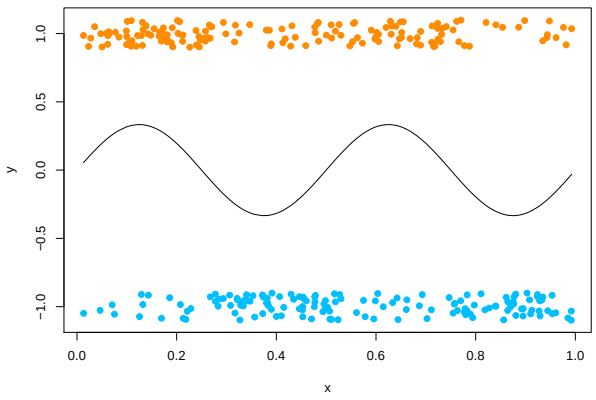
Boosting

Rishabh Vaish

AdaBoost with stump model

Let's write our own code for a one-dimensional AdaBoost using a tree stump model as the weak learner.

```
set.seed(1)
n = 300
x = runif(n)
py <- function(x)
  sin(4 * pi * x) / 3 + 0.5
y = (rbinom(n, 1, py(x)) - 0.5) * 2
plot(
  x,
  y + 0.1 * runif(n, -1, 1),
  ylim = c(-1.1, 1.1),
  pch = 19,
  col = ifelse(y == 1, "darkorange", "deepskyblue"),
  ylab = "y"
)
lines(sort(x), py(x)[order(x)] - 0.5)
```




```
testx = seq(0, 1, length.out = 1000)
testy = (rbinom(1000, 1, py(testx)) - 0.5) * 2
```

- The stump model is a CART model with just one split, hence two terminal nodes. Since we consider just one predictor, the only thing that needs to be searched in this tree model is the cutting point. Write a function to fit the stump model with subject weights:
  - **Input**: A set of data $\mathcal{D}_n = \{x_i, y_i, w_i\}_{i=1}^n$

- **Output**: The cutting point c, and node predictions $f_L, f_R \in \{-1, 1\}$
- **Step 1**: Search for a splitting rule $\mathbf{1}(x \leq c)$ that will maximize the weighted reduction of Gini impurity.

$$\texttt{score} = -\frac{\sum_{\mathcal{T}_L} w_i}{\sum w_i}\text{Gini}(\mathcal{T}_L) - \frac{\sum_{\mathcal{T}_R} w_i}{\sum w_i}\text{Gini}(\mathcal{T}_R),$$

  where, for given data in a potential node $\mathcal{T}$, the weighted version of Gini is

$$\text{Gini}(\mathcal{T}) = \widehat{p}(1-\widehat{p}), \qquad \widehat{p} = (\sum w_i)^{-1} \sum w_i I(y_i = 1).$$

- **Step 2**: Calculate the left and the right node predictions $f_L, f_R \in \{-1, 1\}$ respectively.

```r
# A function to make stump tree using X, Y, W as input
stump <- function(x, y, w) {
  max_score <- -Inf
  cut <- 0
  n = length(x)
  max_x <- max(x)
  min_x <- min(x)
  for (i in 1:n) {
    #checking for edge case
    if (x[i] != max_x && x[i] != min_x) {
      #seperate left and right
      y_left <- y[which(x <= x[i])]
      y_right <- y[which(x > x[i])]
      w_left <- w[which(x <= x[i])]
      w_right <- w[which(x > x[i])]

      #calculate gini impurity
      p_left <- sum(w_left[which(y_left == 1)]) / sum(w_left)
      gini_left <- p_left * (1 - p_left)
      p_right <- sum(w_right[which(y_right == 1)]) / sum(w_right)
      gini_right <- p_right * (1 - p_right)

      #get the weighted gini score
      score <-
        -((sum(w_left) / sum(w)) * gini_left) - ((sum(w_right) / sum(w)) * gini_right)

      #maximize the score
      if (score > max_score) {
        max_score = score
        cut = i
      }
    }
  }

  #decide the left and right tree
  pred_left <- sum(y[which(x <= x[cut])])
  pred_right <- sum(y[which(x > x[cut])])

  #make left and right predictions
  left_node <- ifelse(pred_left <= 0, -1, 1)
  right_node <- ifelse(pred_right <= 0, -1, 1)

  #return results, containing the cut, left predictiona nd right prediction
```

```
  results <- c(x[cut], left_node, right_node)
  return(results)
}
```

- Based on the AdaBoost algorithm, write your own code to fit the classification model, and perform the following
  - You are required to implement a `shrinkage` factor δ, which is commonly used in boosting algorithms.
  - You are not required to do bootstrapping for each tree (you still can if you want).
  - You should generate the following data to test your code and demonstrate that it is correct.
  - Plot the exponential loss $n^{-1}\sum_{i=1}\exp\{-y_i\delta\sum_k \alpha_k f_k(x_i)\}$ over the number of trees and comment on your findings.
  - Try a few different `shrinkage` factors and comment on your findings.
  - Plot the final model (funtional value of F, and also the sign) with the observed data.

