**
            for $i \in S_j$ **do**
                $G_j = G_j + g_i$
                $H_j = H_j + h_i$
            $l_j = -lr \cdot \frac{G_j}{H_j}$
    return l

function AGGREGATECLASSIFIER(c)
    # c: classifier (weak or composite) to aggregate

    append c to $classifiers$ vector **return** $classifiers$

function CLASSIFYMULTI(c, d)
    # c: composite classifier,
    # d: dataset

    $p = 0$
    for $cls \in c$ **do**
        $p = p + c.classify\,(d)$
    return p

Algorithm 3 Multiscale Gradient Boosting Fit (cont.)

function FITMULTI($d, l, p, Tvec, i, lr, st, loss$)
    # d: dataset, l: labels, p: current prediction
    # Tvec: vector of partition sizes, i: current index into Tvec
    # lr: learning rate
    # st: subset threshold below which subsets are not considered
    # loss: loss function specification

    # Assume the weak classifier has the methods .fit() .classifiy()
    $g, h = COMPUTEGH\,(l, p, loss)$
    $Scores, P = DISCOPTSOLUTION\,(g, h, Tvec[i])$
    $leaves = LEAVES\,(g, h, P, lr, sr)$
    $csf = fitweak\,(d, leaves)$
    $c = aggregateclassifiers\,(csf)$
    $i = i + 1$https://www.overleaf.com/project/61320f78be964b35c19be04c
    if $i < size\,(Tvec)$ **then**
        $p = CLASSIFYMULTI\,(d, c)$
        $csf = fitmulti\,(d, leaves, p, Tvec, i, lr, sr)$
    else
        $csf = fitweak\,(d, leaves)$
        $c = aggregateclassifiers\,(csf)$
    return c

1.3 Additional features

The definition of resolution of the iterative step and the use of exact solvers allow us to propose the iterative scheme above. The approach is also enhanced by

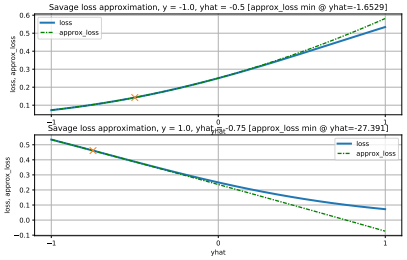
- Loss engineering via synthetic loss functions;
- Priority selection of subsets using the multiple cluster objective of the consecutive partition solver.

1.3.1 Synthetic loss functions

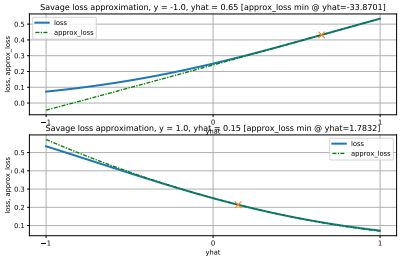
A common choice for loss function l for classification problems, and the default choice in many gradient boost-based software libraries is the cross entropy loss, defined by

$$l\,(y, \hat{y}) = -\left(y * \log\ (\hat{y}) - (1 - y) \log\,(1 - \hat{y})\right), \tag{7}$$

for $y \in \{0, 1\}$, $\hat{y} \in (0, 1)$. A number of alternatives to cross-entropy loss are discussed below. What choice of loss function $\mathcal{L}$ is best for a given classification problem? The problem is beyond the scope of this discussion (e.g. [3], [16]). Instead we propose a method for defining new 'synthetic' loss surrogates based on specification of a related update function and obtain a one-paramater family of loss functions. First assume a normalization, that the values y_i lie in $\{\pm 1\}$. Outside of typical assumptions ($l\,(y, y) = 0$, $l \geq 0$, etc) the only requirement assumed (see [7]) is that the l be C^2 smooth. The gradient boost iteration steps attempts to solve the program in (5) by recasting it by approximating the loss function with a quadratic, which can easily be globally minimized. The approximation chosen, however, is a Taylor series, valid locally at $y_i = \hat{y}^{(t-1)}$, while the global minimum may not have a direct relationship to the global minimum of $l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i))$. It is true that for some poorly-chosen loss functions the approximation can go astray.



(a) Savage loss, $(y, \hat{y}) = (-1, -0.5), (1, -0.75)$



(b) Savage loss, $(y, \hat{y}) = (-1, 0.65), (1, 0.15)$

Figure 1: Savage loss function with approximation. Note that for $(y, \hat{y}) = (1, -0.75)$ the approximation curve is concave, sending the update to $\hat{y}$ closer to $\hat{y} = -1$ rather than the true $y = 1$. The analagous situation occurs with $(y, \hat{y}) = (-1, 0.65)$.

Fix an index i. It should be the case that moving our estimate $\hat{y}_i$ in the direction of the minimum of the approximation to l gets gets us closer to the minimum of the true loss function. Assume for simplicity that the regularization term Ω is zero. The i^{th} summand of the expression

$$\mathcal{L}^{(t)} \approx \sum_{i=1}^{n} [\frac{\partial l(y_i, \hat{y}^{(t-1)})}{\partial \hat{y}^{(t-1)}}) f_t(x_j) + \frac{1}{2} \frac{\partial^2 l(y_i, \hat{y}^{(t-1)})}{\partial^2 \hat{y}^{(t-1)}} f_t(x_j)^2]$$

is minimized for the value $\Phi_i = -\frac{\frac{\partial l(y_i, \hat{y}_i)}{\partial \hat{y}}}{\frac{\partial^2 l(y_i, \hat{y}_i)}{\partial \hat{y}^2}}$, and the value $(y_i, \hat{y}_i + \Phi_i)$ is required to be closer to the minimizer of l than $(y_i, \hat{y}_i)$. So we require that Φ_i have the same sign as $y_i - \hat{y}_i$, or $\Phi(y, \hat{y})(y - \hat{y}) \geq 0$. By (6) it is sufficient to check this condition on the set $\{y\hat{y} \leq 1\}$, The form of Φ for some of the most common classifier loss functions is shown in the following table (we present the cross entropy loss in the form for normalized $y \in \{0, 1\}$ as is standard; all others assume $y \in \{\pm 1\}$). The synthetic p-loss function will be introduced in a later section.

