An Introduction to irboost

Zhu Wang

April 17, 2024

Contents

1 Introduction **1**

2 Installation **1**

3 Applications **1**
3.1 Robust boosting for regression 1
3.2 Robust logistic boosting 6
3.3 Robust multiclass boosting 9
3.4 Robust Poisson boosting 10
3.5 Robust survival boosting with accelerated failure time model 11

1 Introduction

The R package **irboost** fits a predictive model using iteratively reweighted boosting (IRBoost) to minimize robust loss functions within the CC-family (concave-convex). This constitutes an application of iteratively reweighted convex Optimization (IRCO), where convex optimization is performed using the functional descent boosting algorithm. IRBoost assigns weights to facilitate outlier identification. Applications include robust generalized linear models and robust accelerated failure time models. The theory and algorithms in this implementation are described in Wang (2021).

2 Installation

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

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

3 Applications

3.1 Robust boosting for regression

In this example, we predict the median value of owner-occupied homes in the suburbs of Boston, using data publicly available from the UCI machine learning data repository. The dataset comprises 506 observations and 13 predictors. Wang (2024) provides an alternative robust estimation for comparison.

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

We apply IRBoost with the concave component `bcave` and the convex component least squares.

```
library("irboost")
param <- list(objective = "reg:squarederror", max_depth = 2)
fit1 <- irboost(data = dat[, -p], label = dat[, p], cfun = "bcave",
    s = 10, params = param, verbose = 0, nrounds = 50)
plot(fit1$weight_update, ylab = "Weight")  # plot robustness weights
id <- sort.list(fit1$weight_update)[1:4]  # 4 obs. with smallest weights
text(id, fit1$weight_update[id] - 0.02, id, col = "red")  # highlight 4 obs.
```

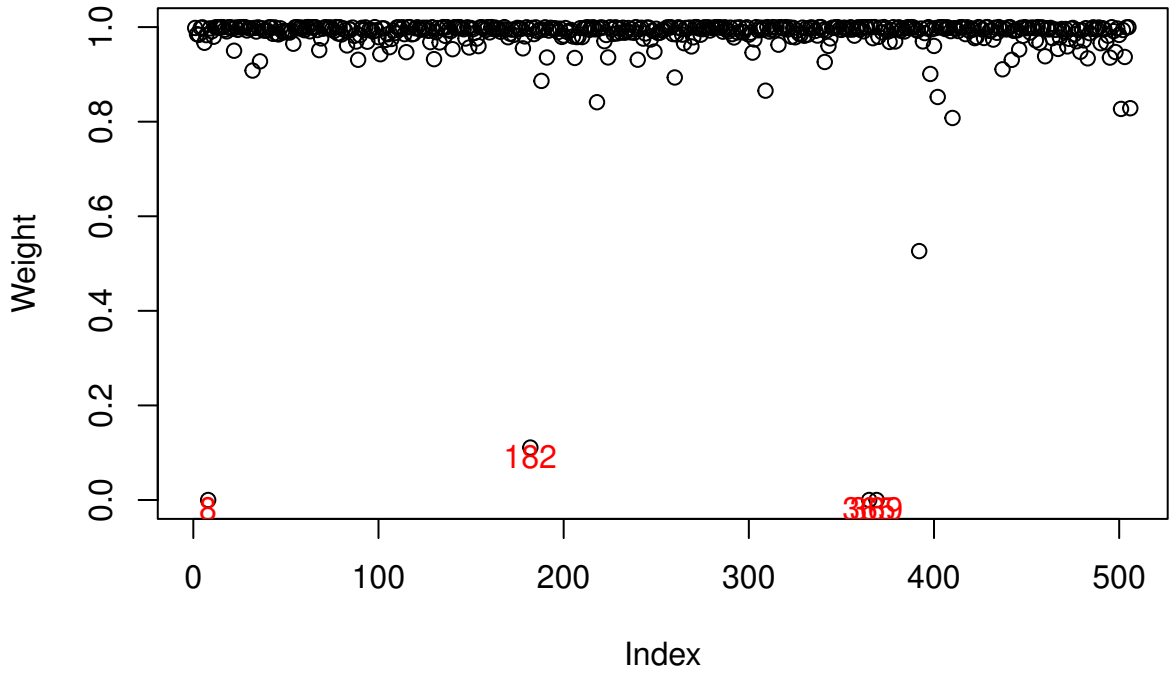

Figure 1: Robustness weights of IRBoost for the Boston housing data.

Figure 1 displays the observation weights used when IRBoost converges, highlighting the four smallest values, which are considered outliers. We plot the observed median housing prices against the predicted values in Figure 2. Notably, the four observations with the smallest weights deviate significantly from their predicted values, but this outcome is not surprising.

```
par(pty = "s")
plot(dat[, p], predict(fit1, newdata = dat[, -p]), xlab = "Observations",
    ylab = "Predictions")  # obs. vs predictions
text(dat[id, p], predict(fit1, newdata = dat[id, -p]) - 1, id,
    col = "red")  # highlight 4 obs. with smallest weights
abline(0, 1, col = "red")  # 45-degree line
```

IRBoost returns a weighted boosting estimation. The implementation of `irboost` is equivalent to `xgboost`

