STAT 542 / CS 598: Homework 7

Fall 2019, by Regan Chan (ttchan2)

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

Contents

Directions

Students are encouraged to work together on homework. However, sharing, copying, or providing any part of a homework solution or code is an infraction of the University's rules on Academic Integrity. Any violation will be punished as severely as possible. Final submissions must be uploaded to your homework submission portal. No email or hardcopy will be accepted. For **late submission policy and grading rubrics**, please refer to the course website.

- You are required to submit two files:
  - Your `.rmd` RMarkdown (or Python) file, which should be saved as `HW7_yourNetID.Rmd`. For example, `HW7_rqzhu.Rmd`.
  - The result of knitting your RMarkdown file as `HW7_yourNetID.pdf`. For example, `HW7_rqzhu.pdf`. Please note that this must be a `.pdf` file. `.html` format cannot be accepted.
- Your resulting `.pdf` file will be considered as a report, which is the material that will determine the majority of your grade. Be sure to visibly include all `R` code and output that is relevant to answering the exercises.
- If you use the example homework `.Rmd` file (provided here) as a template, be sure to remove the directions section.
- Your `.Rmd` file should be written such that, if it is placed in a folder with any data you are asked to import, it will knit properly without modification.
- Include your Name and NetID in your report.
- **Late policy**: You can also choose to submit your assignment as late as five days after the deadline, which is 11:59 PM on Saturday. However, **you will lose 10% of your score**.

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.

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

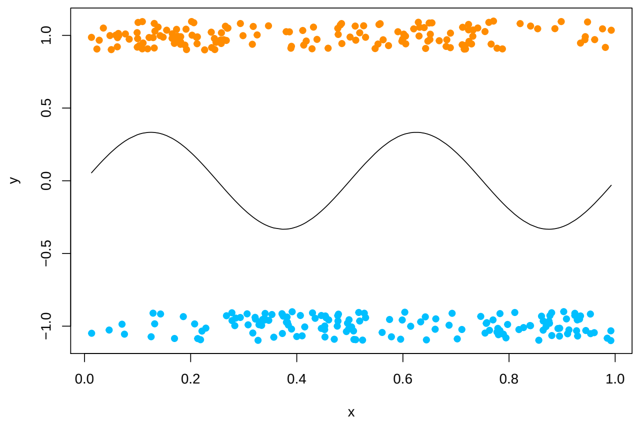
    where, for given data in a potential node $\mathcal{T}$, the weighted version of Gini is

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

  - **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 $n^{-1} \sum_{i=1} \exp\{-y_i \delta \sum_k \alpha_k f_k(x_i)\}$ 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.

```
  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
```

Below is the code for adaboost classification:

```
setClass(
  "Stump",
  slots=c(c="numeric", left="numeric", right="numeric"),
)

setGeneric("classify", function(.Object, newx) standardGeneric("classify"))
```

```
## [1] "classify"
```

```
setMethod("classify", "Stump", function(.Object, newx) {
  ifelse(newx < .Object@c, .Object@left, .Object@right)
})

Stump <- function(x, y, w) {
  n <- length(y)

  # optimized split criteria calculation by first sorting x-values
  # then use dynamic programming to update summations
  split_scores_dp <- function(x, y, w) {
    sum_w <- sum(w)
    left.sum_w <- 0
    left.sum_wy <- 0
    right.sum_w <- sum_w
    right.sum_wy <- as.numeric(w %*% (y==1))

    x_order <- order(x)
    scores <- vector("numeric", n-1)
    scores[x_order] <- mapply(function(y_i, w_i){
      wy <- w_i * (y_i==1)
      left.sum_w <<- left.sum_w + w_i
      left.sum_wy <<- left.sum_wy + wy
      right.sum_w <<- right.sum_w - w_i
      right.sum_wy <<- right.sum_wy - wy
      (left.sum_w * gini_dp(left.sum_wy, left.sum_w) +
       right.sum_w * gini_dp(right.sum_wy, right.sum_w)) / -sum_w
    }, y[x_order], w[x_order])

    return(scores)
  }

  gini_dp <- function(sum_wy, sum_w) {
    p <- sum_wy / sum_w
    p * (1-p)
  }

  weighted_majority <- function(cond) {
    pos_cond <- cond && y == 1
    neg_cond <- cond && y == -1
    ifelse(sum(w[pos_cond]) > sum(w[neg_cond]), 1, -1)
  }
  simple_majority <- function(cond) {
    ifelse(sum(y[cond]==1) > sum(y[cond]!=1), 1, -1)
  }
  scores <- split_scores_dp(x, y, w)
  best_c <- x[which.max(scores)]
  new("Stump", c=best_c, left=weighted_majority(x <= best_c), right=weighted_majority(x > best_c))
}

setClass(
  "AdaboostModel",
  slots=c(a="numeric", c="list", exponential_loss="numeric"),
)

setGeneric("F", function(.Object, newx) standardGeneric("F"))
```

```
## [1] "F"
```