Name	**Loss Specification**	$\Phi(y, \hat{y})$
Square Loss	$(1 - y \cdot \hat{y})^2$	$\frac{1}{y} - \hat{y}$
Exp Loss	$\exp(-y \cdot \hat{y})$	$\frac{1}{y}$
Logistic Loss	$\log(1 + \exp(-y \cdot \hat{y}))$	$\frac{\exp(-y \cdot \hat{y})}{y} + \frac{1}{y}$
Cross Entropy Loss	$-(y \log(\hat{y}) - (1 - y) \log(1 - \hat{y}))$	$\frac{y - \hat{y}}{\hat{y}(1 - \hat{y})}$
Arctan Loss	$(2 \arctan(y \cdot \hat{y}) - 1)^2$	$-\frac{(y^2\hat{y}^2 + 1)(2 \arctan(y\hat{y}) - 1)}{2y(-y\hat{y} + 2y\hat{y} \arctan(y\hat{y}) - 1)}$
Savage Loss	$\frac{1}{(1 + \exp(y \cdot \hat{y}))^2}$	$\frac{1 + \exp(y \cdot \hat{y})}{y(2 \exp(y \cdot \hat{y}) - 1)}$
Synthetic p-Loss	$\begin{cases} \frac{1}{2}(1 - y\hat{y})^2, & p = 1; \\ \int_y^{\hat{y}} -ye^{\frac{1}{-p-1}[(1-yw)^{-p-1} - 1]} dw, & p \neq 1 \end{cases}$	$y(1 - y\hat{y})^p$

For a loss function strictly convex in $\hat{y}$, Φ has the same sign as $-\frac{\partial l(y, \hat{y})}{\partial \hat{y}}$. Furthermore, since l is nondecreasing in $\hat{y}$ for $\hat{y} \geq y$ and nonincreasing for $\hat{y} \leq y$, the condition $\Phi(y, \hat{y})(y - \hat{y}) \geq 0$ is clearly satisfied. For nonconvex loss functions this condition may not hold and the direction indicated by $\Phi(y, \hat{y})$ may actually be incorrect. The Savage, arctan loss functions do not satisfy the convexity requirement and should not be considered for use in the 2nd order gradient boost algorithms (although first order methods have been developed [15]), see, e.g., Figure (17).

The value $\Phi(y,\hat{y})$ also conveys a magnitude which may or may not be related to the distance $y-\hat{y}$. Each solution of the maximization program in (5) and update to the cumulative classifier will make the adjustment

$$\hat{y}_{i+1} = \hat{y}_i + \frac{\sum_{k\in S} \frac{\partial l(y_k,\hat{y}_k)}{\partial \hat{y}}}{\sum_{k\in S} \frac{\partial^2 l(y_k,\hat{y}_k)}{\partial \hat{y}^2}}$$

where $S \subseteq \mathcal{V}$ is the subset containing i, assuming for simplicity a learning rate parameter (lr in algorithm (2)) of 1 across all iterations. For $S=\{i\}$, $\hat{y}_{i+1} = \hat{y}_i + \Phi(y_i,\hat{y}_i)$. For this reason we call Φ the *update function* for l. The update function is related to the *loss margin* for l, in fact the loss margin for l is equal to $\Phi(0)$ [22]. We have found occasionally that out-of-sample classification is more accurate when the magnitude of Φ is small for those cases $(y_i,\hat{y}_i)$ for which $y\hat{y} \geq 1$. The reason for this may be that whenever $\Phi(y_i,\hat{y}_i) \neq 0$, there is a nontrivial contribution to the summand in (5). The index i for an already correct prediction $\hat{y}_i$ for which $y_i\hat{y}_i \geq 0$ will be associated with a nonzero leaf value, i will be a member of some leaf level set, and the classifier fitting will expend some effort increasing the magnitude of $|y_i\hat{y}_i|$, possibly at the expense of the quality of fit on the as of yet misclassified regions. The convex loss functions listed above, besides satisfying the condition $\Phi(y,\hat{y})(y-\hat{y}) \geq 0$, display specific behavior near $y\hat{y}=1$, yielding in sample fitting overshoot (exponential loss, logistic loss to a lesser degree), or a kind of reflection back to the target y in the case of square loss. It might be preferable to choose a loss function that has a flat profile near $y\hat{y}=1$, acting as a sink when $\hat{y}$ is correctly classified.

In the gradient boost algorithm the explicit form of the loss function l is not used, it is only seen through the quantities $g = \frac{\partial l}{\partial \hat{y}}(y,\hat{y})$ and $h = \frac{\partial^2 l}{\partial \hat{y}^s}(y,\hat{y})$. There is a natural progression

$$l \rightarrow (g,h) \rightarrow \Phi$$

in the sense that the calculations in the direction of the arrows can easily be performed for any twice-differentiable l. It would be advantageous in some cases not to specify l, as the important aspect is the quality of the global properties of a local quadratic approximation at each point, but rather Φ, as the loss engineering is more intuitive at that level. Having specified Φ, is it possible to extract g, h in a canonical way? Does the specification come from a convex loss function? To this end, one can specify the behavior of $\Phi(-1,\hat{y}) = a(\hat{y})$, $\Phi(1,\hat{y}) = b(\hat{y})$ and write

$$\Phi \frac{\partial^2 l}{\partial \hat{y}^s}(y,\hat{y}) + \frac{\partial l}{\partial \hat{y}}(y,\hat{y}) = 0, \quad (y,\hat{y}) \in \{\pm 1\} \times [0,1]$$
$$l(y,\hat{y}) = 0$$

for $\Phi(-1,\hat{y}) = a(\hat{y}), \Phi(1,\hat{y}) = b(\hat{y})$. Noting that $-\frac{1}{\Phi} = \frac{\partial}{\partial \hat{y}} \log \frac{\partial l}{\partial \hat{y}}(y,\hat{y})$ we can easily obtain

$$l(y,\hat{y}) = \int_y^{\hat{y}} l_{\hat{y}}(y,0)\, e^{-\int_0^w \frac{dz}{\Phi(y,z)}} dw. \tag{8}$$