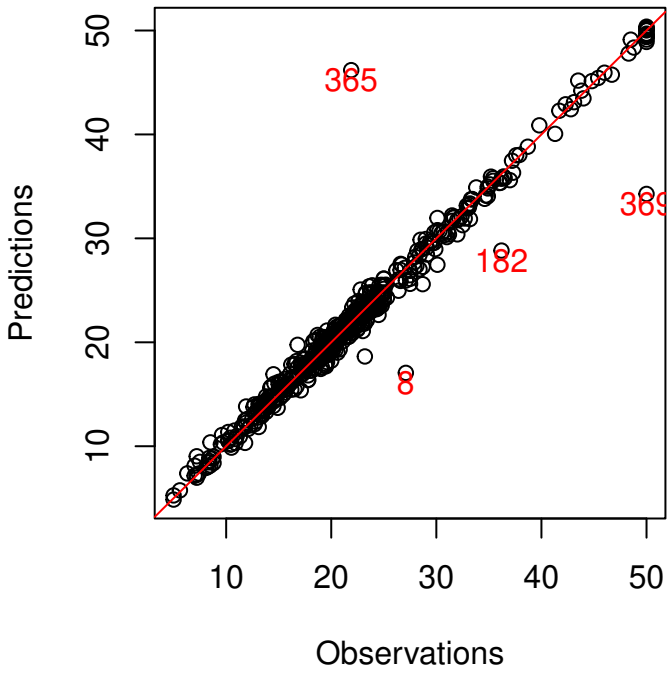

Figure 2: Observed and predicted values for the Boston housing data.

with weights. Using the weights from `irboost` enables identical predictions as with `xgboost`. This equivalence is illustrated in the following example with Figure 3.

```
library("xgboost")
fit_xg <- xgboost::xgboost(data = dat[, -p], label = dat[, p],
    weight = fit1$weight_update, params = param, verbose = 0,
    nrounds = fit1$niter)
par(pty = "s")  # plot type square between irboost and xgboost
plot(predict(fit1, newdata = dat[, -p]), predict(fit_xg, newdata = dat[,
    -p]), xlab = "Predictions by irboost", ylab = "Predictions by xgboost")
abline(0, 1, col = "red")  # 45-degree line
```

We can compare computing times between `irboost` and `xgboost`. As shown below, both computing tasks completed within one second on an Intel® Core™ i9-10900X CPU @ 3.70GHz × 8 processor. Since the former involves iterative reweighting runs of `xgboost`, it is expected to take more computing time than a single run of `xgboost`.

```
# computing time for irboost
system.time(irboost(data = dat[, -p], label = dat[, p], cfun = "bcave",
    s = 10, params = param, verbose = 0, nrounds = 50))["elapsed"]
```

```
## elapsed
##   0.643
```

```
# computing time for xgboost
system.time(xgboost::xgboost(data = dat[, -p], label = dat[,
    p], weight = fit1$weight_update, params = param, verbose = 0,
    nrounds = fit1$niter))["elapsed"]
```

```
## elapsed
##   0.121
```

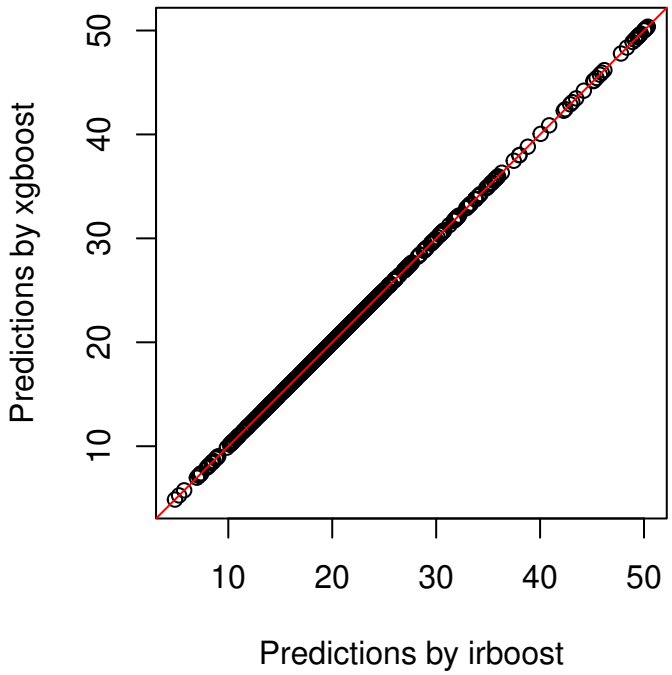

Figure 3: Comparison of irboost and xgboost with robustness weights.

Feature importance from the learned model is displayed in Figure 4. The figure reveals that the top two factors for predicting median housing prices are the average number of rooms per dwelling (RM) and the percentage values of the lower status of the population (LSTAT).

```
importance_matrix <- xgboost::xgb.importance(model = fit1)  # importance metric
xgboost::xgb.plot.importance(importance_matrix = importance_matrix)  # plot
```

The first tree used to build the model is depicted in Figure 5.

```
xgboost::xgb.plot.tree(model = fit1, trees = 0)
```

We can optimize tuning parameters using cross-validation with the built-in functions in **xgboost**. First, update the data format with the estimated robustness weights, then run the following code to determine the optimal IRBoost iteration.

```
dtrain <- xgboost::xgb.DMatrix(data = dat[, -p], label = dat[,
    p])  # create a DMatrix for training data
xgboost::setinfo(dtrain, "weight", fit1$weight_update)  # set weight information
param <- list(booster = "gbtree", objective = "reg:squarederror")
set.seed(136)  # set the seed for reproducibility
xgbcv <- xgboost::xgb.cv(params = param, data = dtrain, nrounds = 200,
    early_stopping_rounds = 20, nfold = 5, prediction = TRUE)
```

```
xgbcv$best_iteration
```

```
## [1] 37
```

Continuing in the same vein, we can identify the optimal robustness parameter θ, and the corresponding IRBoost iteration. For instance, to select a preferable θ from the set $\{5, 10\}$, the cross-validation results below indicate that $\theta = 5$ yields a smaller root mean square error on the test data. Moreover, this procedure identifies the optimal IRBoost iteration as 39.

