Unified Robust Boosting

Zhu Wang
UT Health San Antonio

Abstract

Boosting is a popular algorithm in supervised machine learning with wide applications in regression and classification problems. Boosting can combine a sequence of regression trees to obtain accurate prediction. In the presence of outliers, traditional boosting may show inferior results since the algorithm optimizes a convex loss function. Recent literature has proposed boosting algorithms to optimizing robust nonconvex loss functions. However, there is a lack of weighted estimation to indicate the outlier status of the observations. This article proposes an iteratively reweighted boosting algorithm combining robust loss optimization and weighted estimation, which is conveniently constructed with existing software. The output includes the weights as a valuable diagnostic to the outlier status of the observations. For practitioners interested in the boosting algorithm, the new algorithm can be interpreted as a method to tuning in observation weights, which can lead to a more accurate model. Applications with publicly available data are demonstrated with the R package **irboost** in various robust boosting approaches to generalized linear models, classification and survival data analysis.

Keywords: Machine learning, boosting, robust method, CC-family, IRCO, IRBoost.

1. Introduction

Boosting is a powerful supervised machine learning algorithm. As an ensemble method, boosting combines many weak learners to generate a strong prediction. As a functional decent method, boosting has a wide applications in regression and classification problems. Friedman (2001); Friedman, Hastie, and Tibshirani (2000) discussed boosting for a variety of convex loss functions. Boosting can be utilized to fit a variety of models with different base learners, including linear least squares, smoothing splines and regression trees (Bühlmann and Hothorn 2007; Wang 2018b). To deal with outliers, robust estimation and boosting can jointly provide more accurate estimation. Wang (2018a,b) proposed robust functional gradient boosting for nonconvex loss functions. These methods applied majorization-minimization (MM) scheme, an extension of the popular expectation-maximization (EM) algorithm in statistics. However, there is a lack of the weights as an indication of outlier status of observations, where small weights are assigned to observations deviated from the underlying model. In the classical robust estimation, the weights are derived from some robust loss functions, such as the Huber loss.

There is some recent progress on how to generate weights from robust loss functions in more complex problems. Wang (2020) innovatively proposed a new framework of robust estimation by reducing the weight of the observation that leads to a large loss. The author initiated a unified class of robust loss functions, the concave convex (CC) family, and introduced the

iteratively reweighted convex optimization (IRCO) that minimizes the loss functions in the CC-family. The CC-family includes traditional robust loss functions such as the Huber loss, robust hinge loss for support vector machine, and robust exponential family for generalized linear models. The IRCO algorithm can be conveniently implemented with existing methods and software.

In this article, we integrate the IRCO and boosting into the IRBoost algorithm for the CC-family. This functional optimization is more general than the parameter-based estimation in Wang (2020). For instance, the IRBoost algorithm permits function space derived from the regression trees. Unlike the previous boosting applications including Wang (2018a,b), the major novelty is that the IRBoost framework provides weights to help identify outliers. We illustrate the proposed algorithm through the R **irboost** package with applications to robust exponential family, including regression, logistic regression and Poisson regression. Another illustration is robust survival regression with accelerated failure time model. The package also implements IRBoost to Gamma regression, Tweedie regression, hinge classification and multinomial logistic regression.

2. Robust boosting

2.1. CC-family function estimation

To unify robust estimation, Wang (2020) proposed the concave convex family with functions Γ satisfying the following conditions:

i. $\Gamma = g \circ s$

ii. g is a nonnegative, nondecreasing closed concave function whose domain is the range of function s

iii. $\partial(-g(z))$ $\forall z \in$ range of s is nonempty and bounded

iv. s is convex on $\mathbb{R}$.

Here $\partial(-g(z))$ means subdifferential of function $-g$ at point z, which is equivalent to the derivative $\{-g'(z)\}$ when exists. Examples of concave component are listed in Table 1. Note that the `tcave` is not differentiable everywhere, but subdifferentiable. The parameter σ controls robustness level a model is allowed, and a smaller value leads to more robust estimation. See Wang (2020) for details. The convex component includes common loss functions in regression and classification such as squared loss $s(u) = u^2$ and negative log-likelihood function in the exponential family adopted by the generalized linear models. Other examples include negative log-likelihood function for multinomial logistic regression, Tweedie regression and accelerated failure time model for time-to-event data subject to censoring (Barnwal, Cho, and Hocking 2020). The requirement of $z \geq 0$ on the domain of g may be relaxed for some concave functions g in Table 1 although many commonly used loss functions have a range of nonnegative values. However, $g < 0$ does exist, for instance, when g is a negative log-likelihood value for the Gamma distribution. In this case, a nonnegative value is easily obtained by subtracting some data dependent constant, which is described below.

