STAT 542 / CS 598: Homework 7

Fall 2019, by John Moran (jfmoran2)

Due: Monday, Nov 25 by 11:59 PM Pacific Time

Contents

Question 1 [100 Points] AdaBoost with stump model

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

- 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

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

```r
gini <- function(x, y, w) {
  n = length(x)
  cut.index <- 0
  best.score <- -1e10

  weights.total <- sum(w)

  gscores <- list(n)

  for (i in 1:(n-1)) {
    left.indices <- seq(1,i)
    right.indices <- seq((i+1),n)

    left.sum.total.weights <- sum(w[left.indices])
    right.sum.total.weights <- sum(w[right.indices])

    left.sum.match.weights <-  sum(w[which(y[left.indices]==1)])
    right.sum.match.weights <- sum(w[which(y[right.indices]==1)+i])

    left.tmp <- left.sum.match.weights/left.sum.total.weights
    gini.left <- left.tmp*(1-left.tmp)

    right.tmp <- right.sum.match.weights/right.sum.total.weights
    gini.right <- right.tmp*(1-right.tmp)
    score <- -(left.sum.total.weights/weights.total)*gini.left -
      (right.sum.total.weights/weights.total)*gini.right

    gscores[i] <- score

    if (score > best.score) {
      cut.index <- i
      best.score <- score
    }

  }

  left.plus.weights <-  sum(w[which(y[1:cut.index] ==  1)])
  left.minus.weights <- sum(w[which(y[1:cut.index] == -1)])

  right.plus.weights <-  sum(w[which(y[(cut.index+1):n] ==  1) + cut.index])
  right.minus.weights <- sum(w[which(y[(cut.index+1):n] == -1) + cut.index])

  if (left.plus.weights > left.minus.weights) {
    left <- 1
  } else {
    left <- -1
  }

  if (right.plus.weights > right.minus.weights) {
    right <- 1
  } else {
    right <- -1
  }

  results <- list()

  results$cut.index <- cut.index
  results$left <- left
  results$right <- right
  results$gscores <- gscores

  return(results)
}


weak.classifier <- function(x, y, w, classifier) {
  n = length(x)
  g <- vector(mode="integer", length=n)
  cut <- 1
  left <- 1
  right <- -1

  if (classifier == "random") {
    pos <- sample(1:(n-1),1)
    g <- as.vector(c(rep(1, pos), rep(-1, n-pos)))
    cut <- pos
  } else if (classifier == "gini") {
    gini.results <- gini(x=x, y=y, w=w)

    g[1:gini.results$cut.index] <- gini.results$left
    g[(gini.results$cut.index+1):n] <- gini.results$right
    cut <- gini.results$cut.index

    left <- gini.results$left
    right <- gini.results$right
  }

  results <- list()
  results$cut <- cut
  results$g <- g
  results$left <- left
  results$right <- right

  return(results)
}
```

NOTE: In addition to the Gini option, I wrote a wrapper called *weak.classifier* that I could pass in "gini" or "random" for the classifier type so I could compare my Gini classifier with a random one.

- **Step 2**: Calculate the left and the right node predictions $f_L, f_R \in \{-1, 1\}$ respectively.
- 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 $n^{-1} \sum_{i=1} \exp\{-y_i \delta \sum_k \alpha_k f_k(x_i)\}$
  - 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.