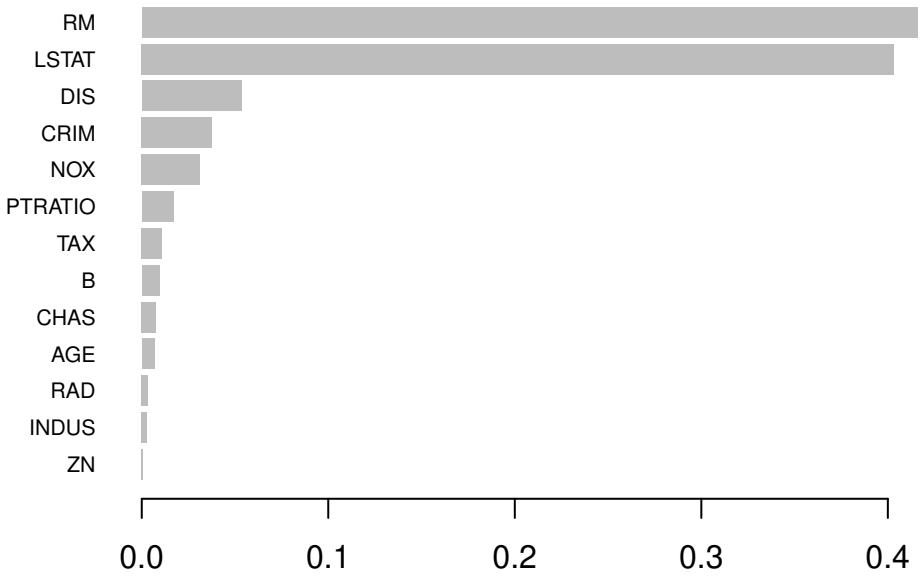

Figure 4: Variable importance measures for the Boston housing data.

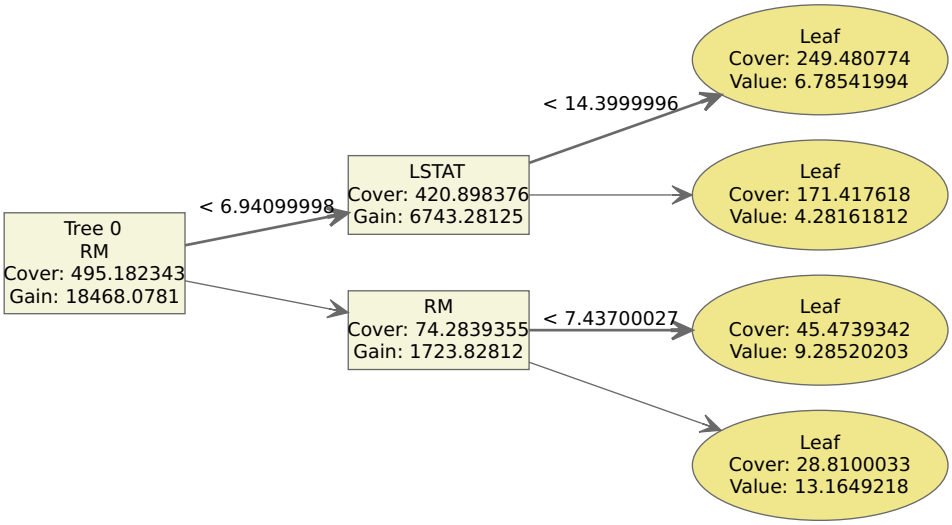

Figure 5: First tree in IRBoost for the Boston housing data.

```
dtrain_cv <- xgboost::xgb.DMatrix(data = dat[, -p], label = dat[,
    p])  # training data
robustness_param <- c(5, 10)  # two theta values
res <- NULL
for (i in 1:length(robustness_param)) {
    fit_init <- irboost(data = dat[, -p], label = dat[, p], cfun = "bcave",
        s = robustness_param[i], params = list(objective = "reg:squarederror",
            max_depth = 2), verbose = 0, nrounds = 50)  # fit irboost model
    xgboost::setinfo(dtrain_cv, "weight", fit_init$weight_update)  # new weights
    set.seed(136)
    xgbcv <- xgboost::xgb.cv(params = param, data = dtrain_cv,
        early_stopping_rounds = 20, nrounds = 200, nfold = 5,
        prediction = TRUE)  # 5-fold CV
    reslog <- xgbcv$evaluation_log[xgbcv$best_iteration]  # best values in CV
    tmp <- unlist(c(theta = robustness_param[i], reslog))  # combine theta
    res <- rbind(res, tmp)  # combine results from previous theta
    rownames(res) <- NULL
}
```

```
print(res, digits = 2)
```

```
##      theta iter train_rmse_mean train_rmse_std test_rmse_mean
## [1,]     5   39            0.30          0.019            2.3
## [2,]    10   37            0.37          0.036            3.0
##      test_rmse_std
## [1,]          0.31
## [2,]          0.33
```

3.2 Robust logistic boosting

A binary classification problem, as proposed by Long and Servedio (2010), involves a response variable y randomly chosen to be -1 or +1 with equal probability. Symbols A, B, and C are randomly generated with probabilities 0.25, 0.25, and 0.5, respectively. The predictor vector $\vec{x}$ with 21 elements is generated as follows: if A is obtained, $x_j = y$ for $j = 1, ..., 21$. If B is generated, $x_j = y$ for $j = 1, ..., 11$, and $x_j = -y$ for $j = 12, ..., 21$. If C is generated, $x_j = y$, where j is randomly chosen from the range of 1 to 11 with a selection of 5 elements, and from the range of 12 to 21 with a selection of 6 elements. For the remaining $j \in \{1, 2, 3, ..., 21\}$, $x_j = -y$. The training data is generated with $n = 400$ samples, and the test data with $n = 200$ samples.

