Lab 6

Tziporah Horowitz

11:59PM March 21, 2020

Load the Boston Housing data and create the vector `y`, design matrix `X` and let `n` and `p_plus_one` be the number of rows and columns.

```
y = MASS::Boston$medv
X = as.matrix(cbind(1, MASS::Boston[, 1 : 13]))
n = nrow(X)
p_plus_one = ncol(X)
```

Create a new matrix `Xjunk` by adding random columns to `X` to make the number of columns and rows the same.

```
Xjunk <- X
for (j in (p_plus_one + 1):n){
  Xjunk <- cbind(Xjunk, rnorm(n))
}
dim(Xjunk)
```

```
## [1] 506 506
```

Test that the projection matrix onto $colsp[Xjunk]$ is the same as I_n:

```
pacman::p_load(testthat)
expect_equal(c(Xjunk %*% solve(t(Xjunk) %*% Xjunk) %*% t(Xjunk)), c(diag(1, n, n)))
```

Write a function spec'd as follows:

```
#' Orthogonal Projection
#'
#' Projects vector a onto v.
#'
#' @param a   the vector to project
#' @param v   the vector projected onto
#'
#' @returns   a list of two vectors, the orthogonal projection parallel to v named a_parallel,
#'            and the orthogonal error orthogonal to v called a_perpendicular
orthogonal_projection = function(a, v){
  a_parallel <- ((v %*% t(v)) / sum(v^2)) %*% a
  a_perpendicular <- a - a_parallel
  list(a_parallel = a_parallel, a_perpendicular = a_perpendicular)
}
```

Provide predictions for each of these computations and then run them to make sure you're correct.

```r
orthogonal_projection(c(1,2,3,4), c(1,2,3,4))
```

```
## $a_parallel
##      [,1]
## [1,]    1
## [2,]    2
## [3,]    3
## [4,]    4
##
## $a_perpendicular
##      [,1]
## [1,]    0
## [2,]    0
## [3,]    0
## [4,]    0
```

```r
#prediction: 0,0,0,0
orthogonal_projection(c(1, 2, 3, 4), c(0, 2, 0, -1))
```

```
## $a_parallel
##      [,1]
## [1,]    0
## [2,]    0
## [3,]    0
## [4,]    0
##
## $a_perpendicular
##      [,1]
## [1,]    1
## [2,]    2
## [3,]    3
## [4,]    4
```

```r
#prediction: 1,2,3,4
result = orthogonal_projection(c(2, 6, 7, 3), c(1, 3, 5, 7))
t(result$a_parallel) %*% result$a_perpendicular
```

```
##               [,1]
## [1,] -3.552714e-15
```

```r
#prediction:
result$a_parallel + result$a_perpendicular
```

```
##      [,1]
## [1,]    2
## [2,]    6
## [3,]    7
## [4,]    3
```

```r
#prediction:
result$a_parallel / c(1, 3, 5 ,7)
```

```
##           [,1]
## [1,] 0.9047619
## [2,] 0.9047619
## [3,] 0.9047619
## [4,] 0.9047619
```

```r
#prediction:
```

Try to project onto the column space of X by projecting y on each vector of X individually and adding up the projections. You can use the function `orthogonal_projection`.

```r
sumProj <- 0
for (j in 1:p_plus_one){
  sumProj <- sumProj + orthogonal_projection(y, X[ , j])$a_parallel
}
```

How much double counting occurred? Measure the magnitude relative to the true LS orthogonal projection.

```r
yhat = lm(y ~ X)$fitted.values
sqrt(sum(sumProj^2)) / sqrt(sum(yhat^2))
```

```
## [1] 8.997118
```

Convert X into V where V has the same column space as X but has orthogonal columns. You can use the function `orthogonal_projection`. This is essentially gram-schmidt.

```r
V <- matrix(NA, nrow = nrow(X), ncol = ncol(X))
V[ , 1] <- X[ , 1]
for (j in 2:p_plus_one){
  V[ , j] <- X[ , j]
  for (k in 1:(j - 1)){
    V[ , j] <- V[ , j] - orthogonal_projection(X[ , j], V[ , k])$a_parallel
  }
}
t(V[ , 1]) %*% V[ , 2]
```

```
##               [,1]
## [1,] -1.544542e-12
```

Convert V into Q whoose columns are the same except normalized

```r
Q = matrix(NA, nrow = nrow(X), ncol = ncol(X))
for (j in 1:p_plus_one){
  Q[ , j] <- V[ , j] / sqrt(sum(V[ , j]^2))
}
```

Verify $Q^T Q$ is I_{p+1}, i.e. Q an orthonormal matrix.

```r
expect_equal(t(Q) %*% Q, diag(p_plus_one))
```