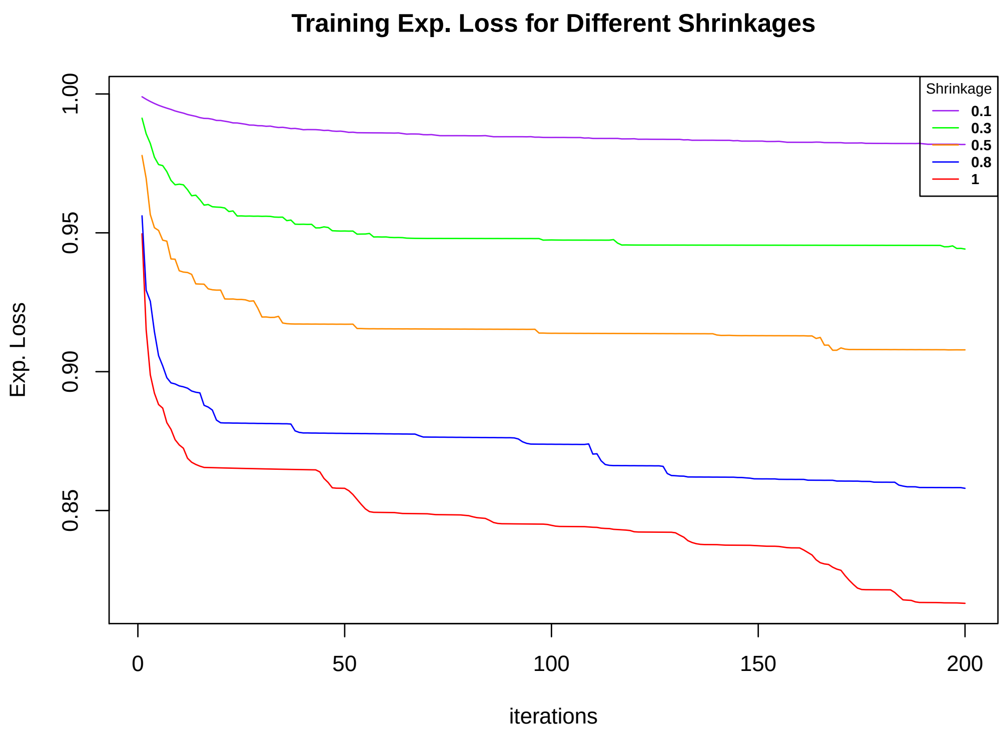
```
adaboost <- function (x, y, w, G, shrinkage) {
  class.results <- weak.classifier(x=x, y=y, w=w, "gini")

  g <- class.results$g
  cut <- class.results$cut
  left <- class.results$left
  right <- class.results$right

  err <- sum((1-y*g)*w)/2

  alpha <- (1/2)*log((1-err)/err)

  G <- G + shrinkage*alpha*g
  w1 <- w*exp(-shrinkage*alpha*y*g)
  w1 <- w1 / sum(w1)

  results <- list()
  results$G <- G
  results$w1 <- w1
  results$w <- w
  results$err <- min(err, 1-err)
```

```r
  results$a <- alpha
  results$cutvalue <- x[cut]
  results$left <- left
  results$right <- right

  return(results)
}

#main program
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
w <- rep(1,n)/n

# sort the data
m <- as.data.frame(cbind(x, y), col=2)
m <- m[order(m$x),]
x <- m$x
y <- m$y

T <- 200
shrinkages <- c(1.0, 0.8, 0.5, 0.3, 0.1)
s.len <- length(shrinkages)

exploss <- matrix(rep(0, s.len*T), nrow=s.len)
err <- matrix(rep(0, s.len*T), nrow=s.len)
G = rep(0,n)
alpha <- matrix(rep(0, s.len*T), nrow=s.len)
g.left <- matrix(rep(0, s.len*T), nrow=s.len)
g.right <- matrix(rep(0, s.len*T), nrow=s.len)
g.cut <- matrix(rep(0, s.len*T), nrow=s.len)

train.accuracy <- rep(0, s.len)
train.finalG <- matrix(rep(0, s.len*n), nrow=s.len)

for (j in 1:s.len) {
  shrinkage <- shrinkages[j]
  G = rep(0,n)
  w <- rep(1,n)/n
  myout <- adaboost(x=x, y=y, w=w, G=G, shrinkage=shrinkage)
  err[j,1] <- sum(sign(myout$G) != y)/n
  exploss[j,1] <- sum(exp(-shrinkage*y*myout$G))/n
  alpha[j,1] <- myout$a
  g.cut[j,1] <- myout$cutvalue
  g.left[j,1] <- myout$left
  g.right[j,1] <- myout$right


  for (i in 2:T) {
    myout <- adaboost(x=x, y=y, w=myout$w1, G=myout$G, shrinkage=shrinkage)
    err[j,i] <- sum(sign(myout$G) != y)/n
    exploss[j,i] <- sum(exp(-shrinkage*y*myout$G))/n
    alpha[j,i] <- myout$a
    g.cut[j,i] <- myout$cutvalue
    g.left[j,i] <- myout$left
    g.right[j,i] <- myout$right
  }

  train.finalG[j,] <- myout$G
  train.accuracy[j] <- sum(sign(train.finalG[j,]) == y)/n
  print(sprintf("Training accuracy: %f at shrinkage: %f", train.accuracy[j], shrinkage))
}
```

```
## [1] "Training accuracy: 0.750000 at shrinkage: 1.000000"
## [1] "Training accuracy: 0.740000 at shrinkage: 0.800000"
## [1] "Training accuracy: 0.740000 at shrinkage: 0.500000"
## [1] "Training accuracy: 0.736667 at shrinkage: 0.300000"
## [1] "Training accuracy: 0.736667 at shrinkage: 0.100000"
```