We fit a robust logistic boosting model with the concave component `acave`, setting the maximum depth of a tree to 5. With a large parameter $\theta = 100$, the robustness weights are very close to 1s as shown in Figure 6.

```
set.seed(1947)
dat2 <- dataLS(ntr = 400, nte = 200, percon = 0)  # percon=0 means clean data
param <- list(objective = "binary:logitraw", max_depth = 5)
fit2 <- irboost(data = dat2$xtr, label = dat2$ytr, cfun = "acave",
    s = 100, params = param, verbose = 0, nrounds = 100)
plot(fit2$weight_update, ylab = "Weight")  # plot robustness weights
```

To add outliers, we simulate data with 10% contamination in the response variables of the training data and then apply IRBoost. Figure 7 displays the robustness weights obtained from the algorithm.

```
set.seed(158)
dat3 <- dataLS(ntr = 400, nte = 200, percon = 0.1)  # 10% data contamination
param <- list(objective = "binary:logitraw", max_depth = 5)
```

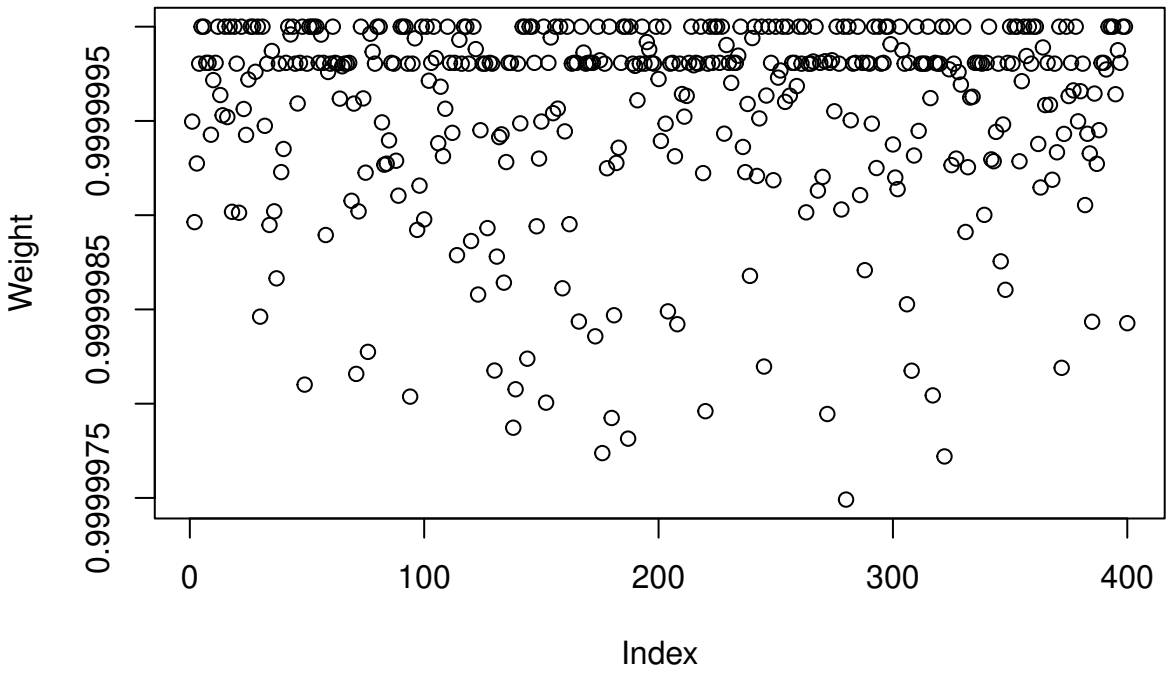

Figure 6: Robustness weights of IRBoost for the simulation data.

```
fit3 <- irboost(data = dat3$xtr, label = dat3$ytr, cfun = "acave",
    s = 3, params = param, verbose = 0, nrounds = 100)
plot(fit3$weight_update, ylab = "Weight")  # plot robustness weights
```

In the third robust logistic boosting, we set the robustness hyperparameter value θ to 1 (`s=1` in the `irboost` function) for a more robust estimation. Consequently, certain observations exhibit decreased weights, as illustrated in Figure 8.

```
param <- list(objective = "binary:logitraw", max_depth = 5)
fit4 <- irboost(data = dat3$xtr, label = dat3$ytr, cfun = "acave",
    s = 1, params = param, verbose = 0, nrounds = 100)
plot(fit4$weight_update, ylab = "Weight")  # plot robustness weights
```