Concave	$g(z), z \geq 0$
hcave	$\begin{cases} z & \text{if } z \leq \sigma^2/2 \\ \sigma(2z)^{\frac{1}{2}} - \frac{\sigma^2}{2} & \text{if } z > \sigma^2/2 \end{cases}$
acave	$\begin{cases} \sigma^2(1 - \cos(\frac{(2z)^{\frac{1}{2}}}{\sigma})) & \text{if } z \leq \sigma^2\pi^2/2 \\ 2\sigma^2 & \text{if } z > \sigma^2\pi^2/2 \end{cases}$
bcave	$\frac{\sigma^2}{6}\left(1 - (1 - \frac{2z}{\sigma^2})^3 I(z \leq \sigma^2/2)\right)$
ccave	$\sigma^2\left(1 - \exp(\frac{-z}{\sigma^2})\right)$
dcave	$\frac{1}{1-\exp(-\sigma)} \log(\frac{1+z}{1+z\exp(-\sigma)})$
ecave	$\begin{cases} \frac{2\exp(-\frac{\delta}{\sigma})}{\sqrt{\pi\sigma\delta}} z & \text{if } z \leq \delta \\ \text{erf}(\sqrt{\frac{z}{\sigma}}) - \text{erf}(\sqrt{\frac{\delta}{\sigma}}) + \frac{2\exp(-\frac{\delta}{\sigma})}{\sqrt{\pi\sigma\delta}}\delta & \text{if } z > \delta \end{cases}$
gcave	$\begin{cases} \frac{\delta^{\sigma-1}}{(1+\delta)^{\sigma+1}} z & \text{if } z \leq \delta \\ \frac{1}{\sigma}(\frac{z}{1+z})^\sigma - \frac{1}{\sigma}(\frac{\delta}{1+\delta})^\sigma + \frac{\delta^\sigma}{(1+\delta)^{\sigma+1}} & \text{if } z > \delta \end{cases}$ where $\delta = \begin{cases} \to 0+ & \text{if } 0 < \sigma < 1 \\ \frac{\sigma-1}{2} & \text{if } \sigma \geq 1 \end{cases}$
tcave	$\min(\sigma, z), \quad \sigma \geq 0$

Table 1: Concave component with $\sigma > 0$ except for tcave with $\sigma \geq 0$.

Given a set of observations $(\mathbf{x}_i, y_i), i = 1, ..., n$, where $y_i \in \mathbb{R}$ and $\mathbf{x}_i = (x_{i1}, ..., x_{ip})^\intercal \in \mathbb{R}^p$, denote Ω the linear span of a set H of base learners including regression trees and linear predictor functions. A model $\mathbf{f} = (f(\mathbf{x}_1), ..., f(\mathbf{x}_n))^\intercal \in \Omega$ can be obtained by minimizing an empirical loss function

$$\sum_{i=1}^{n} \ell(y_i, f(\mathbf{x}_i)), \tag{1}$$

where ℓ is a member of the CC-family, $\ell = g \circ s = g(s(u))$. With some abuse of notation, $s(u)$ is also used to denote $s(y, f(\mathbf{x}))$. For instance, $s(u) = s(y - f(\mathbf{x}))$ in regression, and $s(u) = s(yf(\mathbf{x}))$ in a margin-based classification with $y \in [-1, 1]$. If $s(y, f(\mathbf{x})) < 0$, a simple remedy is to subtract some constant C such that $s(y, f(\mathbf{x})) - C \geq 0$. For the exponential family, $s(y, f(\mathbf{x}_i)) \geq s(y, y)$ holds since $s(y, y)$ is the negative log-likelihood value of a saturated model. Hence, the desired concave loss can be obtained with $s(y_i, f(x_i)) - \min_{i=1,...,n} s(y_i, y_i) \geq 0$. To simplify notations, f is often used to replace $f(\mathbf{x})$.

The robust function estimation problem (1) can be solved by Algorithm 1, where Step 4 involves φ, the Fenchel conjugate of $-g$. Denote

$$\rho(\mathbf{f}^{(k)}) = \sum_{i=1}^{n} \ell(y_i, f_i^{(k)}), \tag{2}$$

where $\mathbf{f}^{(k)}$ are generated in the algorithm. We have the convergence results for the IRBoost algorithm.

Theorem 1. *Suppose that g is a concave component in the CC-family, and g is bounded below. The loss function values $\rho(\mathbf{f}^{(k)})$ generated by Algorithm 1 are nonincreasing and converge.*

Algorithm 1 IRBoost

1: **Input:** training samples $\{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)\}$, concave component g with parameter σ, convex component s, starting point $\mathbf{f}^{(0)}$ and iteration count K.
2: **for** $k = 1$ to K **do**
3: Compute $z_i = s(y_i, f_i^{(k-1)}), i = 1, ..., n$
4: Compute subgradient $v_i^{(k)}$ via $v_i^{(k)} \in \partial(-g(z_i))$ or $z_i \in \partial\varphi(v_i^{(k)}), i = 1, ..., n$
5: Compute $\mathbf{f}^{(k)} = \text{argmin}_{\mathbf{f}\in\Omega} \sum_{i=1}^n s(y_i, f_i)(-v_i^{(k)})$
6: **end for**
7: **Output:** $v_i^{(K)}$ and $\mathbf{f}^{(K)}$.

This result is a generalization of Theorem 4 in Wang (2020), where the function is the linear predictor. Here we study more broadly defined function spaces. On the other hand, if H is a space of linear models, Theorem 1 indeed coincides with the results in Wang (2020). Algorithm 1 is a majorization-minimization algorithm, and the proof follows the same argument of Theorem 4 in Wang (2020), hence only a sketch of the proof is given below.