Differentiating both sides with respect to $\hat{y}$ gives

$$\frac{l_{\hat{y}}(y,\hat{y})}{l_{\hat{y}}(y,0)} = \exp^{-\int_0^{\hat{y}} \frac{dz}{\Phi(y,z)}} =: \Lambda(y,\hat{y}) \tag{9}$$

To solve for $l_{\hat{y}}$ it is necessary that $\Lambda(y,0) \equiv 1$, in which case Λ itself is a solution

$$\frac{\Lambda(y,\hat{y})}{\Lambda(y,0)} = \Lambda(y,\hat{y}). \tag{10}$$

If $\frac{l_1(y,\hat{y})}{l_1(y,0)} = \frac{l_2(y,\hat{y})}{l_2(y,0)} = \Lambda(y,\hat{y})$, then clearly $l_1(y,\hat{y}) = q(y)\, l_2(y,\hat{y})$ for some function q of y. So necessarily $l_{\hat{y}}(y,\hat{y}) = q(y)\Lambda(y,\hat{y})$. We also require convexity in $\hat{y}$, or $\frac{\partial}{\partial \hat{y}} l_{\hat{y}}(y,\hat{y}) \geq 0$, for $y \in \{\pm 1\}$.

The choice $q(y) = -y$ will guarantee that $q(y)\frac{\partial}{\partial \hat{y}}\Lambda(y,\hat{y}) \geq 0$, for $y \in \{\pm 1\}$, $\hat{y} \in [-1,1]$. We have some freedom to chose q as the original partial differential equation is underdetermined; the behavior of $\Phi(\cdot)$ is only constrained by $l(y,\hat{y}) = 0$. The partial derivatives can now be written as $\frac{\partial}{\partial \hat{y}} l(y,\hat{y}) = -y\frac{\partial}{\partial \hat{y}}\Lambda(y,\hat{y})$, $\frac{\partial^2}{\partial \hat{y}^2} l(y,\hat{y}) = \frac{y}{\Phi(y,\hat{y})}\frac{\partial^2}{\partial \hat{y}^2}\Lambda(y,\hat{y})$. For example, with the knowledge that $l(y,\hat{y}) = l_{sq}(y,\hat{y}) = (1-y\hat{y})^2$, we obtain $l_{\hat{y}}(y,\hat{y}) = -2y(1-y\hat{y})$ and $\Phi(y,\hat{y}) = \frac{1}{y} - \hat{y}$. Without prior knowledge of l, starting with Φ gives $\Lambda(y,\hat{y}) = (1-y\hat{y})$. Therefore $l_{\hat{y}}(y,\hat{y}) = -y(y)\Lambda(y,\hat{y}) = -y(1-y\hat{y})$, which matches the true partial up to a constant.

In this way we can create *synthetic* loss functions by engineering Φ to suit the problem at hand. It is often easier to reason about properties of the update function rather than directly about the loss function. For example, for the square loss function above, $\Phi(-1,\hat{y}) = -1-\hat{y}$, and $\Phi(1,\hat{y}) = 1-\hat{y}$. Note the reflective property around $y\hat{y} = 1$; the contribution of Φ will be to direct $\hat{y}$ back to ± 1 along linear increments.

For $p \in \mathbf{R}$ we can specify $\Phi(y,\hat{y}) = y(1-y\hat{y})^p$, obtaining a parameterized family of loss functions.

Definition 5. *For $p \in \mathbf{R}$ define the* synthetic p-loss function *by*

$$l_p(x,y) = \left\{ \begin{array}{ll} \frac{1}{2}(1-y\hat{y})^2, & p = 1; \\ \int_y^{\hat{y}} -ye^{\frac{1}{-p-1}[(1-yw)^{-p-1}-1]}dw, & p \neq 1 \end{array} \right\}$$

Denote by $\Phi_p(y,\hat{y}) = y(1-y\hat{y})^p$ the associated update function.

This family generalizes some well-known losses: l_0 is the exponential loss, while l_1 is the square or quadratic loss. It is similar to the *comp-sum* loss functions introduced in [14], although the derivation differs. In our case desired behavior is expressed in the form of the update function Φ rather than through the loss function. The qualifier 'synthetic' will sometimes be dropped, so that the above will be simply referred to as a p-loss functions. Note that the power specification p is applied to Φ, not the loss function l, so that the 1-power loss is a quadratic, e.g. The condition $\Phi(y,\hat{y})(y-\hat{y}) = (1-y\hat{y})^{p+1} \geq 0$ is satisfied, for all p. To flatten the behavior of Φ for $y\hat{y}$ near 1, the 3-loss function could be used. Further engineering can be done by modifying Φ_3. For example, to suppress the behavior in the regions $\hat{y} < -1$ or $\hat{y} > 1$ characterized by overshoot, increasing the magnitude of a correc classification, we could define

$$\Phi(y,\hat{y}) = \left\{ \begin{array}{ll} \min\left(y(1-y\hat{y})^3, 0\right), & y = -1; \\ \max\left(y(1-y\hat{y})^3, 0\right), & y = 1 \end{array} \right.$$

thereby clamping the behavior for $|y\hat{y}|$ large. In this case we do not need to derive $l_{\hat{y}}$, $l_{\hat{y}\hat{y}}$ by the same process above, we may use the previous $l_{\hat{y}\hat{y}}$ and clamp the behavior of $l_{\hat{y}}$ directly

$$l_{\hat{y}}(y,\hat{y}) = \left\{ \begin{array}{ll} \min\left(-ye^{-\frac{1}{4}[(1-y\hat{y})^{-4}-1]}, 0\right), & y = -1; \\ \max\left(-ye^{-\frac{1}{4}[(1-y\hat{y})^{-4}-1]}, 0\right), & y = 1 \end{array} \right.$$