The prediction accuracy can be compared for different models. The prediction error of the test data at each IRBoost iteration is depicted in Figure 9, demonstrating that the most accurate predictions come from the robust model, even with outliers.

```
err2 <- err3 <- err4 <- rep(NA, 100)
for (i in 1:100) {
    pred2 <- predict(fit2, newdata = dat2$xte, iterationrange = c(1,
        i + 1))  # prediction with the first i trees
    err2[i] <- mean(sign(pred2) != dat2$yte)  # error at iteration i
    pred3 <- predict(fit3, newdata = dat3$xte, iterationrange = c(1,
        i + 1))
    err3[i] <- mean(sign(pred3) != dat3$yte)
    pred4 <- predict(fit4, newdata = dat3$xte, iterationrange = c(1,
        i + 1))
    err4[i] <- mean(sign(pred4) != dat3$yte)
}
plot(err2[1:100], ylim = c(0.05, 0.3), type = "l", xlab = "IRBoost iteration",
```

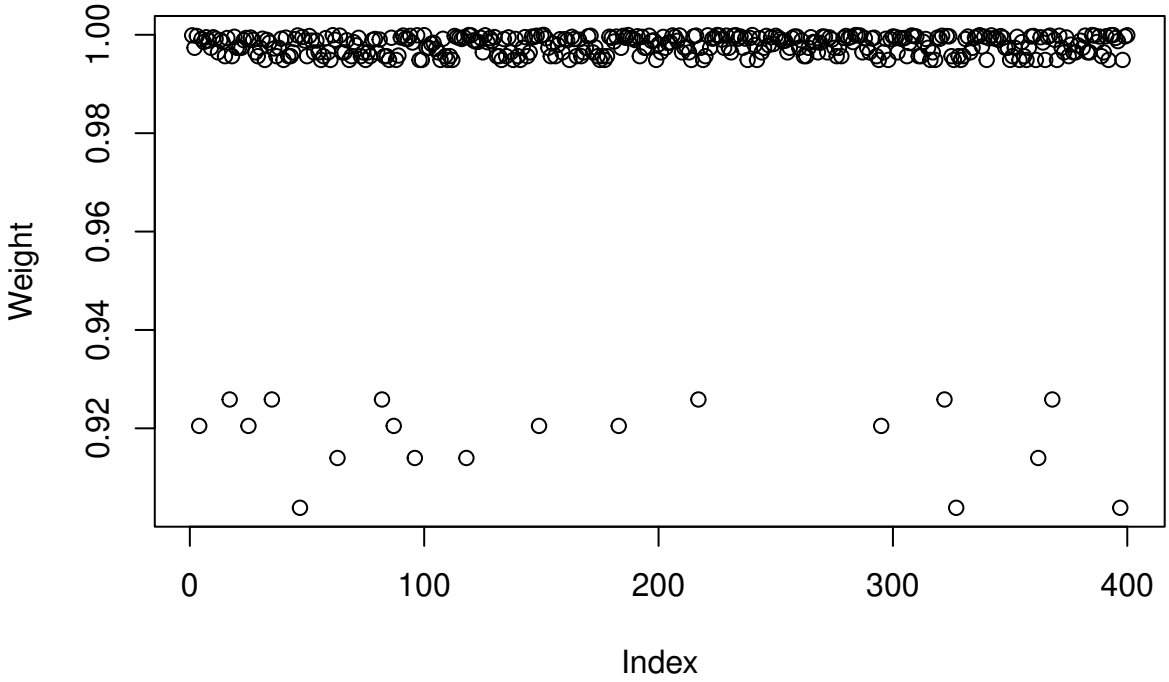

Figure 7: Robustness weights of IRBoost with $\theta = 3$ for the contaminated simulation data.

