Biostat 830 Homework 1

David (Daiwei) Zhang

Problem 1

We wan to minimize

$$EPE(f) = \int \int |y - f(x)| P(dy) P(dx),$$

so we find a f that minimizes

$$\begin{aligned} EPE(f) &= \int |y - f(x)| P(dy) \\ &= \int_{-\infty}^{f(x)} [f(x) - y] P(dy) + \int_{f(x)}^{\infty} [y - f(x)] P(dy) \end{aligned}$$

By Leibniz's Rule, we have

$$\begin{aligned} \frac{\partial}{\partial f} EPE &= \int_{-\infty}^{f(x)} 1 P(dy) + f(x) - f(x) + \int_{f(x)}^{\infty} (-1) P(dy) + f(x) - f(x) \\ &= \int_{-\infty}^{f(x)} P(dy) - \int_{f(x)}^{\infty} P(dy) \end{aligned}$$

Then when $\frac{\partial}{\partial f} EPE = 0$, we have

$$\int_{-\infty}^{f(x)} P(dy) = \int_{f(x)}^{\infty} P(dy),$$

that is, $f(x) = Median(Y|X = x)$.

Problem 2

For the least square estimator, we have

$$Y^{pred} = X^{pred} (X^T X)^{-1} X^T Y,$$

so

$$Y_i^{pred} = \sum_j w_{i,j} Y_j,$$

where

$$w_{i,j} = x_i^{pred} (X^T X)^{-1} x_i^T = x_i^{pred} x_i^T$$

assuming orthonormal design (i.e. $X^T X = I$), and x_i is the i^{th} row of X (and the same notation applies to x_i^{pred}). Since $\{x_i\}$ is an orthonormal basis, we can interpret that the weight of the j^{th} training response's effect on the i^{th} testing response is equal to the i^{th} testing predictor's j^{th} coefficient when written in the basis of the training predictors.

Problem 3

We find the EPE of the simple linear classifier by using bootstrap and compare the estimates to those given by cross validation.

```
# Generate data
gen.data <- function(){
  f<-function(x){
    return(0.2 + x - 0.5*x^2 + 0.1*x^3 - 0.5*x^4)
  }
  xv = seq(0,1,0.001)
  yv = f(xv)
  dx = runif(500)
  dy = runif(500)
  boundry = f(dx)
  label = (dy>boundry)+0
  x_value = dx
  y_value = dy + rnorm(length(dy),sd=0.1)
  training_data = cbind(y_value, x_value, label)
  return(training_data)
}
training_data <- gen.data()

# function for validate a single batch
# the parameters are training data, batch labels and the id of the test batch
validate_batch<-function(data, batch, test_batch_id){
  d = data[batch!=test_batch_id,]
  # fit the probit regression
  y = d[,1]
  x = d[,2]
  l = d[,3]
  fit = glm(l~y+x,family=binomial(link="probit"))
  beta = matrix(ncol=1, fit$coef)
  test_d = data[batch==test_batch_id,1:2]
  if(!is.null(dim(test_d))){
   test_d = matrix(ncol=3,cbind(rep(1,dim(test_d)[1]), test_d))
  }else{
    test_d = matrix(ncol=3,c(1,test_d))
  }
  pred = test_d%*%beta
  pred_label= (pred>=0)+0
  true_label = data[batch==test_batch_id,3]
  return(length(which(true_label!=pred_label)))
}

K_fold_CV<-function(data, K){
  N = dim(data)[1]
  batch = rep(1:K, ceiling(N/K))[1:N]
  total_error = sum(sapply(1:K, function(x) validate_batch(data,batch,x)))
  EPE = total_error/N
  return(EPE)
}

# Fit a probit model using the training sample
```

```
clas.lin <- function(test.samp, train.samp, k){
  n.test <- nrow(test.samp)
  y <- train.samp[,1]
  x <- train.samp[,2]
  label <- train.samp[,3]
  fit <- glm(label~y+x,family=binomial(link="probit"))
  # Make prediction on the testing sample
  beta <- matrix(ncol=1, fit$coef)
  test.samp.pred <- cbind(rep(1, n.test), test.samp[,1:2]) # Add a col of 1 to the left for intercept
  pred <- test.samp.pred %*% beta
  pred.label <- as.numeric(pred>=0)
  return(pred.label)
}


# Find EPE of a classifier by bootstrap
# fun: classifying function
# m: Number of bootstrap samples
# k: parameter for classifying function
epe.boot <- function(training_data, fun, k, m){
  n <- nrow(training_data)
  # Generate bootstrap sample indexes
  train.id.multi <- matrix(sample(1:n, n*m, replace=TRUE), nrow = m)
  stopifnot(min(table(train.id.multi)) > 0) # Make sure every data point in the data is selected in at
  n.test.total <- 0 # Total number of predictions made
  err.total <- 0 # Sum of errors
  for(i in 1:m){
    # Get training and testing ids
    train.id <- train.id.multi[i,]
    test.id <- setdiff(1:n, train.id)
    n.test <- length(test.id)
    stopifnot(n > 1)
    # Get training and testing samples
    train.samp <- training_data[train.id, ]
    test.samp <- training_data[test.id, ]
    pred.label <- fun(test.samp, train.samp, k)
    # Get the errors
    true.label <- test.samp[, 3]
    err.total <- err.total + sum(pred.label != true.label)
    n.test.total <- n.test.total + n.test
  }
  epe <- err.total / n.test.total
  return(epe)
}


## k-nearest neighbor classifier

vote <- function(target, td, K){
   dist = apply(td,1, function(x) (x[1]-target[1])^2+(x[2]-target[2])^2)
   # find the first k-ranked points
   index = which(rank(dist)<=K)
   rst = 1
   if(sum(td[index,3])<K/2){
     rst = 0
   }
   return(rst)
}

clas.vote <- function(target, training_data, K){
  est_rst <- apply(target, 1 , function(x) vote(x, training_data, K))
  return(est_rst)
}


get.vote.epe <- function(training_data, m, kk){
  clas.vote.epe.multi <- c()
  for(k in kk){
    clas.vote.epe <- epe.boot(training_data, clas.vote, k, m)
    clas.vote.epe.multi <- c(clas.vote.epe.multi, clas.vote.epe)
  }
  return(clas.vote.epe.multi)
}

# Compare cross validation and bootstrap estimations for epe
K_fold_CV(training_data,2)
```

```
## [1] 0.112
```

```
K_fold_CV(training_data,5)
```

```
## [1] 0.108
```

```
K_fold_CV(training_data,10)
```

```
## [1] 0.112
```

```
epe.boot(training_data, clas.lin, 1, 1)
```

```
## [1] 0.150838
```

```
epe.boot(training_data, clas.lin, 1, 10)
```

```
## [1] 0.1120598
```

```
epe.boot(training_data, clas.lin, 1, 100)
```

```
## [1] 0.1120223
```