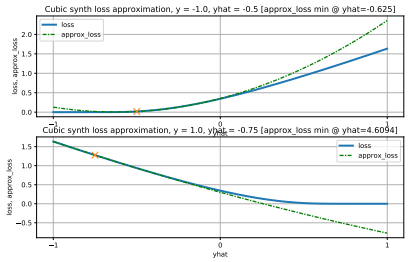
as the second order information is not needed where $l_{\hat{y}} = 0$.

1.3.2 Prioritization via clustering objective

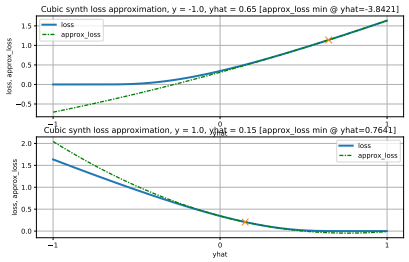
The use of an exact solver to specify the leaf values and subsets at each iteration step makes use of the Consecutive Partition Solver framework in [19]. The methods developed there generalize the solution of the program

$$\mathcal{P}^* = \underset{\mathcal{P}=\{S_1,\ldots,S_T\}}{\operatorname{argmax}} \sum_{j=1}^{T} F(S_j)$$

from $T = 1$ to $T \geq 2$, where F is a discrete scoring function for a continuous function f. The value $F(S_j)$ is defined as the *score* of S_j. The $T = 1$ solution can be interpreted as solving two different

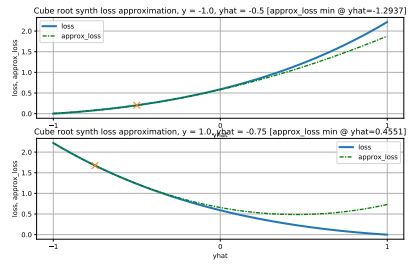


(a) synthetic 3-loss, $(y, \hat{y}) = (-1, -0.5), (1, 0.75)$

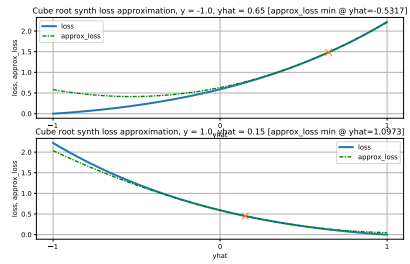


(b) synthetic 3-loss, $(y, \hat{y}) = (-1, -0.5), (1, -0.75)$

Figure 2: Synthetic p-loss, $p = 3$, with approximation.



(a) synthetic $\frac{1}{3}$-loss, $(y, \hat{y}) = (-1, -0.5), (1, 0.75)$



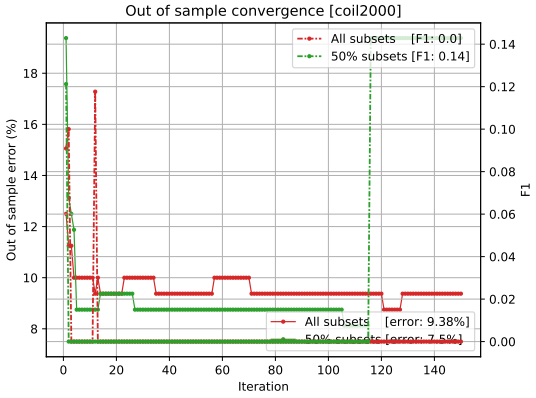
(b) synthetic $\frac{1}{3}$-loss, $(y, \hat{y}) = (-1, -0.5), (1, -0.75)$

Figure 3: Synthetic p-loss, $p = \frac{1}{3}$, with approximation.

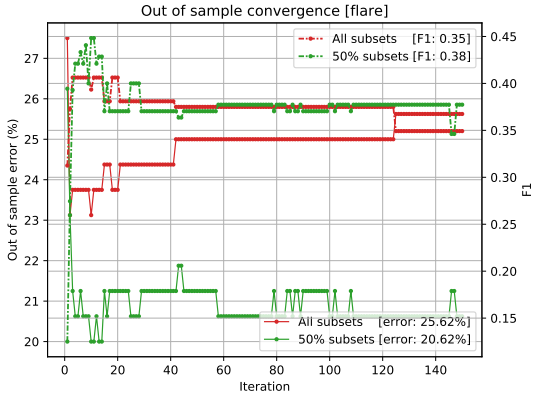
problems, 1, partitioning $\mathcal{V}$ optimally into two subsets of maximally different scores, S_1 and $\mathcal{V} \setminus S_1$, or 2, finding the single subset S_1 of most elevated risk. The two different interpretations generalize to different approaches for $T \geq 2$, the *risk partitioning* and *multiple cluster* approaches, respectively. We will use the first approach to solve equation (5). After obtaining the optimal $\mathcal{P}^*$, the subsets S_j are ranked by score, low-scoring sets are discarded, according to a threshold criteria. This allows the classifier to focus on misclassified points associated with high-scoring subsets. From our observations, applying a simple threshold-based cutoff in this manner can almost always improve out of sample fits, sometimes dramatically.