For a differentiable concave function g, the first-order condition is $\forall u, v \in \text{dom } g$

$$g(u) \le g(v) + g'(v)(u - v). \tag{3}$$

Substitute u with $s(u)$, and v with $s(v)$ in (3), we get

$$g(s(u)) \le g(s(v)) + g'(s(v))(s(u) - s(v)). \tag{4}$$

Substitute $s(u) = s(y_i, f_i), s(v) = s(y_i, f_i^{(k)})$ in (4), and sum up for $i = 1, ..., n$, we get

$$\sum_{i=1}^n g(s(y_i, f_i)) \le \sum_{i=1}^n g(s(y_i, f_i^{(k)})) + g'(s(y_i, f_i^{(k)}))(s(y_i, f_i) - s(y_i, f_i^{(k)})). \tag{5}$$

Denote $Q(\mathbf{f}|\mathbf{f}^{(k)})$ the right hand side of (5), the following inequalities hold:

$$\rho(\mathbf{f}^{(k+1)}) \le Q(\mathbf{f}^{(k+1)}|\mathbf{f}^{(k)}) \le Q(\mathbf{f}^{(k)}|\mathbf{f}^{(k)}) = \rho(\mathbf{f}^{(k)}). \tag{6}$$

Alternatively, the majorization (6) can be constructed from a different surrogate function derived from the Fenchel convex conjugate:

$$Q(\mathbf{f}|\mathbf{f}^{(k)}) = \sum_{i=1}^n s(y_i, f_i)(-v_i^{(k+1)}) + \varphi(v_i^{(k+1)}).$$

Step 4 computes weights in two different ways corresponding to different surrogate functions. The solutions can be shown to be the same based on the Fenchel-Moreau theorem.

2.2. Boosting algorithm for function estimation

An important question is how to compute step 5 in Algorithm 1. For ease of notation, we only present methods to the following unweighted estimation since the weighted estimation does not pose technical difficulties:

$$\underset{\mathbf{f}\in\Omega}{\text{argmin}} \sum_{i=1}^n s(y_i, f_i). \tag{7}$$

In a boosting algorithm, the solution is an additive model given by

$$\hat{f}_i = F_M(\mathbf{x}_i) = \sum_{i=1}^{M} t_m(\mathbf{x}_i), \tag{8}$$

where $F_M(\mathbf{x}_i)$ is stagewisely constructed by sequentially adding an update $t_m(\mathbf{x}_i)$ to the current estimate $F_{m-1}(\mathbf{x}_i)$:

$$F_m(\mathbf{x}_i) = F_{m-1}(\mathbf{x}_i) + t_m(\mathbf{x}_i), m = 1, ..., M. \tag{9}$$

There are different ways to compute $\mathbf{t}_m(\mathbf{x}) = (t_m(\mathbf{x}_1), ..., t_m(\mathbf{x}_n))^\intercal$: gradient and Newton-type update are the most popular (Sigrist 2020). When the second derivative of loss function exists, the Newton-type update is preferred over gradient update to achieve fast convergence:

$$\mathbf{t}_m(\mathbf{x}) = \underset{\mathbf{f} \in H}{\operatorname{argmin}} \sum_{i=1}^{n} h_{m,i}(-\frac{d_{m,i}}{h_{m,i}} - f(x_i))^2, \tag{10}$$

where the first and second derivatives of the loss function s for observations i are given by:

$$d_{m,i} = \frac{\partial}{\partial f} s(y_i, f)|_{f=F_{m-1}(x_i)}, \tag{11}$$

$$h_{m,i} = \frac{\partial^2}{\partial f^2} s(y_i, f)|_{f=F_{m-1}(x_i)}. \tag{12}$$

For quadratic loss $s(y_i, f) = \frac{(y_i - f)^2}{2}$, we obtain $h_{m,i} = 1$. In this case, the Newton-update becomes the gradient update.

2.3. Penalized estimation

To avoid overfitting, we can add the objective function with a regularization term:

$$\sum_{i=1}^{n} \ell(y_i, \hat{f}_i) + \sum_{m=1}^{M} \Lambda(t_m), \tag{13}$$

where Λ penalizes the model complexity. If H is the space of linear regression with a p-dimensional predictor, i.e., $t_m(\mathbf{x}_i) = \mathbf{x}_i^\intercal \boldsymbol{\beta}_m, \boldsymbol{\beta}_m = (\beta_{1m}, ..., \beta_{pm})^\intercal$, denote

$$\Lambda(t_m) = \frac{1}{2}\lambda \sum_{j=1}^{p} \beta_{jm}^2 + \alpha \sum_{j=1}^{p} |\beta_{jm}|, \tag{14}$$

where $\lambda \geq 0, \alpha \geq 0$. Note that $\Lambda(t_m)$ provides shrinkage estimators and can conduct variable selection. Suppose that H is the space of regression trees. Each regression tree splits the whole predictor space into disjoint hyper-rectangles with sides parallel to the coordinate axes (Wang 2018b). Specifically, denote the hyper-rectangles in the m-th boosting iteration $R_{jm}, j = 1, ..., J$. Let $t_m(\mathbf{x}_i) = \beta_{jm}, \mathbf{x}_i \in R_{jm}, i = 1, ..., n, j = 1, ..., J$. With $\gamma \geq 0$, the penalty can be defined as in Chen and Guestrin (2016):

$$\Lambda(t_m) = \gamma J + \frac{1}{2}\lambda \sum_{j=1}^{p} \beta_{jm}^2 + \alpha \sum_{j=1}^{p} |\beta_{jm}|. \tag{15}$$