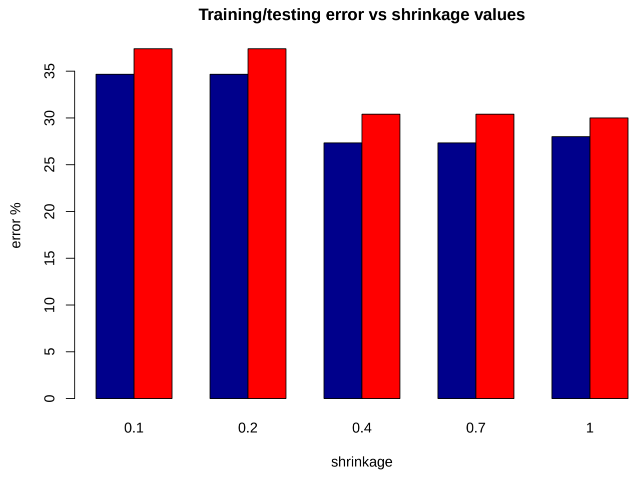
```
setMethod("F", "AdaboostModel", function(.Object, newx){
  rowSums(mapply(function(a_t, c_t){
    a_t * classify(c_t, newx)
  }, .Object@a, .Object@c))
})
```

```r
setMethod("classify", "AdaboostModel", function(.Object, newx) {
  ifelse(F(.Object, newx) > 0, 1, -1)
})

adaBoost <- function(x, y, T, shrinkage) {
  n <- length(y)
  w <- rep(1/n, n)
  classifiers <- vector("list", T)
  loss <- y * 0
  exponential_loss <- vector("numeric", T)

  a <- sapply(1:T, function(k) {
    c <- Stump(x, y, w)
    classifiers[[k]] <<- c

    f_k <- classify(c, x)
    e_k <- sum(w * (f_k != y))
    a_k <- log((1-e_k)/e_k) / 2
    w_new <- w * exp(-a_k * y * shrinkage * f_k)
    w <<- w_new / sum(w_new)

    # exponential loss
    loss <<- loss - y * shrinkage * (a_k * f_k)
    exponential_loss[k] <<- mean(exp(loss))

    return(a_k)
  })

  new("AdaboostModel", a=a, c=classifiers, exponential_loss=exponential_loss)
}
```