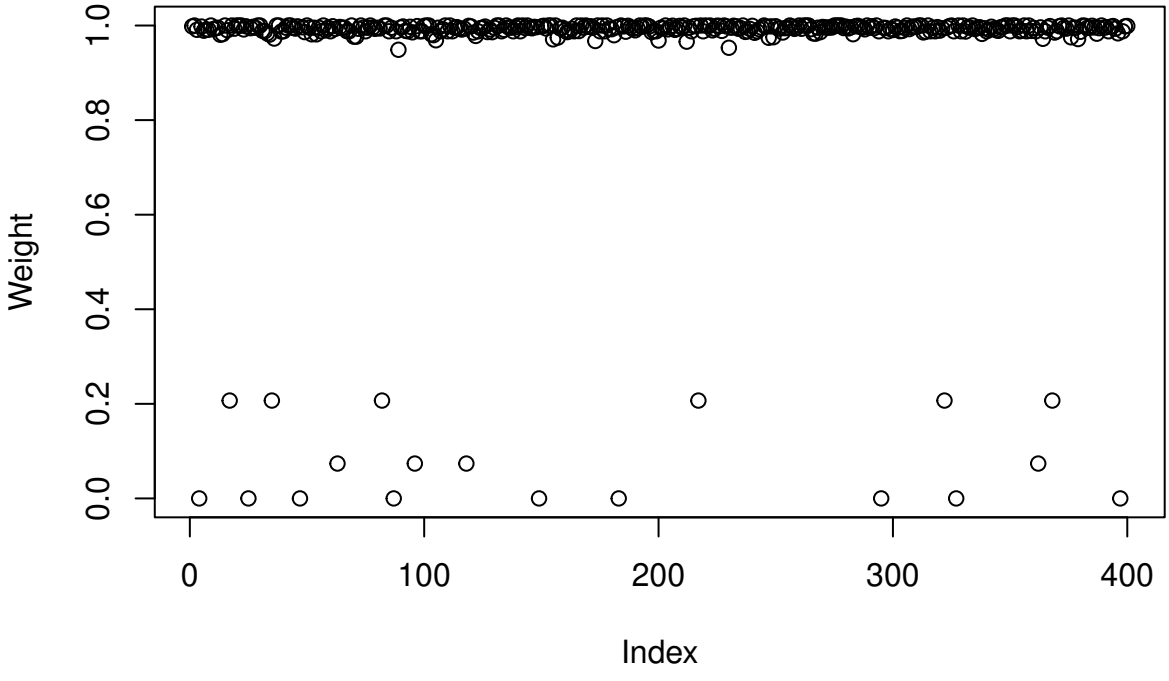

Figure 8: Robustness weights of IRBoost with $\theta = 1$ for the contaminated simulation data.

```
    ylab = "Classification error")
points(err3[1:100], col = "red", type = "l", lty = "dashed")
points(err4[1:100], col = "blue", type = "l", lty = "dotted")
legend("topright", lty = c("solid", "dashed", "dotted"), col = c("black",
    "red", "blue"), legend = c(expression(paste("clean data with ",
    theta == 100)), expression(paste("cont'd data with ", theta ==
    3)), expression(paste("cont'd data with ", theta == 1))))
```

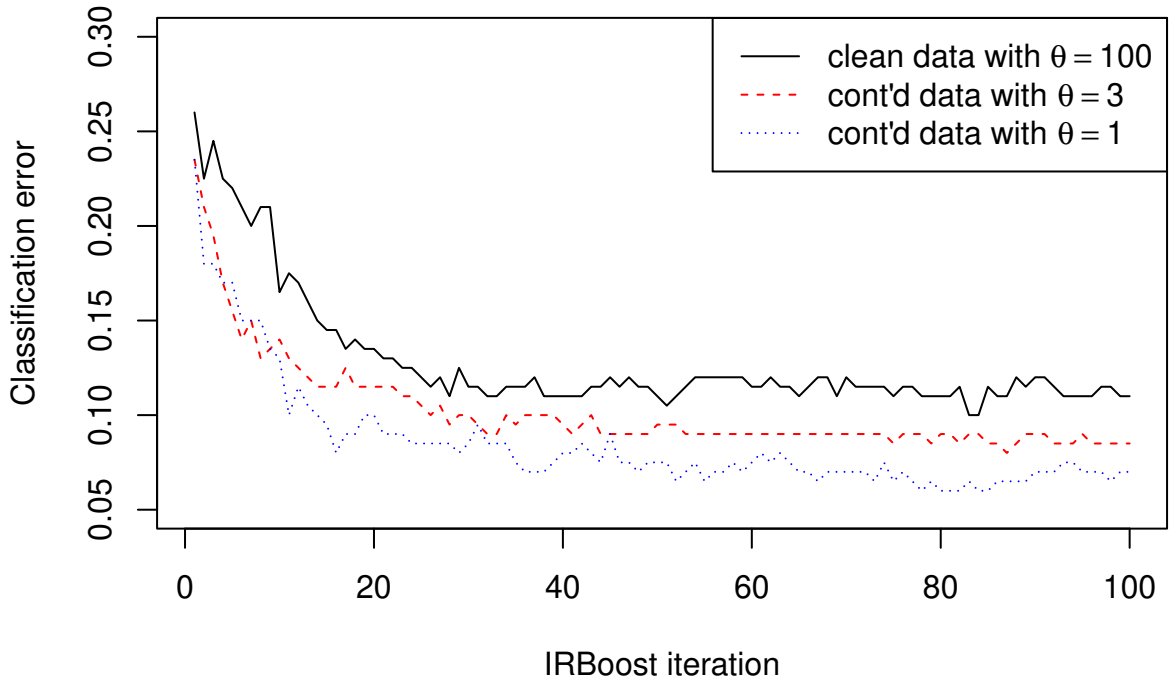

Figure 9: Classification errors and IRBoost iterations for the simulation data.

3.3 Robust multiclass boosting

In a 3-class classification using the `iris` dataset, we run IRBoost with concave function `acave` and $\theta = 1$. The robustness weights are illustrated in Figure 10, and the model achieves perfect prediction.

```
lb <- as.numeric(iris$Species) - 1  # convert text to numeric values
num_class <- 3
set.seed(11)
param <- list(objective = "multi:softprob", max_depth = 4, eta = 0.5,
    nthread = 2, subsample = 0.5, num_class = num_class)
fit5 <- irboost(data = as.matrix(iris[, -5]), label = lb, cfun = "acave",
    s = 1, params = param, verbose = 0, nrounds = 10)
```

```
plot(fit5$weight_update, ylab = "Weight")  # plot robustness weights
```

```
# compute num_class probabilities per case
pred5 <- predict(fit5, newdata = as.matrix(iris[, -5]))
# reshape to a num_class-columns matrix
pred5 <- matrix(pred5, ncol = num_class, byrow = TRUE)
pred5_labels <- max.col(pred5) - 1  # probabilities to labels
sum(pred5_labels != lb)  # classification errors
```