A different penalized estimation is to implement a shrinkage parameter $0 < \nu \leq 1$ in the update (9):

$$F_m(\mathbf{x}_i) = F_{m-1}(\mathbf{x}_i) + \nu t_m(\mathbf{x}_i), m = 1, ..., M. \tag{16}$$

2.4. Implementation and tuning parameter selection

In summary, we use Algorithm 1 coupled with the boosting algorithm to minimize the following objective function:

$$\sum_{i=1}^{n} \ell(y_i, \hat{f}_i), \tag{17}$$

where $\hat{f}_i$ is given by (8). There are two layers of iterations: the outer layer is the weights update and the inner layer is the boosting iterations. An early stop of iterations in boosting doesn't guarantee convergence. On the other hand, the output $\mathbf{f}^{(K)}$ may overfit the data. In practice, we may consider a two stage process: In the first stage, apply Algorithm 1 to obtain optimal weights of observations. In the second stage, we can use a data-driven method such as cross-validation to select optimal boosting iteration M, penalty numbers γ for trees, λ and α. The same strategy can also be applied to the robust parameter σ. However, since this parameter is typically considered a hyperparameter, a more computationally convenient approach in the literature is to conduct estimation for different values of σ and compare the results. One can begin with a large value σ with less robust estimation, and move towards smaller value σ for more robust results.

The source version of the **irboost** package is freely available from the Comprehensive R Archive Network (http://CRAN.R-project.org). The reader can install the package directly from the R prompt via

```
R> install.packages("irboost")
```

All analyses presented below are contained in a package vignette. The rendered output of the analyses is available by the R-command

```
R> library("irboost")
R> vignette("irbst",package = "irboost")
```

To reproduce the analyses, one can invoke the R code

```
R> edit(vignette("irbst",package = "irboost"))
```

3. Data examples

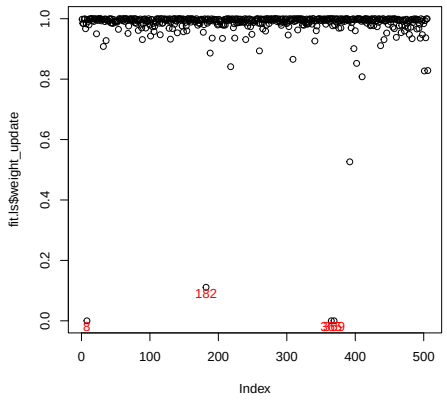
3.1. Robust boosting for regression

In this example, we predict median value of owner-occupied homes in suburbs of Boston, with data publicly available from the UCI machine learning data repository. There are 506 observations and 13 predictors. A different robust estimation can be found in Wang (2020).

```
R> urlname <- "https://archive.ics.uci.edu/ml/"
R> filename <- "machine-learning-databases/housing/housing.data"
R> dat <- read.table(paste0(urlname, filename), sep="", header=FALSE)
R> dat <- as.matrix(dat)
R> colnames(dat) <- c("CRIM", "ZN", "INDUS", "CHAS", "NOX", "RM", "AGE",
+                     "DIS", "RAD", "TAX", "PTRATIO", "B", "LSTAT", "MEDV")
R> p <- dim(dat)[2]
```

We apply the IRBoost with concave component `bcave` and convex component least squares. The observation weights are plotted with highlighted four smallest values. The corresponding four observations are considered outliers. We can plot the original median housing price vs

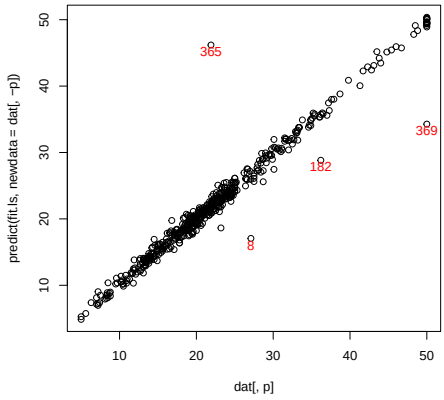
```
R> library("irboost")
R> fit.ls <- irboost(dat[,-p], dat[,p], cfun="bcave",s=10,
+                    dfun="reg:squarederror", verbose=0,
+                    max.depth=2, nrounds=50)
R> plot(fit.ls$weight_update)
R> id <- sort.list(fit.ls$weight_update)[1:4]
R> text(id, fit.ls$weight_update[id]-0.02, id, col="red")
```



the predicted values. Not surprisingly, those 4 observations with the smallest weights have poor predictions. We can view feature importance/influence from the learned model. The figure shows that the top two factors to predict median housing price are average number of rooms per dwelling (RM) and percentage values of lower status of the population (LSTAT).