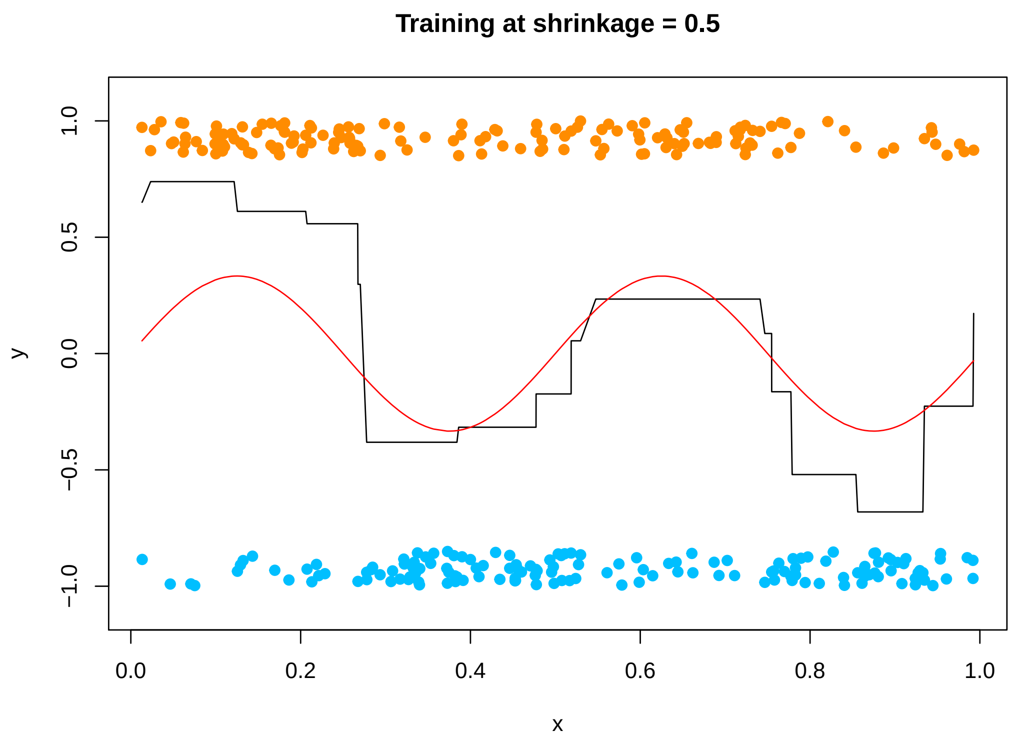
The above code tries different shrinkage values: 1.0, 0.8, 0.5, 0.3, 0.1 and the graph of the exponential loss is show below:

```
my.colors <- rainbow(s.len)
my.colors <- c("red", "blue", "darkorange", "green", "purple")
plot(c(1,T), c(min(exploss), max(exploss)), type="n", xlab="iterations", ylab="Exp. Loss",
     main="Training Exp. Loss for Different Shrinkages")
for (j in 1:s.len) {
  lines(exploss[j,], col=my.colors[j])
}

legend( x="topright",
        legend=shrinkages[s.len:1],
        col=my.colors[s.len:1],
        lty=1,
        title="Shrinkage",
        text.font=2,
        cex = 0.65)
```



The shrinkage scales the contribution of each tree by a factor equal to the shrinkage, where $\mathbf{0} < \textbf{shrinkage} < \mathbf{1}$. Smaller shrinkage values implies a slower learning rate. With no shrinkage (i.e. shrinkage $= 1$) the Adaboost algorithm can be prone to overfitting the training data.

```
plot(x, y + ifelse(y == 1, runif(300, -0.15, 0), runif(n, 0, 0.15)), ylim = c(min(-1.1),
     max(1.1)), pch = 19,
     col = ifelse(y == 1, "darkorange", "deepskyblue"), ylab = "y",
     main="Training at shrinkage = 0.5")
lines(x, train.finalG[3,])
lines(x, py(x)-.5, col="red")
```



