AdaBoost

9.1 Exponential loss vs misclassification loss

In boosting, we label (for convenience) the classes as +1 and −1, respectively. We furthermore let our classifier \( x \) be on the form \( \hat{y}(x) = \text{sign}(C(x)) \), i.e., a thresholding at 0 of some real-valued function \( C(x) \). Assume that we have learned some function \( \hat{C}(x) \) from training data such that we can make predictions \( \hat{Y} = \hat{G}(x) = \text{sign}(\hat{C}(x)) \). Fill in the missing columns in the table below:

<table>
<thead>
<tr>
<th>( \hat{C}(x_*) )</th>
<th>( \hat{y}_* )</th>
<th>Exponential loss ( \exp(-y_<em>\hat{C}(x_</em>) )</th>
<th>Misclassification loss ( I(y_* \neq \hat{y}_*) )</th>
<th>( y_* )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3</td>
<td></td>
<td></td>
<td></td>
<td>−1</td>
</tr>
<tr>
<td>−0.2</td>
<td></td>
<td></td>
<td></td>
<td>−1</td>
</tr>
<tr>
<td>1.5</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>−4.3</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

In what sense is the exponential loss more informative than the misclassification loss, and how can that information be used when training a classifier?

9.2 AdaBoost for spam classification

Consider the very same setting data set for the data set \texttt{email.data} as in problem 8.2, but now use AdaBoost instead. AdaBoost is available in R as the command \texttt{boosting()} in the \texttt{adabag} package, and is used similarly to the \texttt{lda} command. Call \texttt{boosting()} with the argument \texttt{boos=FALSE} in order to get the AdaBoost classifier as presented in the lecture notes. What test error do you achieve?

9.3 Exploring the AdaBoost algorithm

The R script file \texttt{adaboost_illustration.R}, provided as a separate file and at the very end of this document, illustrates the AdaBoost algorithm in the same way as was done in lecture 7. Familiarize yourself with the code and compare it to the pseudocode given in the course literature (Chapter 5 in the lecture notes). Explore what happens if you make changes in the input data and the number of trees (stumps), \( B \), used!
9.4 Deriving the AdaBoost weights

The AdaBoost classifier can be written as $\hat{y}_{\text{boost}}(x) = \text{sign}(C^B(x))$ where the functions $C^1(x), \ldots, C^B(x)$ are constructed sequentially as

$$C^b(x) = C^{b-1}(x) + \alpha_b \hat{y}_b(x),$$

initialized with $C^0(x) \equiv 0$. The $b$th ensemble member $\hat{y}_b(x)$ is found by applying the chosen base classifier to a weighted version of the training data. Once this has been found, we also need to compute the corresponding “confidence” coefficient $\alpha_b$. This is done by minimizing the weighted exponential loss of the training data,

$$\alpha_b = \arg\min_{\alpha} \left\{ \sum_{i=1}^{N} w_i^b \exp(-\alpha y_i \hat{y}_b(x_i)) \right\}$$

(9.1)

where $w_i^b = \exp(-y_i C^{b-1}(x_i))$.

Show that the optimal solution is given by

$$\alpha_b = \frac{1}{2} \log \left( \frac{1 - \bar{err}_b}{\bar{err}_b} \right)$$

where $\bar{err}_b = \frac{\sum_{i=1}^{N} w_i^b I(y_i \neq \hat{y}_b(x_i))}{\sum_{i=1}^{N} w_i^b}$.

Hint 1: We use class labels $-1$ and $1$, i.e. $y_i \in \{-1,1\}$ and $\hat{y}_b(x_i) \in \{-1,1\}$. Using this fact, divide the sum in the objective function in (9.1) into one sum ranging over all correctly classified training data points and one sum ranging over all misclassified training data points.

Hint 2: You are allowed to use the fact that the objective function in (9.1) has a single stationary point corresponding to the global minima.

9.5 Gradient boosting in R

Explore gradient boosting by using the xgboost package on the spam data set email.data.
Solutions

9.1

<table>
<thead>
<tr>
<th>( \hat{C}(x) )</th>
<th>( \hat{y} )</th>
<th>Exponential loss ( \exp(-y \hat{C}(x)) )</th>
<th>Misclassification loss ( I(y \neq \hat{y}) )</th>
<th>( y )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3</td>
<td>1</td>
<td>( \exp(0.3) \approx 1.35 )</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>-0.2</td>
<td>-1</td>
<td>( \exp(-0.2) \approx 0.82 )</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>1.5</td>
<td>1</td>
<td>( \exp(-1.5) \approx 0.22 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>-4.3</td>
<td>-1</td>
<td>( \exp(4.3) \approx 73.70 )</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Whereas the misclassification loss only gives the information “correct” or “misclassified”, the exponential loss contains the same information (< 1: correct, > 1: misclassified) as well as information about the margin \( y \hat{C}(x) \), which in a sense gives an idea of “how much” wrong or correct the classifier is.