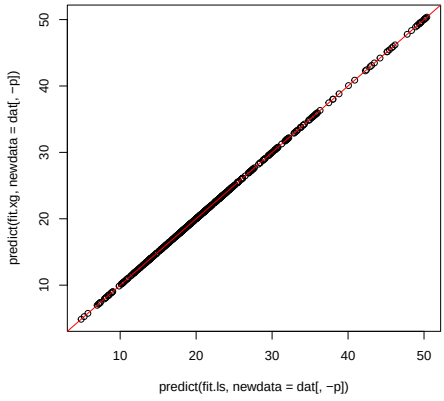
```
R> gr <- xgboost::xgb.plot.tree(model = fit.ls, trees=0:1)
```

```
R> plot(dat[,p], predict(fit.ls, newdata=dat[, -p]))
R> text(dat[id,p], predict(fit.ls, newdata=dat[id, -p])-1, id, col="red")
```

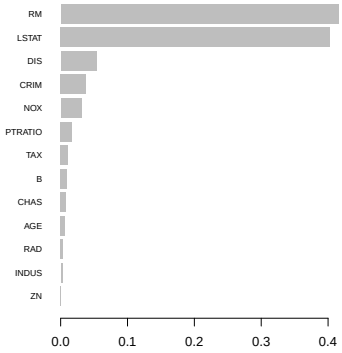


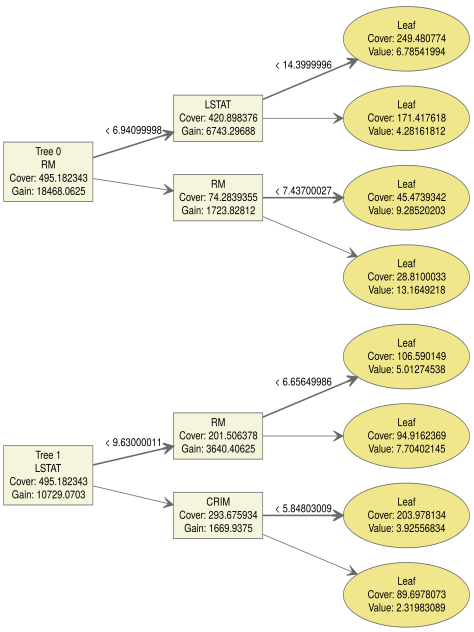
The IRBoost algorithm in **irboost** is a weighted `xgboost`, where the weights are tuned by robust argument. This can be illustrated below.

```
R> library("xgboost")
R> fit.xg <- xgboost(dat[,-p], dat[,p], weight=fit.ls$weight_update,
+                    objective="reg:squarederror", verbose=0, max.depth=2,
+                    nrounds=fit.ls$niter)
R> plot(predict(fit.ls, newdata=dat[, -p]), predict(fit.xg, newdata=dat[, -p]))
R> abline(0, 1, col="red")
```



```
R> importance_matrix <- xgboost::xgb.importance(model = fit.ls)
R> xgboost::xgb.plot.importance(importance_matrix = importance_matrix)
```



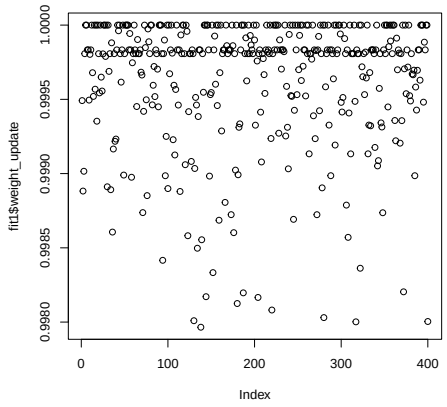


3.2. Robust logistic boosting

A binary classification problem was proposed by Long and Servedio (2010). Response variable y is randomly chosen to be -1 or +1 with equal probability. We randomly generate symbols A, B and C with probability 0.25, 0.25 and 0.5, respectively. The predictor vector **x** with 21 elements is generated as follows. If A is obtained, $x_j = y, j = 1, ..., 21$. If B is generated, $x_j = y, i = 1, ..., 11, x_j = -y, j = 12, ..., 21$. If C is generated, $x_j = y$, where j is randomly chosen from 5 out of 1-11, and 6 out of 12-21. For the remaining $j \in (1, 21)$, $x_j = -y$. We generate training data $n = 400$ and test data $n = 200$.

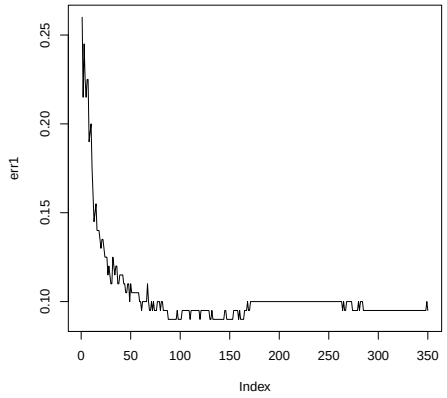
We fit a robust logistic boosting model with concave component `acave`, where the maximum depth of a tree is 5. Other concave components in Table 1 can be applied similarly. We can

```
R> set.seed(1947)
R> dat <- dataLS(ntr=400, nte=200, percon=0)
R> fit1 <- irboost(dat$xtr, dat$ytr, cfun="acave",s=3, dfun="binary:logitraw",
+                  verbose=0, max.depth=5, nrounds=50)
R> plot(fit1$weight_update)
```

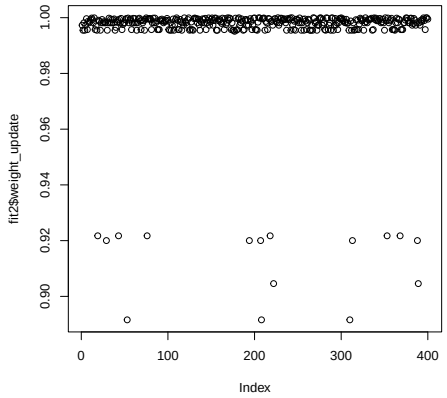


compute prediction error of test data at each boosting iteration. Furthermore, we simulate data with 10% contamination of response variables, and apply the IRBoost. In the third robust logistic boosting, we reduce parameter value σ (`s` in the `irboost` function) for more robust estimation. As a result, some observations would have decreased weights in the model.