Project y onto $colsp[Q]$ and verify it is the same as the OLS fit.

```r
expect_equal(c(unname(Q %*% t(Q) %*% y)), unname(yhat))
```

Project y onto the columns of Q one by one and verify it sums to be the projection onto the whole space.

```r
sumProj <- 0
for (j in 1:p_plus_one){
  sumProj <- sumProj + orthogonal_projection(y, Q[ , j])$a_parallel
}
```

Verify the sum of projections is $yhat$

```r
expect_equal(c(sumProj), unname(yhat))
```

Split the Boston Housing Data into a training set and a test set where the training set is 80% of the observations. Do so at random.

```r
prop_train <- 0.8
n_train <- round(n * prop_train)
indx_train <- sample(1:n, n_train, replace = FALSE)
indx_test <- setdiff(1:n, indx_train)
expect_equal(sort(c(indx_test, indx_train)), 1:n)

X_train <- X[indx_train, ]
X_test <- X[indx_test, ]
y_train <- y[indx_train]
y_test <- y[indx_test]


dim(X_test)
```

```
## [1] 101  14
```

```r
dim(X_train)
```

```
## [1] 405  14
```

Find the s_e in sample and out of sample. Which one is greater? Note: we are now using s_e and not RMSE since RMSE has the $-(p+1)$ in the denominator which makes comparison more difficult when the n's are different.

```r
mod <- lm(y_train ~ ., data.frame(X_train))

y_hat_oos <- predict(mod, data.frame(X_test))
oos_residuals <- y_test - y_hat_oos

c(sd(mod$residuals), sd(oos_residuals))
```

```
## [1] 4.826669 4.177014
```

Do these two exercises 1,000 times and find the average difference between s_e and $\text{oos}s_e$. This is just `sd(e)` the standard deviation of the residuals.

```r
diff <- c()

for (i in 1:1000) {
  indx_train <- sample(1:n, n_train, replace = FALSE)
  indx_test <- setdiff(1:n, indx_train)

  X_train <- X[indx_train, ]
  X_test <- X[indx_test, ]
  y_train <- y[indx_train]
  y_test <- y[indx_test]

  mod <- lm(y_train ~ ., data.frame(X_train))

  y_hat_oos <- predict(mod, data.frame(X_test))
  oos_residuals <- y_test - y_hat_oos

  diff <- c(diff, sd(mod$residuals) - sd(oos_residuals))
}

mean(diff)
```

```
## [1] -0.1849492
```

Using `Xjunk` from above, divide the data into training and testing sets. Fit the model in-sample and calculate s_e in-sample by varying the number of columns used beginning with the first column. Keep the s_e values in the variable `s_es` which has length n. Show that it reaches 0 at n i.e. the model overfits.

```r
Xjunk_train <- Xjunk[indx_train, ]
Xjunk_test <- Xjunk[indx_test, ]


s_es <- c()
for (i in 1:ncol(Xjunk_train)){
  mod_in <- lm(y_train ~ ., data.frame(Xjunk_train[ , 1:i]))
  s_es <- c(s_es, sd(mod_in$residuals))
}

expect_equal(which(s_es == 0)[1], nrow(Xjunk_train))
```

Do the same thing but now calculate $\text{oos}s_e$. Does this go to zero? What is the index corresponding to the best model?

```r
s_es <- c()

for (i in 2:ncol(Xjunk_train)){
  mod_in <- lm(y_train ~ ., data.frame(Xjunk_train[ , 1:i]))
  y_hat_oos <- predict(mod_in, data.frame(Xjunk_test[ , 1:i]))
  oos_residuals <- y_test - y_hat_oos

  s_es <- c(s_es, sd(oos_residuals))
}

which.min(s_es)
```

```
## [1] 13
```

Beginning with the Boston Housing Data matrix `X`, pull out the second column, the `crim` feature and call it `x2`. Then, use the `cut` function to bin each of its n values into two bins: the first is all values <= the median of `crim` and the second is all values > median of `crim`. Call it `x2bin`. Use the `table` function to ensure that half of the values are in the first group and half in the second group. This requires reading the documentation for `cut` carefully and using the `quantile` function carefully.

```r
x2 <- X[ , 2]
x2_bin <- cut(x2, breaks = quantile(x2, c(0, .5, 1)), include.lowest = TRUE)
table(x2_bin)
```

```
## x2_bin
## [0.00632,0.257]      (0.257,89]
##             253             253
```

Now convert the factor variable `x2bin` to two dummies, `X2dummy`, a matrix of $n \times 2$ and verify the rowsums are all 1. They must be 1 because either the value is <= median or > median.

```r
x2dummy <- model.matrix(~ 0 + ., data.frame(x2_bin))
table(rowSums(x2dummy))
```

```
##
##   1
## 506
```