To demonstrate, we show out of sample convergence (by error and F1 score) for several datasets, on a single non-recursive depth-one V-cycle. The selection of hyperparameters is not optimized and is intended to show the effect of subset refinement in isolation. The parameter *priority selection* determins the proportion of subsets, ranked by score at each iteration step, are used to specify leaf value and level sets for the classifier fit stage. For out of sample fits, on the datasets we've studied, the prioritization of subsets almost always improves out of sample fits. The datasets are all of size $n = 1000$, we chose $T = 500$, row subsampling of 75%, loss function is the square loss. For each dataset, four plots are shown, error and F1 score for the classifier using a) all subsets at every iteration, and 50% of all subsets. Each fit was performed over 100 iterations. The results for the datasets[2] *coil2000*, *flare*, *hypothyroid*, *income*, *spambase*, and *phoneme*, and are shown in figures (4, 5, 6).

[2] All of the datasets referred to in this article come from the Penn Machine Learning Benchmarks repository, available here or through OpenML

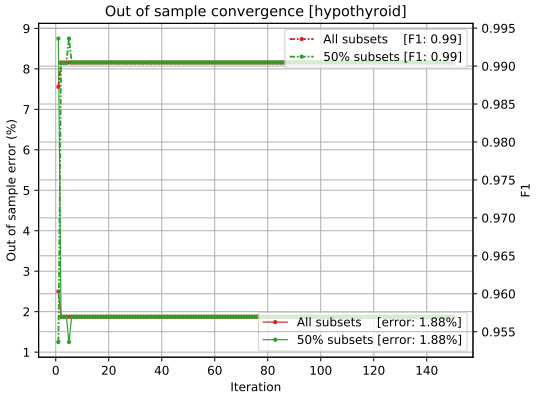


(a) Out of sample error, F1, no priority selection vs 50% priority selection, coil2000 data.

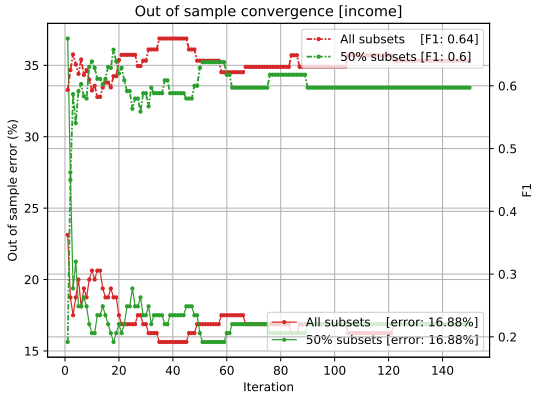


(b) Out of sample error, F1, no priority selection vs 50% priority selection, flare data.

Figure 4: Out of sample fits priority selection ratios 100%, 50%



(a) Out of sample error, F1, no priority selection vs 50% priority selection, hypothyroid data.



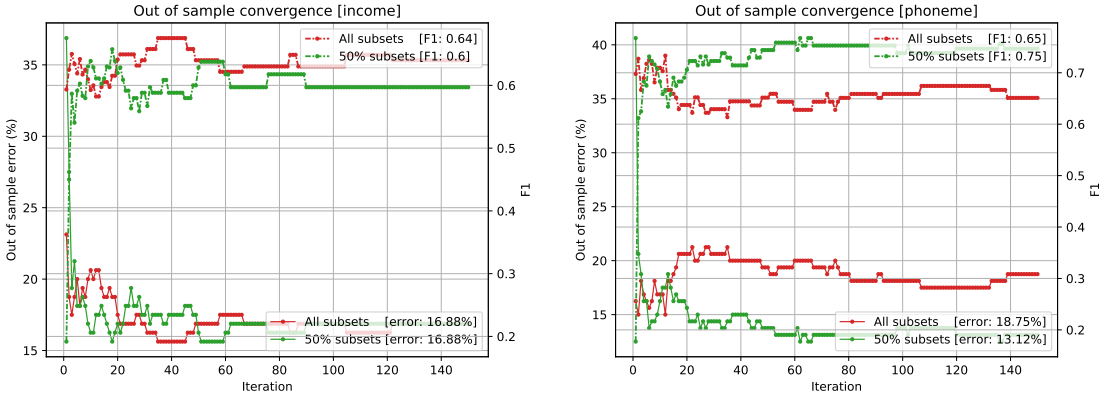
(b) Out of sample error, F1, no priority selection vs 50% priority selection, income data.

Figure 5: Out of sample fits priority selection ratios 100%, 50%

1.4 Empirical results

For the datasets *buggyCrx*, *house votes 84*, the following test TESTSUITE of the effectiveness of multiboost parameters is performed. The quality of fit of multiboost model fits is performed ranging over a set of p-loss functions and size $\{1, 3, 5, 7, 9\}$-level recursive partition size specifications. For each such specification we fit twice, first retaining all subsets at each iterative step, second retaining only the top 50% of subsets by score at each step ($st = 0.00, 0.50$, respecitively). The fits are displayed by loss function p-value, all else fixed, with a quadratic fit to the out of sample error accross p. The results indicate

- Fits are noisy across p-loss specification;
- Deeper recursion specification improves quality of fits;
- Priority selection at 50% improves quality of fits;



(a) Out of sample error, F1, no priority selection vs 50% priority selection, income data.

(b) Out of sample error, F1, no priority selection vs 50% priority selection, phoneme data.

Figure 6: Out of sample fits priority selection ratios 100%, 50%

This is consistent with our findings across a larger set of datasets. Hyperparameter selection will be the topic of future work.