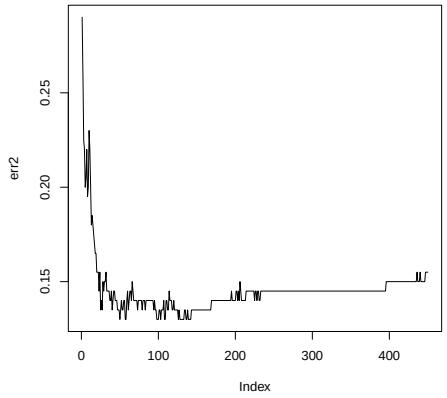
```
R> err1 <- rep(NA, 100)
R> for(i in 1:fit1$niter){
+    pred1 <- predict(fit1, newdata=dat$xte, iterationrange=c(1, i+1))
+    err1[i] <- mean(sign(pred1)!=dat$yte)
+  }
R> plot(err1, type="l")
```



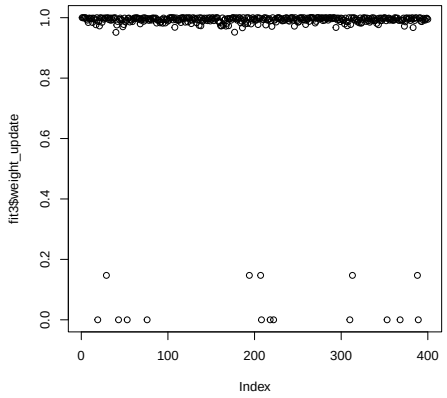

```
R> dat2 <- dataLS(ntr=400, nte=200, percon=0.1)
R> fit2 <- irboost(dat2$xtr, dat2$ytr, cfun="acave",s=3, dfun="binary:logitraw",
+                  verbose=0, max.depth=5, nrounds=50)
R> plot(fit2$weight_update)
```



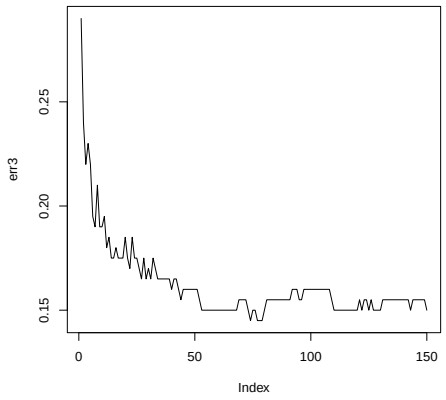
```
R> err2 <- rep(NA, 100)
R> for(i in 1:fit2$niter){
+    pred2 <- predict(fit2, newdata=dat2$xte, iterationrange=c(1, i+1))
+    err2[i] <- mean(sign(pred2)!=dat2$yte)
+  }
R> plot(err2, type="l")
```




```
R> fit3 <- irboost(dat2$xtr, dat2$ytr, cfun="acave",s=1, dfun="binary:logitraw",
+                   verbose=0, max.depth=5, nrounds=50)
R> plot(fit3$weight_update)
```



```
R> err3 <- rep(NA, 100)
R> for(i in 1:fit3$niter){
+    pred3 <- predict(fit3, newdata=dat2$xte, iterationrange=c(1, i+1))
+    err3[i] <- mean(sign(pred3)!=dat2$yte)
+  }
R> plot(err3, type="l")
```



3.3. Robust multiclass boosting

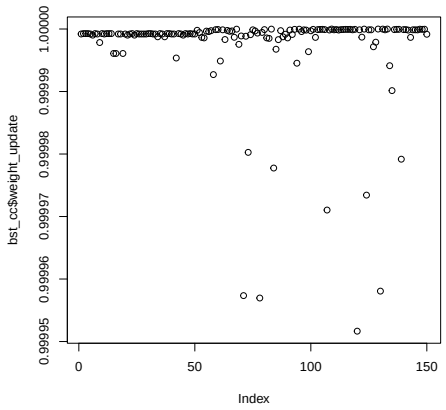
In a 3-class classification in iris dataset, **xgboost** generates classification error 0.02. Letting the initial boosting parameters the same, the IRBoost algorithm in **irboost** automatically updates the observation weights and leads to a different decision while maintaining the similar classification accuracy.

```
R> lb <- as.numeric(iris$Species)-1
R> num_class <- 3
R> set.seed(11)
R> # xgboost
R> bst <- xgboost(data=as.matrix(iris[, -5]), label=lb, max_depth=4,
+                 eta=0.5, nthread=2, nrounds=10, subsample=0.5,
+                 objective="multi:softprob", num_class=num_class)
R> # predict for softmax returns num_class probability numbers per case:
R> pred <- predict(bst, as.matrix(iris[, -5]))
R> # reshape it to a num_class-columns matrix
R> pred <- matrix(pred, ncol=num_class, byrow=TRUE)
R> # convert the probabilities to softmax labels
R> pred_labels <- max.col(pred)-1

R> # classification error
R> sum(pred_labels!=lb)/length(lb)

[1] 0.02

R> # irboost
R> bst_cc <- irboost(x=as.matrix(iris[, -5]), y=lb, cfun="acave", s=50,
+                    dfun="multi:softprob", verbose=0,
+                    max.depth=4, eta=0.5, nthread=2, nrounds=10,
+                    subsample=0.5, num_class=num_class)
```