9.2 With AdaBoost, we get a test error of 4.5%

```r
library(adabag)

# First run the code from problem 8.2 to load and split the data

# Use AdaBoost ...
boosting.fit <- boosting(formula=Class~.,data=email.train,boos=FALSE)

# ... and do prediction on test data
boosting.pred <- predict(boosting.fit, newdata=email.test)

# test error rate
boosting.test.error <- mean(email.test$Class != boosting.pred$class)
boosting.test.error
```

9.3 -
9.4 We want to minimize the expression

\[ E(\alpha) := \sum_{i=1}^{N} w_i^b \exp(-\alpha y_i \hat{y}^b(x_i)) \] (9.2)

with respect to \( \alpha \). Using the second hint of the problem formulation, we can do this by differentiating the expression, setting the derivative to zero, and solving for \( \alpha \).

Before we do this, however, we make use of the first hint of the problem formulation to simplify the expression under study. Note first that using the class labels \(-1\) and \(1\) implies that

\[ y_i \hat{y}^b(x_i) = \begin{cases} 
1 & \text{if } \hat{y}^b(x_i) = y_i, \\
-1 & \text{if } \hat{y}^b(x_i) \neq y_i.
\end{cases} \]

Thus, the expression (9.2) can be written as

\[ E(\alpha) = e^{-\alpha} \sum_{i=1}^{N} w_i^b I(\hat{y}^b(x_i) = y_i) + e^{\alpha} \sum_{i=1}^{N} w_i^b I(\hat{y}^b(x_i) \neq y_i), \]

where we have used the indicator function to split the sum into two sums: the first ranging over all correctly classified data points, and the second ranging over all erroneously classified data points. Furthermore, for notational simplicity we define \( W_c \) and \( W_e \) for the sum of weights of correctly classified data points and erroneously classified data points, respectively.

Now, we compute the derivative and set to zero:

\[
\frac{dE}{d\alpha} = -e^{-\alpha} W_c + e^{\alpha} W_e = 0 \iff e^{\alpha} W_e = e^{-\alpha} W_c \iff e^{2\alpha} = \frac{W_c}{W_e} \iff \alpha = \frac{1}{2} \log \left( \frac{W_c}{W_e} \right).
\]

It remains to write the solution on the format asked for in the problem formulation. With \( W := \sum_{i=1}^{N} w_i^b \) denoting the sum of all weights, we note first that \( W_c = W - W_e \) and thus,

\[
\alpha = \frac{1}{2} \log \left( \frac{W - W_e}{W_e} \right) = \frac{1}{2} \log \left( \frac{1 - W_c/W}{W_e/W} \right).
\]

Finally, noting that \( \text{err}_b = W_e/W \) completes the proof.