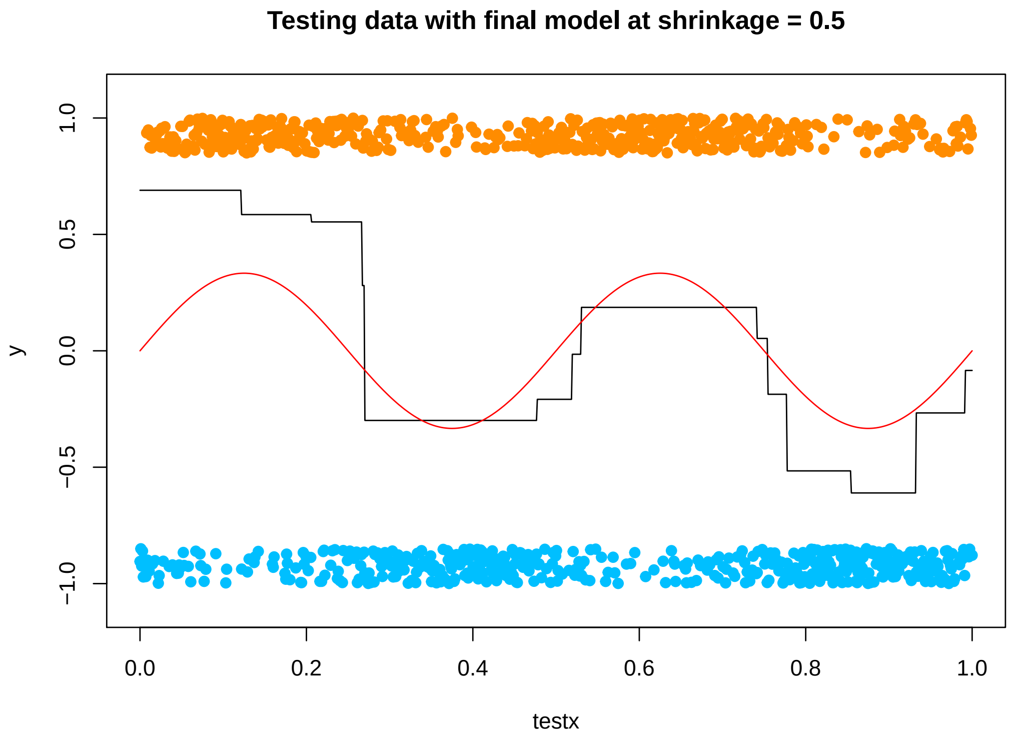
The above graph shows the training y values and actual y predictions when shrinkage = 0.5. Shrinkage of 0.5 was chosen after looking at the accuracy/misclassification rates on the test data below.

```
pred <- function(x, alpha, shrinkage, cut, left, right, T) {
  pred.sum <- 0

  for (i in 1:T) {
    if (x <= cut[i]) {
      pred.sum <- pred.sum + left[i]*shrinkage*alpha[i]
    } else {
      pred.sum <- pred.sum + right[i]*shrinkage*alpha[i]
    }
  }

  return(pred.sum)
}


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

t.pred <- matrix(rep(0, s.len*length(testx)), nrow=s.len)
t.pred.sign <- matrix(rep(0, s.len*length(testx)), nrow=s.len)
test.accuracy <- rep(0, s.len)

for (j in 1:s.len) {
  t.alpha <- alpha[j,]
  t.g.cut <- g.cut[j,]
  t.g.left <- g.left[j,]
  t.g.right <- g.right[j,]
  for (i in 1:length(testx)) {
    t.pred[j,i] <- pred(testx[i], t.alpha, shrinkages[j], t.g.cut, t.g.left, t.g.right, T)
  }

  t.pred.sign[j,] <- sign(t.pred[j,])
  test.accuracy[j] <- sum(t.pred.sign[j,] == testy)/length(testx)
  print(sprintf("Testing accuracy is %f at shrinkage: %f", test.accuracy[j],
                shrinkages[j]))
}
```

```
## [1] "Testing accuracy is 0.690000 at shrinkage: 1.000000"
## [1] "Testing accuracy is 0.695000 at shrinkage: 0.800000"
## [1] "Testing accuracy is 0.695000 at shrinkage: 0.500000"
## [1] "Testing accuracy is 0.693000 at shrinkage: 0.300000"
## [1] "Testing accuracy is 0.693000 at shrinkage: 0.100000"
```

The graph below shows the testy values and actual y predictions when shrinkage = 0.5 using the final model.

```
plot(testx, testy + ifelse(testy == 1, runif(1000, -0.15, 0), runif(n, 0, 0.15)), ylim = c(min(-1.1), ma
     col = ifelse(testy == 1, "darkorange", "deepskyblue"), ylab = "y",
     main="Testing data with final model at shrinkage = 0.5")

lines(testx, t.pred[4,])
lines(testx, py(testx)-.5, col="red")
```



At shrinkage of 0.5, the training accuracy was 0.740 and the final model testing accuracy was 0.695.