Drop the first column of this matrix to arrive at `X2dummyfeatures`.

```r
X2dummyfeatures <- x2dummy[ , 2]
```

What you did with `crim`, do for all 13 variables in the Boston housing data, ie create `X2dummyfeatures` for all and then column bind them all together into a massive `Xdummy` matrix. Then run a regression of y on those features and report R^2.

```r
# NOTE: X[ , 5] is already binary
Xdummy <- X2dummyfeatures
for (j in 3:14){
  x <- X[ , j]
  # if the quantiles are not unique, you cannot cut by the median, so cut by the mean
  if (min(x) == median(x) | max(x) == median(x)) {
    b <- c(min(x), mean(x), max(x))
  }
  else {
    b <- quantile(x, c(0, .5, 1))
  }
  x_bin <- cut(x, breaks = b, include.lowest = TRUE)
  xdummy <- model.matrix(~ 0 + ., data.frame(x_bin))
  xdummyfeatures_j <- xdummy[ , 2]
  Xdummy <- cbind(Xdummy, xdummyfeatures_j)
}
colnames(Xdummy) <- colnames(X[ ,2:14])

mod1 <- lm(y ~ 0 + ., data = data.frame(Xdummy))
summary(mod1)$r.squared
```

```
## [1] 0.8636389
```

This time create two dummies for each variable: (1) between the 33%ile and 66%ile and (2) greater than the 66%ile. Run the regression on all dummies for all variables and report R^2. Hint: you do not need to go through the exercise of creating the dummy columns manually; use `factor` instead. Then use `lm` to run the regression (do not do it manually using the X matrices).

```
Xnew <- c()
for (i in 2:ncol(X)) {
  if (i == 5) {
    Xn <- factor(X[, i])
    Xn <- model.matrix( ~ 0 + ., data = data.frame(Xn))
  }
  else if (i == 3) {
    b <- c(min(x), mean(x), max(x))
    Xn <- factor(cut(x, breaks = b, include.lowest = TRUE))
    Xn <- model.matrix( ~ 0 + ., data = data.frame(Xn))[ , 2]
  }
  else {
    Xn <- factor(cut(X[ , i], breaks = quantile(X[ , i], seq(0, 1, length.out = 4)), include.lowest = T
    Xn <- model.matrix( ~ 0 + ., data = data.frame(Xn))[ , 2:3]
  }
  Xnew <- cbind(Xnew, Xn)
}

mod2 <- lm(y ~ 0 + Xnew)
summary(mod2)$r.squared
```

```
## [1] 0.9498768
```

Keep doing this until each continuous variable has 31 dummies for a total of $p = 403 + 1$ variables. Report all R^2;s. Why is it increasing and why is the last one so high?

```
bin_num <- 31
R2s <- c(summary(mod1)$r.squared, summary(mod2)$r.squared)
i=31
for (i in 4:bin_num) {
  Xnew <- c()
  for (j in 2:ncol(X)){
    if (j == 5) {
      Xn <- factor(X[, j])
      Xn <- model.matrix( ~ 0 + ., data = data.frame(Xn))
    }
    else if (j == 4 & i > 5){
      Xn <- factor(cut(X[ , j], breaks = quantile(X[ , j], seq(0, 1, length.out = 6)), include.lowest =
      Xn <- model.matrix( ~ 0 + ., data = data.frame(Xn))[ , 2:5]
    }
    else if (j == 6 & i > 27){
      Xn <- factor(cut(X[ , j], breaks = quantile(X[ , j], seq(0, 1, length.out = 28)), include.lowest
      Xn <- model.matrix( ~ 0 + ., data = data.frame(Xn))[ , 2:27]
    }
    else if (j == 8 & i > 12){
      Xn <- factor(cut(X[ , j], breaks = quantile(X[ , j], seq(0, 1, length.out = 13)), include.lowest
      Xn <- model.matrix( ~ 0 + ., data = data.frame(Xn))[ , 2:12]
    }
    else if (j == 10) {
      Xn <- factor(cut(X[ , j], breaks = quantile(X[ , j], seq(0, 1, length.out = 4)), include.lowest =
      Xn <- model.matrix( ~ 0 + ., data = data.frame(Xn))[ , 2:3]
    }
    else if (j == 11 & i > 7){
      Xn <- factor(cut(X[ , j], breaks = quantile(X[ , j], seq(0, 1, length.out = 8)), include.lowest =
      Xn <- model.matrix( ~ 0 + ., data = data.frame(Xn))[ , 2:7]
    }
    else if (j == 12 & i > 5){
      Xn <- factor(cut(X[ , j], breaks = quantile(X[ , j], seq(0, 1, length.out = 6)), include.lowest =
      Xn <- model.matrix( ~ 0 + ., data = data.frame(Xn))[ , 2:5]
    }
    else if (j == 13 & i > 4){
      Xn <- factor(cut(X[ , j], breaks = quantile(X[ , j], seq(0, 1, length.out = 5)), include.lowest =
      Xn <- model.matrix( ~ 0 + ., data = data.frame(Xn))[ , 2:4]
    }
    else if (j == 3) {
      b <- b <- c(min(X[ , 3]), mean(X[ , 3]), max(X[ , 3]))
      Xn <- factor(cut(X[ , 3], breaks = b, include.lowest = TRUE))
      Xn <- model.matrix( ~ 0 + ., data = data.frame(Xn))[ , 2]
    }
    else {
      Xn <- factor(cut(X[ , j], breaks = quantile(X[ , j], seq(0, 1, length.out = i + 1)), include.lowe
      Xn <- model.matrix( ~ 0 + ., data = data.frame(Xn))[ , 2:i]
    }
    Xnew <- cbind(Xnew, Xn)
  }
  modn <- lm(y ~ 0 + Xnew)
  R2s <- c(R2s, summary(modn)$r.squared)
}

R2s
```