9.5 -
Code for illustrating AdaBoost

```r
# clear workspace
library(rpart)

# some data we are going to work with
X1 <- c(0.2358, 0.1252, 0.4278, 0.6398, 0.6767, 0.8733, 0.3648, 0.6336, 0.3433, 0.8410)
X2 <- c(0.1761, 0.4465, 0.7539, 0.9037, 0.7111, 0.8414, 0.6060, 0.5107, 0.1430, 0.1994)
Y <- as.factor(c(1, 1, 1, 1, 1, -1, -1, -1, -1, -1))

# note: the class labels are 1 and -1
thedata <- data.frame(X1, X2, Y)
n <- length(Y)

# plot the data
plot(X1[Y == 1], X2[Y == 1], pch = 4, col = "red", xlim = c(0, 1), ylim = c(0, 1), xlab = expression("X"[1]), ylab = expression("X"[2]))
points(X1[Y == -1], X2[Y == -1], pch = 16, col = "blue")

# halt the execution for a second, so you have time to look at the plot
Sys.sleep(2)

# define some variables we will fill out in the loop below
stumps <- list()  # we need a list to save the tree part objects
alpha <- c()  # a list of the boosting coefficients alpha
w <- rep(1/n, n)  # the boosting weights

B <- 3;
for (b in 1:B)
{
  # open a new plot with good size and labels
  plot(X1, X2, type = "n", xlim = c(0, 1), ylim = c(0, 1), xlab = expression("X"[1]), ylab = expression("X"[2]))

  # learn a stump, i.e., one-depth tree. Gini index is used by default in rpart.
  stumps[[b]] <- rpart(formula = Y ~ X1 + X2, data = thedata, method = "class", weights = w, control = rpart.control(maxdepth = 1, minsplit = 2))

  # extract some information from the obtained stump
  split.variable <- rownames(stumps[[b]]$splits)[1]
  split.value <- stumps[[b]]$splits[1, 4];
  split.ncat <- stumps[[b]]$splits[1, 2];

  # plot the stump
  if (split.variable == "X1")
  {
    # if the split is in the X1-direction
    polygon(x = c(0, split.value, split.value, 0), y = c(0, 0, 1, 1), col = rgb(max(split.ncat, 0), 0, max(-split.ncat, 0), .5), border = NA)
    polygon(x = c(1, split.value, split.value, 1), y = c(0, 0, 1, 1), col = rgb(max(-split.ncat, 0), 0, max(split.ncat, 0), .5), border = NA)
  } else
  {
    # if the split is in the X2-direction
    polygon(y = c(0, split.value, split.value, 0), x = c(0, 0, 1, 1), col = rgb(max(split.ncat, 0), 0, max(-split.ncat, 0), .5), border = NA)
    polygon(y = c(1, split.value, split.value, 1), x = c(0, 0, 1, 1), col = rgb(max(-split.ncat, 0), 0, max(split.ncat, 0), .5), border = NA)
  }

  # plot the weighted data (bigger points = bigger weights)
  points(X1[Y == 1], X2[Y == 1], pch = 4, cex = n*w[Y == 1]/sum(w), col = "red")
  points(X1[Y == -1], X2[Y == -1], pch = 16, cex = n*w[Y == -1]/sum(w), col = "blue")

  # make a prediction on our data
  stump.pred <- predict(stumps[[b]], newdata = thedata, type = "class")

  # Determine if the datapoints are correct or not
  correct <- (Y == stump.pred)

  # plot a red ring around the misclassified points
  points(X1[!correct], X2[!correct], pch = 0, cex = 2, col = "red")
  Sys.sleep(2)

  # Update the boosting weights w according to the AdaBoost algorithm
  W <- sum(w)  # Compute the sum of all weights
  We <- sum(w[!correct])  # Compute the sum of all weights corresponding to misclassified data points
  em <- We/W  # Compute em
  alpha[b] <- 1/2*log((1-em)/em)  # Compute the alpha-weights
}
```

\[
\begin{align*}
  \text{w[correct]} & \leftarrow \text{w[correct]} \times \exp(-\alpha[b]) \quad \# \text{Compute the omega-weights} \\
  \text{w[!correct]} & \leftarrow \text{w[!correct]} \times \exp(\alpha[b]) \\
\end{align*}
\]

\# Add everything together into a boosted classifier

```r
boosted.classifier <- function(X1, X2) {  
  # Wrap our classifier as a function
  predictions <- predict(stumps[[1]], newdata=data.frame(X1, X2), type="class")
  for (b in 2:B){
    predictions[b] <- predict(stumps[[b]], newdata=data.frame(X1, X2), type="class")
  }
  # Determine if the stumps are correct or not
  C <- sum(alpha[predictions==1]) - sum(alpha[predictions==-1])
  return(sign(C))
}
```

\# open a new plot window

```r
plot(X1, X2, type="n", xlim=c(0,1), ylim=c(0,1), main="Boosted classifier", xlab=expression("X"[1]), ylab=expression("X"[2]))
```

\# classify many points, and plot a colored square around each point

```r
res <- 0.2  # resolution of the squares
for (xs1 in seq(0,10,res)) {
  for (xs2 in seq(0,10,res)) {
    pred <- boosted.classifier(xs1, xs2)
    if (pred==1)
      {  
        polygon(x=c(xs1-res/2,xs1+res/2,xs1+res/2,xs1-res/2),
                y=c(xs2-res/2,xs2+res/2,xs2+res/2,xs2-res/2), col=rgb(1,0,0,0.5), border=NA)
      }
    else
      {  
        polygon(x=c(xs1-res/2,xs1+res/2,xs1+res/2,xs1-res/2),
                y=c(xs2-res/2,xs2+res/2,xs2+res/2,xs2-res/2), col=rgb(0,0,1,0.5), border=NA)
      }
  }
}
# plot the data
points(X1[Y==1], X2[Y==1], pch=4, col="red")
points(X1[Y==-1], X2[Y==-1], pch=16, col="blue")
```