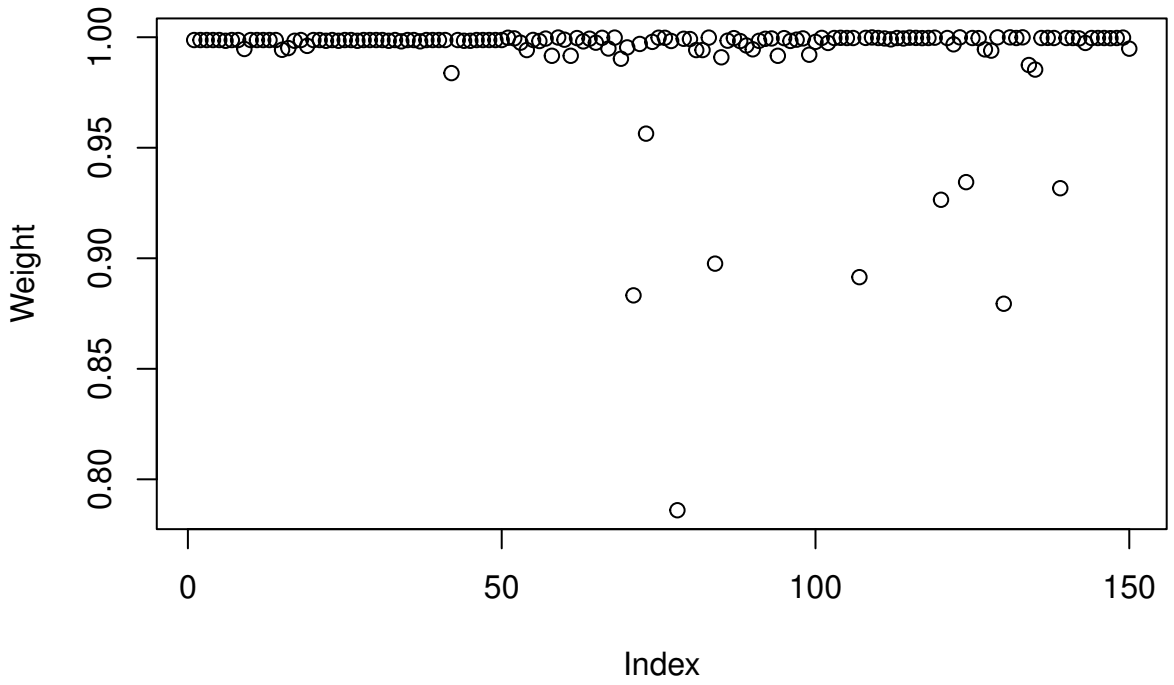

Figure 10: Robustness weights of IRBoost for the iris data.

```
## [1] 0
```

3.4 Robust Poisson boosting

A survey, collected from 3066 Americans, studied health care utilization Heritier et al. (2009), Wang (2024). The dataset includes information on doctor office visits and 24 risk factors. A robust Poisson boosting model is fitted with the concave component `ccave`, and the estimated robustness weights are illustrated in Figure 11. Doctor office visits in two years are highlighted for the 8 smallest weights, ranging from 200 to 750.

```
data(docvisits, package = "mpath")
# convert factors and other types of variables into a
# numeric matrix
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)
param <- list(objective = "count:poisson", max_depth = 1)
fit6 <- irboost(data = x, label = docvisits$visits, cfun = "ccave",
    s = 20, params = param, verbose = 0, nrounds = 50)
```

```
plot(fit6$weight_update, ylab = "Weight")  # plot robustness weights
id <- sort.list(fit6$weight_update)[1:8]  # 8 obs. with smallest weights
text(id, fit6$weight_update[id] - 0.02, docvisits$visits[id],
    col = "red")  # highlight 8 obs.
```

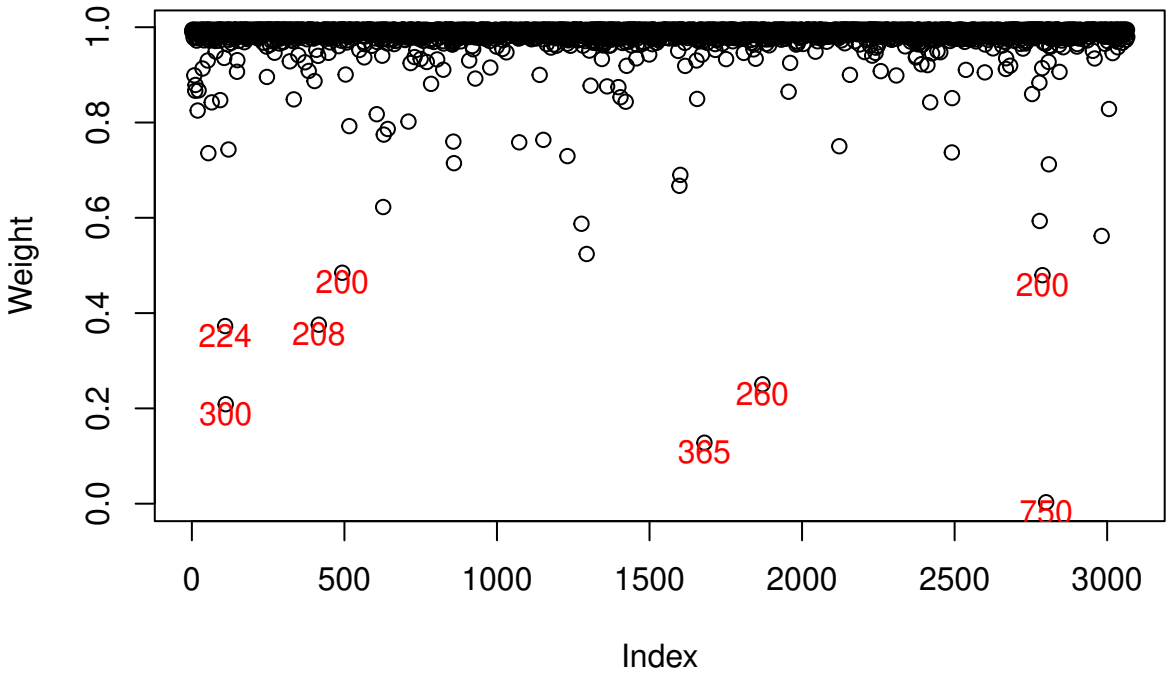

Figure 11: Robustness weights of IRBoost for the doctor office visits data.