The weights are shown in a figure blow. Rerun `xgboost` but with new weights from `irboost`. Compare model `bst` and `fit7`, with small change of weights, a different classification rule is obtained with similar error.

```
R> fit7 <- xgboost(data=as.matrix(iris[, -5]), label=lb,
+                  weight=bst_cc$weight_update, max_depth=4,
+                  eta=0.5, nthread=2, nrounds=10, subsample=0.5,
+                  objective="multi:softprob", num_class=num_class)
R> pred7 <- predict(fit7, as.matrix(iris[, -5]))
R> pred7 <- matrix(pred7, ncol=num_class, byrow=TRUE)
R> # convert the probabilities to softmax labels
R> pred7_labels <- max.col(pred7) - 1

R> # classification error
R> sum(pred7_labels != lb)/length(lb)
```

```
R> plot(bst_cc$weight_update)
```




```
[1] 0.02
```

```
R> table(pred_labels, pred7_labels)
```

```
           pred7_labels
pred_labels  0  1  2
          0 50  0  0
          1  0 47  0
          2  0  2 51
```

3.4. Robust Poisson boosting

A survey collected from 3066 Americans was studied on health care utilization in lieu of doctor office visits (Heritier, Cantoni, Copt, and Victoria-Feser 2009). The data contained 24 risk factors. Robust Poisson regression was conducted in Wang (2020). Here robust Poisson boosting model is fitted with concave component `ccave`. The observation weights are estimated. The doctor office visits in two years are highlighted for the 8 smallest weights, ranging from 200 to 750. We can view feature importance/influence from the learned model. The

```
R> data(docvisits, package="mpath")
R> x <- model.matrix(~age+factor(gender)+factor(race)+factor(hispan)
+                    +factor(marital)+factor(arthri)+factor(cancer)
+                    +factor(hipress)+factor(diabet)+factor(lung)
+                    +factor(hearth)+factor(stroke)+factor(psych)
+                    +factor(iadla)+factor(adlwa)+edyears+feduc
+                    +meduc+log(income+1)+factor(insur)+0, data=docvisits)
R> fit.pos <- irboost(x, docvisits$visits, cfun="ccave",s=20,
+                      dfun="count:poisson", verbose=0, max.depth=1, nrounds=50)
R> plot(fit.pos$weight_update)
R> id <- sort.list(fit.pos$weight_update)[1:8]
R> text(id, fit.pos$weight_update[id]-0.02, docvisits$visits[id], col="red")
```

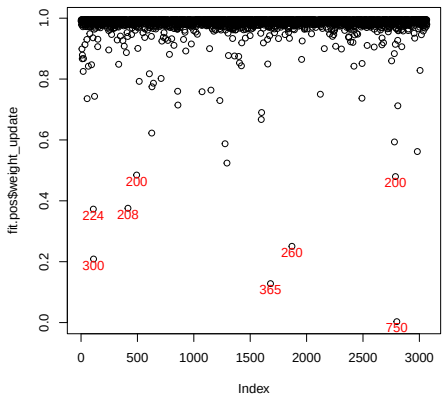
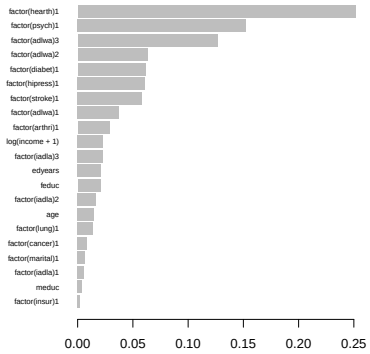


figure shows that the top two reasons of doctor office visits are heart disease and psychiatric problems.

```
R> importance_matrix <- xgboost::xgb.importance(model = fit.pos)
R> xgboost::xgb.plot.importance(importance_matrix = importance_matrix)
```