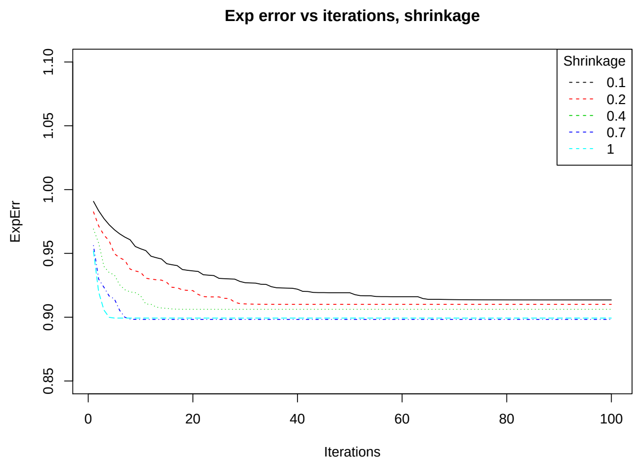
Testing different shrinkage values of δ

Conclusion: A higher shrinkage value δ converges faster as well as achieving lower errors; at 1.0 it converges to a suboptimal training accuracy but actually gave a better testing accuracy, compared to 0.7 which seem to be a good balance.

```r
shrinkageValues <- c(0.1, 0.2, 0.4, 0.7, 1.0)
shrinkageErrors <- NULL
shrinkageExpErr <- NULL
for(k in 1:length(shrinkageValues)) {
  shrinkage <- shrinkageValues[k]
  model <- adaBoost(x, y, 100, shrinkage)
  pred <- classify(model, x)
  trainErr <- sum(pred != y) / length(y) * 100
  pred <- classify(model, testx)
  testErr <- sum(pred != testy) / length(testy) * 100
  shrinkageErrors <- cbind(shrinkageErrors, c(trainErr, testErr))
  shrinkageExpErr <- cbind(shrinkageExpErr, model@exponential_loss)
}

colnames(shrinkageErrors) <- shrinkageValues
barplot(shrinkageErrors, main="Training/testing error vs shrinkage values",
        xlab="shrinkage", ylab="error %", col=c("darkblue","red"), beside=TRUE)
legend(x=10, y=45, legend=c("training", "test"), col=c("darkblue", "red"))
```

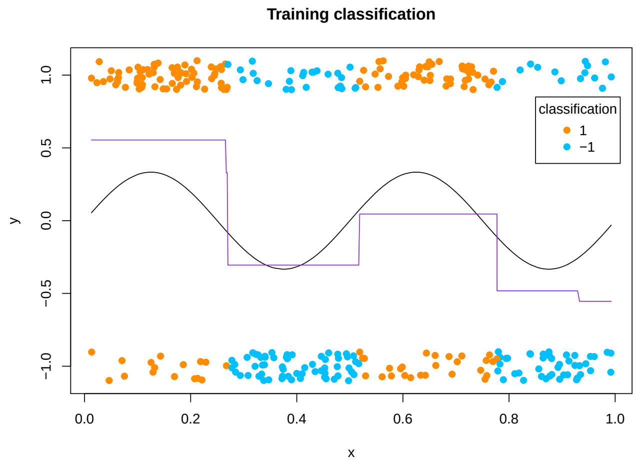


```r
matplot(shrinkageExpErr, type="l", ylim=c(0.85, 1.1), col=1:5, pch=1:5,
        main="Exp error vs iterations, shrinkage", xlab="Iterations", ylab="ExpErr")
legend(x="topright", lty=2, legend=shrinkageValues, col=1:5, title="Shrinkage")
```



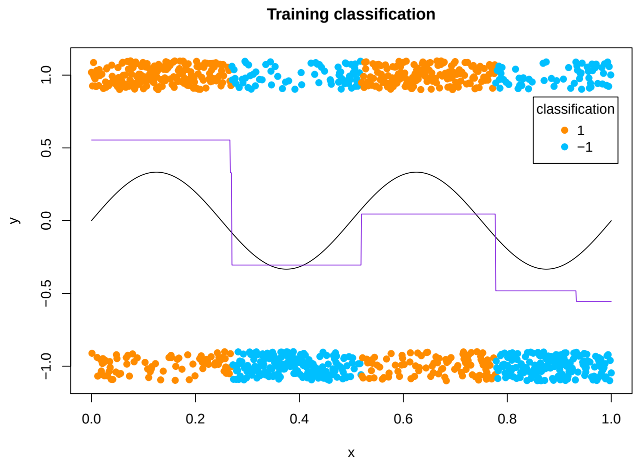
Final classification models

```r
model <- adaBoost(x, y, 50, 1.0)

plotAda <- function(x, y) {
  n <- length(x)
  output.train <- F(model, x)
  pred.train <- classify(model, x)
  plot(x, y + 0.1*runif(n, -1, 1), ylim=c(-1.1, 1.1), pch=19,
       col=ifelse(pred.train==1, "darkorange", "deepskyblue"),
       ylab="y", main="Training classification")
  lines(sort(x), py(x)[order(x)]-0.5)
  lines(sort(x), output.train[order(x)], type="l", col=c("blueviolet"))
  legend(x=0.85, y=0.85, legend=c(1, -1), col=c("darkorange", "deepskyblue"),
         pch=19, title="classification")
  return(mean(pred.train==y))
}

paste("Training error: ", plotAda(x, y))
```



```
## [1] "Training error:  0.72"
```

```
paste("Test error: ", plotAda(testx, testy))
```



```
## [1] "Test error:  0.7"
```