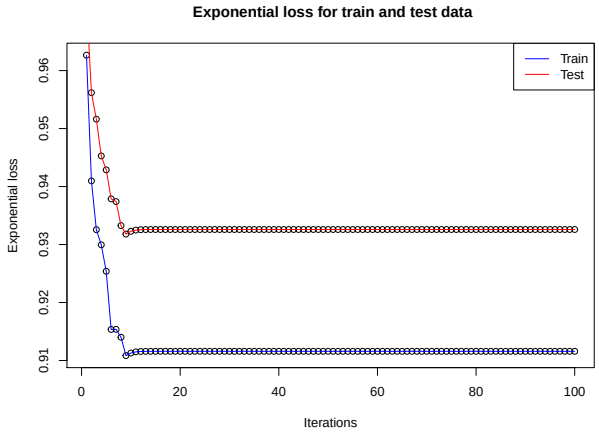
```
shrinkage <- 0.7 #shrinkage
iter <- 100 #iterations

# adaboost algorithm
adaboost <- function(x, y, shrinkage, iter) {
  #Initialise values
  w <- rep(1 / length(x), length(x)) #weights
  alpha = rep(0, iter) #alpha
  z <- rep(0, iter) #z values
  model <-
    matrix(0, nrow = iter, ncol = 3) #storing iter number of tree stumps
  epsilon <- rep(0, iter) # error rate for each tree
  for (i in 1:iter) {
    #create stump model
    model[i,] <- stump(x, y, w)
    #make predictions
    y_pred <- ifelse(x <= model[i, 1], model[i, 2], model[i, 3])
    #calculate the misclassification
    epsilon[i] <- sum(w[which(y_pred != y)])
    #reverse predictions if epsilon >0.5
    if (epsilon[i] > 0.5) {
      y_pred <- -y_pred
      model[i, 2] <- -model[i, 2]
      model[i, 3] <- -model[i, 3]
      epsilon[i] <- 1 - epsilon[i]
    }
    #calculate alpha
    alpha[i] <-
      0.5 * shrinkage * log((1 - epsilon[i]) / (epsilon[i]))
    #calculate z
    z[i] <- 2 * sqrt((1 - epsilon[i]) * (epsilon[i]))
    #update weights
    w <- (w / z[i]) * exp(-alpha[i] * y * y_pred)
  }
  results <- cbind(model, alpha)
  return(results)
}
```

```r
model <- adaboost(x, y, shrinkage, iter)
#making predictions on train and calculating exponential loss
y_pred_test <- rep(0, length(testx))
y_pred_train <- rep(0, length(x))
exp_loss_train <- rep(0, iter)
exp_loss_test <- rep(0, iter)

for (i in 1:iter) {
  #prediction
  y_pred_train <-
    y_pred_train + model[i, 4] * ifelse(x <= model[i, 1], model[i, 2], model[i, 3])
  #exponential loss
  exp_loss_train[i] <-
    (1 / length(x)) * sum(exp(-y * y_pred_train * shrinkage))

  y_pred_test <-
    y_pred_test + model[i, 4] * ifelse(testx <= model[i, 1], model[i, 2], model[i, 3])
  exp_loss_test[i] <-
    (1 / length(testx)) * sum(exp(-testy * y_pred_test * shrinkage))
}
#plot exponential loss on training data
plot(1:100,
     exp_loss_train,
     xlab = "Iterations",
     ylab = "Exponential loss",
     main = " Exponential loss for train and test data")
lines(exp_loss_train, col = "Blue")
#plot exponential loss on testing data
points(1:100, exp_loss_test)
lines(exp_loss_test, col = "Red")
legend(
  "topright",
  legend = c("Train", "Test"),
  col = c("Blue", "Red"),
  lty = 1
)
```



```r
#changing shrinkage value and storing test, train accuracy
shrinkage <-  seq(from = 0.1, to = 1, by = 0.1)
accuracy_test <-  rep(0, length(shrinkage))
accuracy_train <-  rep(0, length(shrinkage))

for (j in 1:length(shrinkage)) {
  y_pred_test <- rep(0, length(testx))
  y_pred_train <- rep(0, length(x))
  model <- adaboost(x, y, shrinkage[j], 100)

  for (i in 1:iter) {
    y_pred_train <-
      y_pred_train + model[i, 4] * ifelse(x <= model[i, 1], model[i, 2], model[i, 3])
    y_pred_test <-
      y_pred_test + model[i, 4] * ifelse(testx <= model[i, 1], model[i, 2], model[i, 3])
  }

  y_pred_train <- ifelse(y_pred_train >= 0, 1, -1)
  accuracy_train[j] <-
    (100 * (length(which(y_pred_train == y)) / length(y)))

  y_pred_test <- ifelse(y_pred_test >= 0, 1, -1)
  accuracy_test[j] <-
    (100 * (length(which(
      y_pred_test == testy
    )) / length(testy)))
}
#best shrinkage = 0.5
cbind(shrinkage = shrinkage, accuracy_test, accuracy_train)
```