Repeat this exercise with a 20% test set held out. Record in sample s_e's and ooss_e's. Do we see the canonical picture?

```
se_s <- c()
oose_s <- c()

for (i in 4:bin_num) {
  Xnew <- c()
  for (j in 2:ncol(X)){
    if (j == 5) {
      Xn <- factor(X[, j])
      Xn <- model.matrix( ~ 0 + ., data = data.frame(Xn))
    }
    else if (j == 4 & i > 5){
      Xn <- factor(cut(X[ , j], breaks = quantile(X[ , j], seq(0, 1, length.out = 6)), include.lowest =
      Xn <- model.matrix( ~ 0 + ., data = data.frame(Xn))[ , 2:5]
    }
    else if (j == 6 & i > 27){
      Xn <- factor(cut(X[ , j], breaks = quantile(X[ , j], seq(0, 1, length.out = 28)), include.lowest =
      Xn <- model.matrix( ~ 0 + ., data = data.frame(Xn))[ , 2:27]
    }
    else if (j == 8 & i > 12){
      Xn <- factor(cut(X[ , j], breaks = quantile(X[ , j], seq(0, 1, length.out = 13)), include.lowest =
      Xn <- model.matrix( ~ 0 + ., data = data.frame(Xn))[ , 2:12]
    }
    else if (j == 10) {
      Xn <- factor(cut(X[ , j], breaks = quantile(X[ , j], seq(0, 1, length.out = 4)), include.lowest =
      Xn <- model.matrix( ~ 0 + ., data = data.frame(Xn))[ , 2:3]
    }
    else if (j == 11 & i > 7){
      Xn <- factor(cut(X[ , j], breaks = quantile(X[ , j], seq(0, 1, length.out = 8)), include.lowest =
      Xn <- model.matrix( ~ 0 + ., data = data.frame(Xn))[ , 2:7]
    }
    else if (j == 12 & i > 5){
      Xn <- factor(cut(X[ , j], breaks = quantile(X[ , j], seq(0, 1, length.out = 6)), include.lowest =
      Xn <- model.matrix( ~ 0 + ., data = data.frame(Xn))[ , 2:5]
    }
    else if (j == 13 & i > 4){
      Xn <- factor(cut(X[ , j], breaks = quantile(X[ , j], seq(0, 1, length.out = 5)), include.lowest =
      Xn <- model.matrix( ~ 0 + ., data = data.frame(Xn))[ , 2:4]
    }
    else if (j == 3) {
      b <- b <- c(min(X[ , 3]), mean(X[ , 3]), max(X[ , 3]))
      Xn <- factor(cut(X[ , 3], breaks = b, include.lowest = TRUE))
      Xn <- model.matrix( ~ 0 + ., data = data.frame(Xn))[ , 2]
    }
    else {
      Xn <- factor(cut(X[ , j], breaks = quantile(X[ , j], seq(0, 1, length.out = i + 1)), include.lowe
      Xn <- model.matrix( ~ 0 + ., data = data.frame(Xn))[ , 2:i]
    }
    Xnew <- cbind(Xnew, Xn)

    X_train <- Xnew[indx_train, ]
    X_test <- Xnew[indx_test, ]
    y_train <- y[indx_train]
    y_test <- y[indx_test]
```

```r
    y_hat_oos <- predict(mod, data.frame(X_test))
    oos_residuals <- y_test - y_hat_oos
  }
  modn <- lm(y ~ 0 + X_train)
  se_s <- c(se_s, sd(summary(modn)$residuals))
  oose_s <- c(oose_s, sd(oos_residuals))
}

se_s
oose_s
```

What is the optimal number of bins (dummies) for each feature? Worded another way, what is the optimal complexity model among this set modeling strategy (binning)?

```r
which.min(se_s)
```