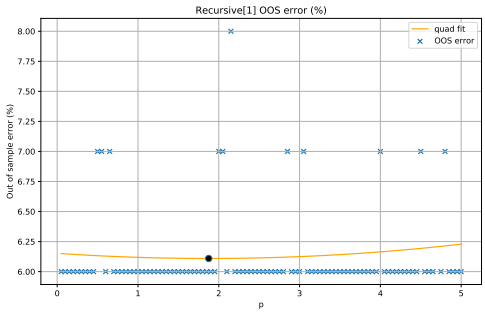
Algorithm 4 Test Suite

function TESTSUITE(dataset, labels, PartitionSizes)
    # Test of effectiveness of subset of multiscale boosting parameters
    #
    # dataset, labels
    # PartitionSize = $\{PartitionSizes_1, \ldots, PartitionSizes_9\}$: a set of partition sizes

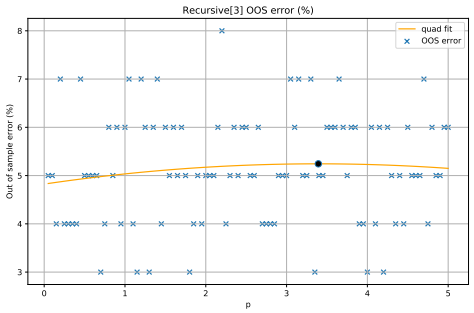
    # Recall Signature of FITMULTI
    # FITMULTI(d, l, p, Tvec, i, lr, st, loss)
    # d: dataset, l: labels, p: current prediction
    # Tvec: vector of partition sizes, i: current index into Tvec
    # lr: learning rate
    # st: subset threshold below which subsets are not considered
    # loss: loss function specification

    # Model fits
    # Learning rate
    $lr = 0.01$

    for $p \in [0.00, \ldots, 5.00]$ step .05 **do**
        $loss = p\text{-}loss\ function$
        for $Tvec \in \{PartitionSizes_1, \ldots, PartitionSizes_9\}$ **do**
            for $st \in \{0.00, 0.50\}$ **do**
                # Initial prediction
                $pred = 0$
                error = FITMULTI $(dataset, labels, pred, Tvec, 0, lr, st, loss)$
    return *errors*

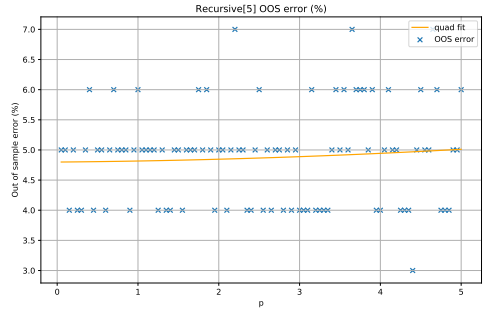


(a) Out of sample error, 1-level recursion

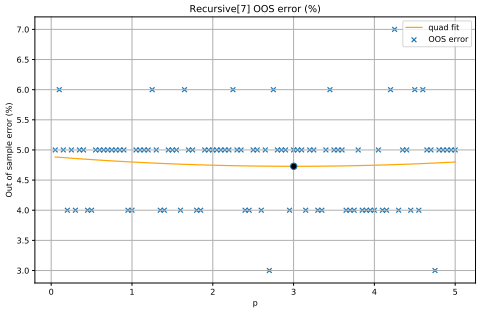


(b) Out of sample error, 3-level recursion

Figure 7: Out of sample error for house votes 84 data, 1-, 3-level recursion.



(a) Out of sample error, 5-level recursion



(b) Out of sample error, 7-level recursion

Figure 8: Out of sample error for house votes 84 data, 5-, 7-level recursion.

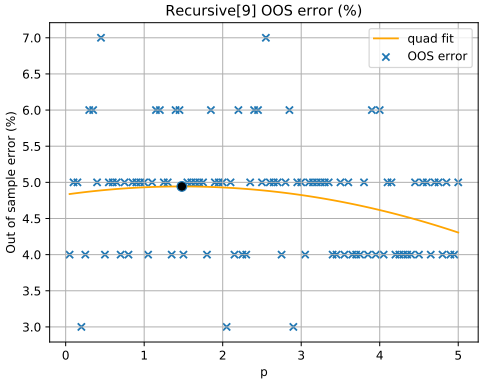
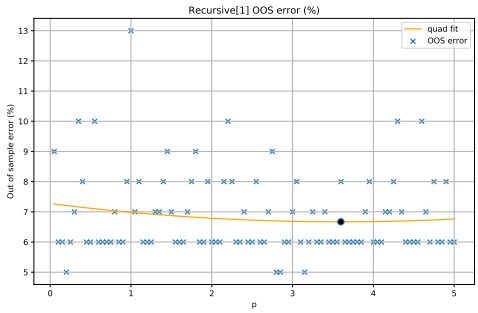
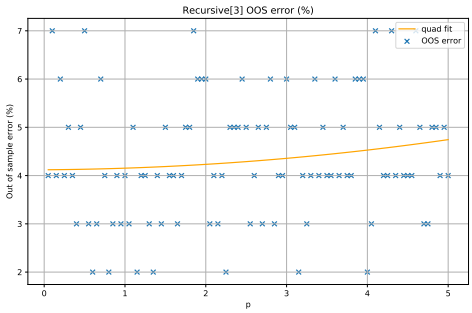


Figure 9: Out of sample error for house votes 84 data, 9-level recursion.

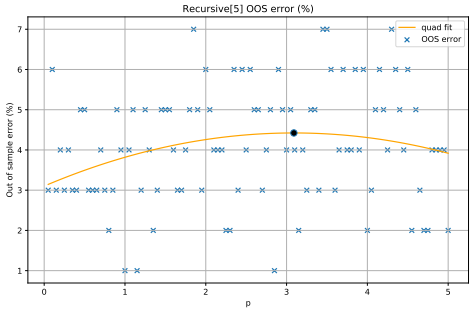


(a) Out of sample error, 1-level recursion

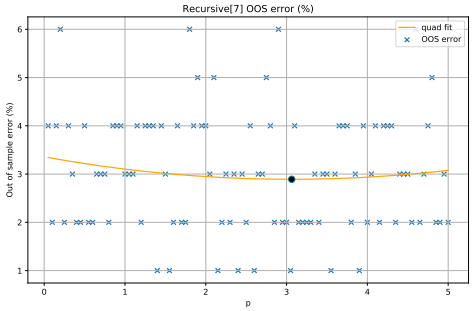


(b) Out of sample error, 3-level recursion

Figure 10: Out of sample error for house votes 84 data, 1-, 3-level recursion, 50% subset retention.