We see that the EPE estimated by bootstraping (when taking enough samples) is close to the estimate given by 2-fold cross validation, which happens to be also close to the result of the 1o-fold cross validation.

Problem 5

We first find the best number of neighbors.

```
m <- 2
kk <- seq(1, 501, 10)
clas.vote.epe <- get.vote.epe(training_data, m, kk)
idx.best <- which.min(clas.vote.epe)
```

```
kk.best <- kk[idx.best]
print(kk.best)
```

```
## [1] 291
```

Then the optimized EPE is

```
epe.best <- clas.vote.epe[idx.best]
```

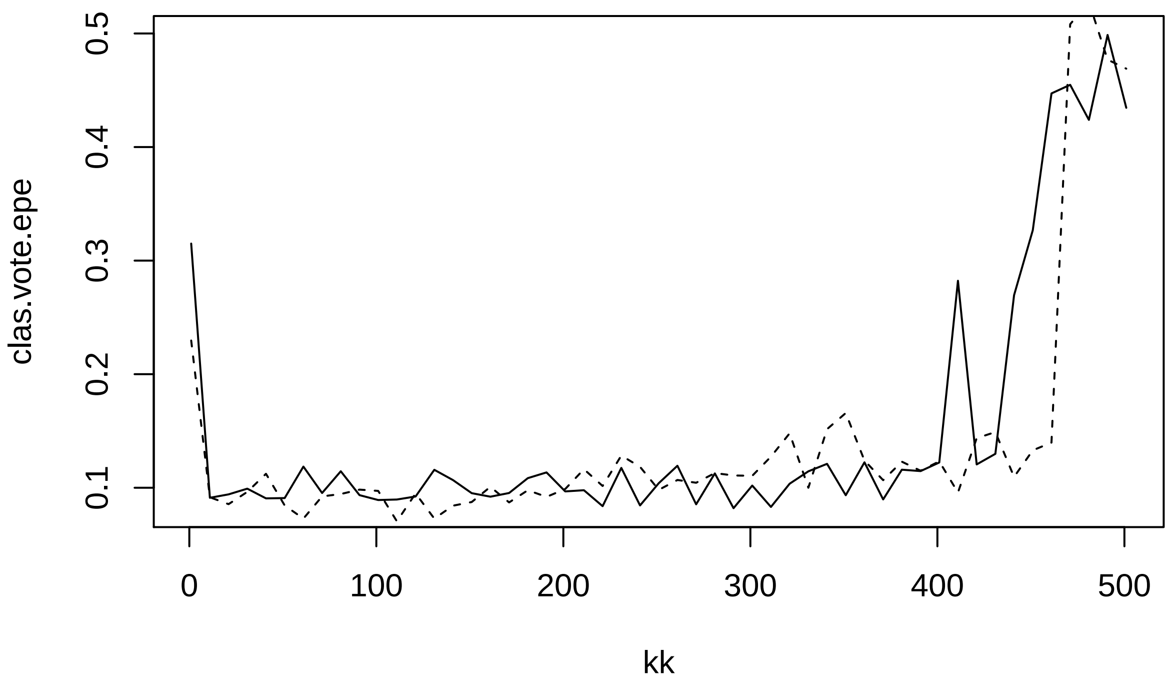
Next, we regenerate the data and use the same number of neighbors on this data to find the EPE.

```
training_data_new <- gen.data()
clas.vote.epe.new <- get.vote.epe(training_data_new, m, kk)
print(clas.vote.epe.new[idx.best])
```

```
## [1] 0.1108108
```

We see that EPE has gone up on the new data. In addition, we plot the EPE for different number of neighbors for the two data sets.

```
plot(kk, clas.vote.epe, type="l")
points(kk, clas.vote.epe.new, type="l", lty = 2)
```


clas.vote.epe
kk