3.5. Robust survival boosting with accelerated failure time model

It is worth noting that the Cox regression in survival analysis is based on partial likelihood function, which doesn't follow the IRBoost algorithm. Alternatively, one may apply robust survival regression with accelerated failure time model in **irboost**. The following code provides survival analysis in patients with advanced lung cancer from the North Central Cancer Treatment Group. Performance scores rate how well the patient can perform usual daily activities.

```
R> library("survival")
R> lung1 <- lung[complete.cases(lung), ]
R> y_upper_bound <- rep(NA, dim(lung1)[1])
R> for(i in 1:dim(lung1)[1]){
+    if(lung1$status[i]==2){
+      y_upper_bound[i] <- lung1$time[i]
+    }else
+      y_upper_bound[i] <- "+Inf"
+  }
R> x <- as.matrix(lung1[, !names(lung1) %in% c("time", "status")])
R> dtrain <- xgb.DMatrix(data=x, label_lower_bound=lung1$time,
+                         label_upper_bound=y_upper_bound)
R> params <- list(objective='survival:aft',
+                 eval_metric='aft-nloglik',
+                 aft_loss_distribution='normal',
+                 aft_loss_distribution_scale=1.20,
+                 max_depth=3)
R> bst <- xgb.train(params, dtrain, nrounds=50)
R> #evaluate model prediction accuracy
R> library("Hmisc")
R> rcorr.cens(predict(bst, dtrain), Surv(lung1$time, lung1$status))

       C Index            Dxy           S.D.              n
  9.024991e-01   8.049981e-01   2.375401e-02   1.670000e+02
       missing     uncensored Relevant Pairs     Concordant
  0.000000e+00   1.200000e+02   2.112800e+04   1.906800e+04
     Uncertain
  6.572000e+03

R> bst_cc <- irboost_aft(params, dtrain, cfun="hcave", s=3, nrounds=50)
R> summary(bst_cc$weight_update)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.7593  0.8078  0.8522  0.8851  1.0000  1.0000

R> rcorr.cens(predict(bst_cc, dtrain), Surv(lung1$time, lung1$status))

       C Index            Dxy           S.D.              n
  9.805945e-01   9.611889e-01   5.940054e-03   1.670000e+02
```

```
      missing     uncensored Relevant Pairs     Concordant
 0.000000e+00   1.200000e+02   2.112800e+04   2.071800e+04
    Uncertain
 6.572000e+03
```

```
R> #could be overfitting due to updated nrounds?
R> bst_cc$niter
```

```
[1] 250
```

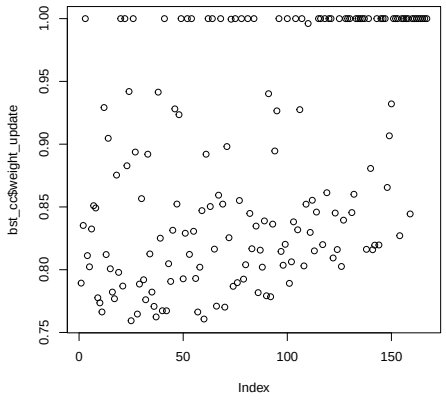
```
R> #udpate data with weights and rerun xgboost with the same niter
R> setinfo(dtrain, 'weight', bst_cc$weight_update)
```

```
[1] TRUE
```

```
R> #compare bst and bst_xg with different weights
R> bst_xg <- xgb.train(params, dtrain, nrounds=50)
R> rcorr.cens(predict(bst_xg, dtrain), Surv(lung1$time, lung1$status))
```

```
      C Index           Dxy          S.D.             n
 8.984286e-01   7.968573e-01   2.427654e-02   1.670000e+02
      missing     uncensored Relevant Pairs     Concordant
 0.000000e+00   1.200000e+02   2.112800e+04   1.898200e+04
    Uncertain
 6.572000e+03
```

```
R> plot(bst_cc$weight_update)
```



4. Conclusion

In this article we propose IRBoost as a unified robust boosting algorithm, and illustrate its applications in regression, generalized linear models, classification and time-to-event data analysis. The method can be used for outlier detection and can provide more robust predictive models. Based on existing weighted boosting software, we can explore the developed models on variable importance and the underlying trees. The R package **irboost** is a useful tool in the machine learning applications.

References

Barnwal A, Cho H, Hocking TD (2020). "Survival regression with accelerated failure time model in XGBoost." *arXiv preprint arXiv:2006.04920*.

Bühlmann P, Hothorn T (2007). "Boosting algorithms: Regularization, prediction and model fitting (with discussion)." *Statistical Science*, **22**(4), 477–505.

Chen T, Guestrin C (2016). "Xgboost: A scalable tree boosting system." In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 785–794.

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

Friedman J, Hastie T, Tibshirani R (2000). "Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors)." *Annals of Statistics*, **28**(2), 337–407.

Heritier S, Cantoni E, Copt S, Victoria-Feser MP (2009). *Robust Methods in Biostatistics*, volume 825. John Wiley & Sons.

Long PM, Servedio RA (2010). "Random classification noise defeats all convex potential boosters." *Machine learning*, **78**(3), 287–304.

Sigrist F (2020). "Gradient and Newton Boosting for Classification and Regression." *arXiv e-prints*. https://arxiv.org/abs/1808.03064, 1808.03064.

Wang Z (2018a). "Quadratic majorization for nonconvex loss with applications to the boosting algorithm." *Journal of Computational and Graphical Statistics*, **27**(3), 491–502.

Wang Z (2018b). "Robust boosting with truncated loss functions." *Electronic Journal of Statistics*, **12(1)**, 599–650. doi:10.1214/18-EJS1404. URL https://doi.org/10.1214/18-EJS1404.

Wang Z (2020). "Unified Robust Estimation." *arXiv e-prints*, arXiv:2010.02848. https://arxiv.org/abs/2010.02848, 2010.02848.

Affiliation:

Zhu Wang
Department of Population Health Sciences
UT Health San Antonio
USA
E-mail: zhuwang@gmail.com