(a) Out of sample error, 5-level recursion



(b) Out of sample error, 7-level recursion

Figure 11: Out of sample error for house votes 84 data, 5-, 7-level recursion, 50% subset retention.

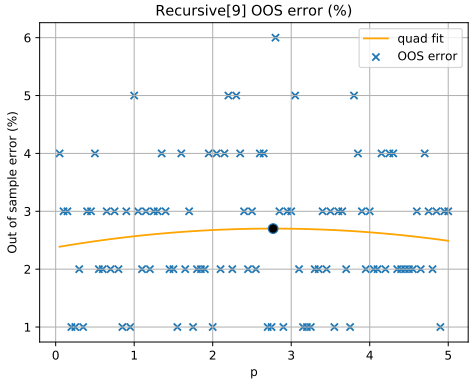
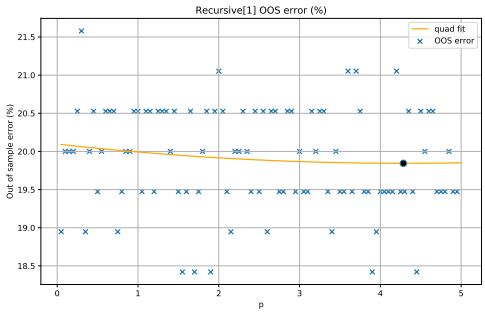
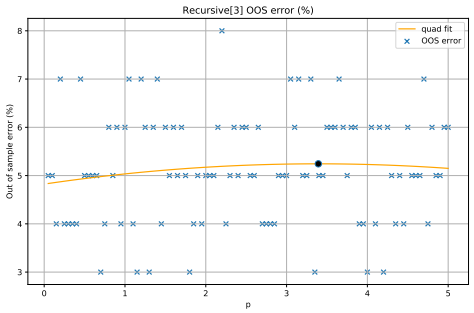


Figure 12: Out of sample error for house votes 84 data, 9-level recursion, 50% subset retentiion.

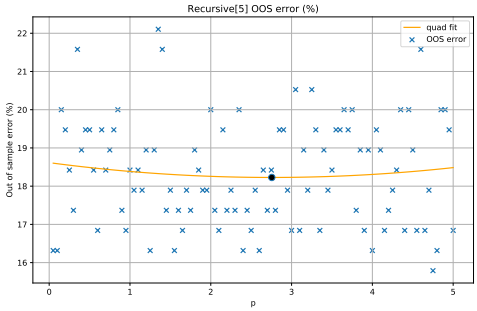


(a) Out of sample error, 1-level recursion

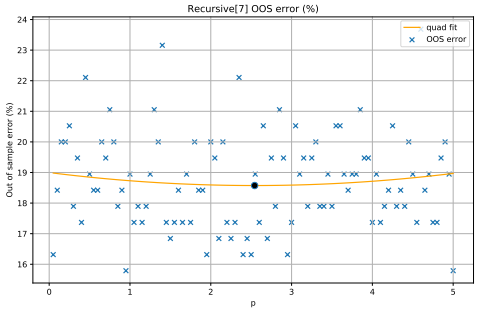


(b) Out of sample error, 3-level recursion

Figure 13: Out of sample error for buggyCrx data, 1-, 3-level recursion.



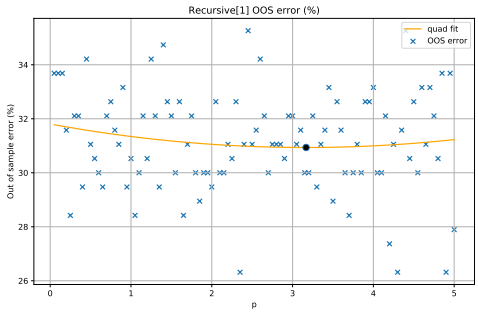
(a) Out of sample error, 5-level recursion



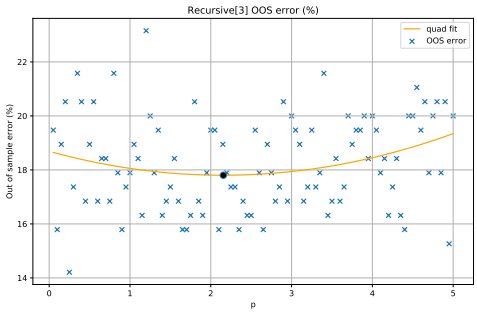
(b) Out of sample error, 7-level recursion

Figure 14: Out of sample error for buggyCrx data, 5-, 7-level recursion.

Figure 15: Out of sample error for buggyCrx data, 9-level recursion.

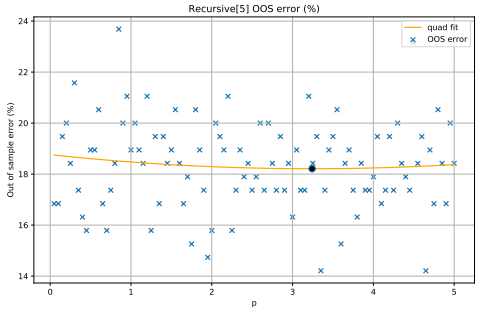


(a) Out of sample error, 1-level recursion

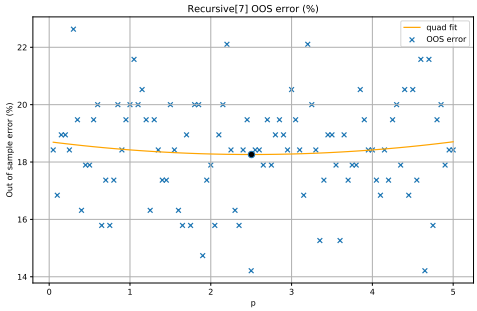


(b) Out of sample error, 3-level recursion

Figure 16: Out of sample error for buggyCrx data, 1-, 3-level recursion, 50% subset retention.