3.5 Robust survival boosting with accelerated failure time model

Cox regression in survival analysis is based on a partial likelihood function. A robust extension in the CC-family is beyond the scope of this article. Alternatively, one may apply robust survival regression with the accelerated failure time model in **irboost**. The following code provides robust survival analysis for patients with advanced lung cancer from the North Central Cancer Treatment Group. Model performance is evaluated with multiple measures for survival data. Figure 12 illustrates the robustness weights using IRBoost.

```
library("survival")
lung1 <- lung[complete.cases(lung), ]  # remove missing data
y_upper_bound <- rep(NA, dim(lung1)[1])
# set right-censoring obs. to y_upper_bound = +Inf
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"
}
x <- as.matrix(lung1[, !names(lung1) %in% c("time", "status")])  # predictors
dtrain <- xgboost::xgb.DMatrix(data = x, label_lower_bound = lung1$time,
    label_upper_bound = y_upper_bound)  # input data format
param <- list(objective = "survival:aft", eval_metric = "aft-nloglik",
    aft_loss_distribution = "normal", aft_loss_distribution_scale = 1.2,
    max_depth = 3)
library("Hmisc")
fit7 <- irb.train(params = param, data = dtrain, cfun = "hcave",
    s = 3, nrounds = 50)
# evaluate model prediction accuracy
Hmisc::rcorr.cens(predict(fit7, newdata = dtrain), Surv(time = lung1$time,
    event = 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
```

```
plot(fit7$weight_update, ylab = "Weight")  # plot robustness weights
```

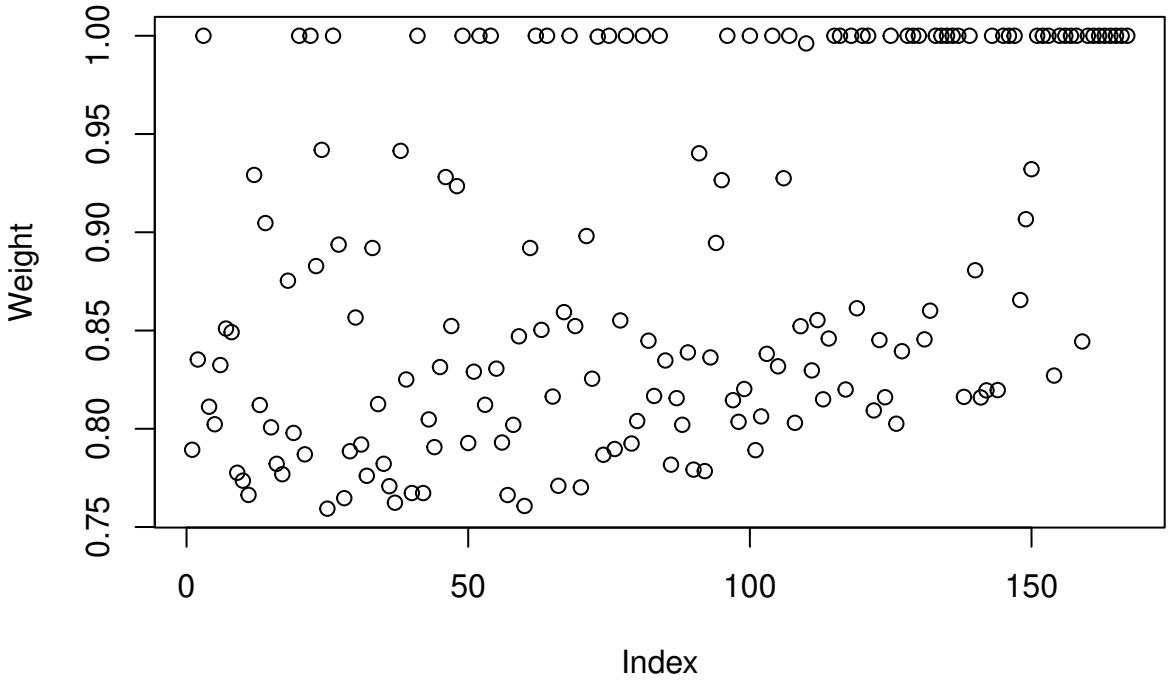

Figure 12: Robustness weights of IRBoost for the lung cancer data.

Heritier, Stephane, Eva Cantoni, Samuel Copt, and Maria-Pia Victoria- Feser. 2009. *Robust Methods in Biostatistics*. Vol. 825. John Wiley & Sons.

Long, Philip M, and Rocco A Servedio. 2010. "Random Classification Noise Defeats All Convex Potential Boosters." *Machine Learning* 78 (3): 287–304.

Wang, Zhu. 2021. "Unified Robust Boosting." *arXiv Preprint arXiv:2101.07718*.

———. 2024. "Unified robust estimation." *Australian & New Zealand Journal of Statistics* 66 (1): 77–102.