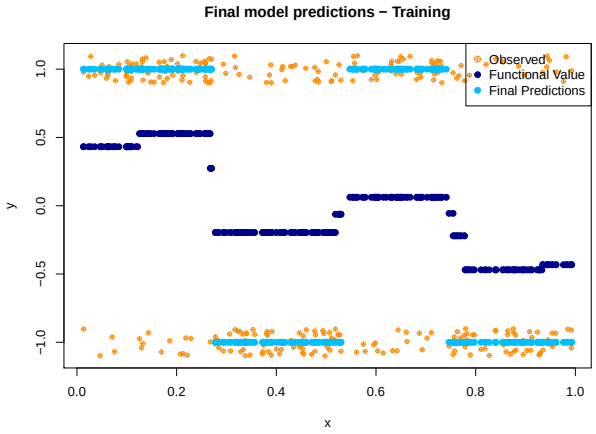
```
##       shrinkage accuracy_test accuracy_train
##  [1,]       0.1          62.6       65.66667
##  [2,]       0.2          62.6       65.66667
##  [3,]       0.3          62.6       65.66667
##  [4,]       0.4          62.6       65.66667
##  [5,]       0.5          70.1       73.66667
##  [6,]       0.6          69.6       73.66667
##  [7,]       0.7          69.6       73.66667
##  [8,]       0.8          69.6       73.66667
##  [9,]       0.9          70.0       73.00000
## [10,]       1.0          70.0       73.00000
```

```r
# final model
shrinkage = 0.5
model <- adaboost(x, y, shrinkage, iter)
# predictions for test and train
y_pred_test <- rep(0, length(testx))
y_pred_train <- rep(0, length(x))

for (i in 1:iter) {
  y_pred_train <-
    y_pred_train + model[i, 4] * ifelse(x <= model[i, 1], model[i, 2], model[i, 3])
  y_pred_test <-
```

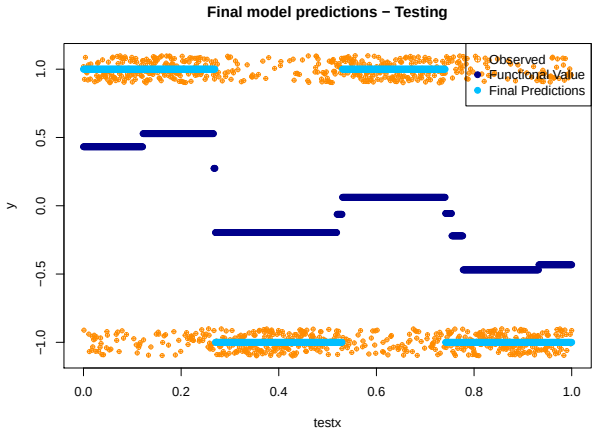
```
    y_pred_test + model[i, 4] * ifelse(testx <= model[i, 1], model[i, 2], model[i, 3])
}
# getting signs
y_pred_train_sign <- ifelse(y_pred_train >= 0, 1, -1)
y_pred_test_sign <- ifelse(y_pred_test >= 0, 1, -1)

#plotting the final model and sign - training
plot(
  x,
  y + 0.1 * runif(n, -1, 1),
  ylim = c(-1.1, 1.1),
  pch = 10,
  cex = 0.7,
  col = "darkorange",
  ylab = "y",
  main = "Final model predictions - Training"
)
points(x, y_pred_train, col = "darkblue", pch = 19)
points(x, y_pred_train_sign, col =  "deepskyblue", pch = 19)
legend(
  "topright",
  legend = c("Observed", "Functional Value", "Final Predictions"),
  col = c("darkorange", "darkblue", "deepskyblue"),
  pch = c(10, 19, 19)
)
```




```
#plotting the final model and sign - testing
plot(
  testx,
  testy + 0.1 * runif(n, -1, 1),
  ylim = c(-1.1, 1.1),
  pch = 10,
  cex = 0.7,
  col = "darkorange",
  ylab = "y",
  main = "Final model predictions - Testing"
)
```

```
points(testx, y_pred_test, col = "darkblue", pch = 19)
points(testx, y_pred_test_sign, col =  "deepskyblue", pch = 19)
legend(
  "topright",
  legend = c("Observed", "Functional Value", "Final Predictions"),
  col = c("darkorange", "darkblue", "deepskyblue"),
  pch = c(10, 19, 19)
)
```



Low shrinkage(0-0.2) reduces both test and train accuracy. High shrinkage(0.8-1) gives better results for both test and train data. Although the optimum value that we get for this data is 0.5, both train and test error are maximized at this point.