(a) Out of sample error, 5-level recursion



(b) Out of sample error, 7-level recursion

Figure 17: Out of sample error for buggyCrx data, 5-, 7-level recursion, 50% subset retention.

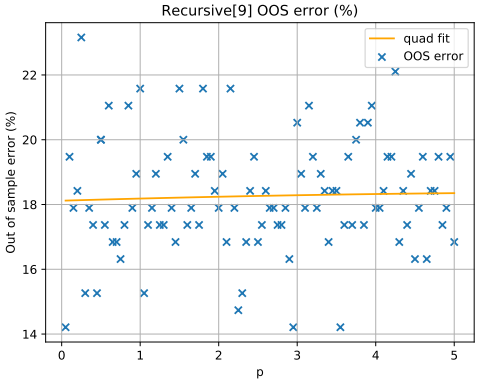


Figure 18: Out of sample error for house votes 84 data, 9-level recursion, 50% subset retention.

References

[1] Ali Abolhassani, Marcos O. Prates, , *An up-to-date review of scan statistics*, Statistics Surveys, Vol. 15 pp 111–153, 2021.

[2] Francis Bach, *Learning with Submodular Functions: A Convex Optimization Perspective*, Carnegie Mellon University, Pittsburgh, USA, 2011

[3] Peter L. Bartlett, Michael I. Jordan, Jon D. McAuliffe, *Convexity, Classification, and Risk Bounds*, Journal of the American Statistical Association, Vol 101, No 473, pp 138-156, Mar 2006.

[4] Yoshua Bengio, *How Auto-Encoders Could Provide Credit Assignment in Deep Networks via Target Propagation*, arXiv:1407.7906v3, 18 Sep 2014

[5] William L. Briggs, Van Emden Henson, Steve F. McCormick, *A Multigrid Tutorial, 2nd Edition*, Society for Industrial and Applied Mathematics, 978-0898714623, 2000

[6] J. Feng, Y. Yu, and Z.-H. Zhou, "Multi-layered gradient boosting decision trees," in Advances in Neural Information Processing Systems. Curran

[7] Tianqi Chen and Carlos Guestrin, *XGBoost*, Proceedings of the 22nd ACM SIGKDD International Confernce of Knowledge Discovery and Data Mining, August, 2016.

[8] Tianqi Chen, Sameer Singh, Ben Taskar, Carlos Guestrin, *Efficient Second-Order Gradient Boosting for Conditional Random Fields*, Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics, PMLR 38:147-155, 2015.

[9] J. Friedman, *Stochastic gradient boosting*, Computational statistics & data analysis, vol. 38, no. 4, pp. 367-378, 2002 Associates, Inc., 2018, pp. 3551–3561.

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

[11] Leo Grinsztajn, Edouard OYallon, Gael Varoquaux, *Why do tree-based models still outperform deep learning on tabular data?*, 2207.08815v1, 18 Jul 2022.

[12] Martin Kulldorf, *Spatial Scan Statistics: Models, Calculations, and Applications*, Statistics for Industry and Technology, Birkhauser, pp 303-322, 1999.

[13] LightGBM documentation, https://lightgbm.readthedocs.io/en/latest/

[14] Anqi Mao, Mehryar Mohri, Yutao Zhong, *Cross-Entropy Loss Functions: Theoretical Analysis and Applications*, ICML'23: Proceedings of the 40th International Conference on Machine Learning, Articl No:992, pp 23803-23828, July 2023.

[15] Hamed Masnadi-Shirazi and Nuno Vasconcelos, *On the Design of Loss Functions for Classification: theory, robustness to outliers, and SavageBoost*, Proceedings of Neural Information Processing Systems (NIPS), Vancouver, Canada, Dec 2008.

[16] Hamed Masnadi-Shirazi, Nuno Vasconcelos, *On the Design of Loss Functions for Classification: theory, robustness to outliers, and SavageBoost*, Advances in neural information processing systems, pages 1049–1056, 2009.

[17] Mason, L.; Baxter, J.; Bartlett, P. L.; Frean, Marcus (1999). *Boosting Algorithms as Gradient Descent*, S.A. Solla and T.K. Leen and K. Müller (ed.). Advances in Neural Information Processing Systems 12. MIT Press. pp. 512–518.

[18] Dong-Hyun Lee, Saizheng Zhang, Asja Fischer, Yoshua Bengio, *Difference Target Propagation*, Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Lecture Notes in Computer Science Vol 9284, Springer, pp 498-515, 2015.

[19] Pehlivanian, Charles A. and Neill, Daniel B., *Efficient optimization of partition scan statistics via the Consecutive Partitions*, Property. Journal of Computational and Graphical Statistics 32(2): 712-729, 2023

[20] Liudmila Prokhorenkova1, Gleb Gusev1, Aleksandr Vorobev1, Anna Veronika Dorogush1, Andrey Gulin, *CatBoost: unbiased boosting with categorical features*, arXiv:1706.09516v5, 20 Jan 2019

[21] Yair Shapira, *Algebraic Multigrid*, Matrix-based multigrid: theory and applications, Springer, p. 66, ISBN 978-1-4020-7485-1, 2003

[22] Vasconcelos, Nuno; Masnadi-Shirazi, Hamed, *A View of Margin Losses as Regularizers of Probability Estimates*, Journal of Machine Learning Research. 16 (85): 2751–2795. ISSN 1533-7928, 2015.