BIOST 527 Homework 4

Benjamin Stan

May 24, 2021

Question 1

(a)

The simulation was generated with the following parameters:

- Sample size (N) of 50
- Number of features (p) set to 5000
- X matrix of (50x5000) independent observations generated from a $N(0, 1)$ distribution
- $\vec{y}$ vector of 50 values with 25 observations labeled class 1 and 25 observation labeled class 2 at random
- The subset of the covariates will take the 100 most highly correlated features
- K-fold cross-validation will be performed with k=5

The code to create these variables is shown below:

```
######
# 1a
######
## Create dataset for simulation
set.seed(44)
N = 50
p = 5000
y = sample(c(rep(1,(N/2)),rep(2,(N/2))), N, replace=FALSE)
X = matrix(rnorm(N*p),N,p)
k_folds = sample(rep(1:5,(N/5)), N, replace=FALSE)
```

(b)

To perform cross-validation "the wrong way", the 100 most correlated features in X were identified. The subset was then used to train a KNN model, and cross-validation was used to estimate the test error. Using this method for 50 simulations, the estimated error was 1.4%.

```
######
# 1b
######
## Perform CV error calculation the "wrong way"
rm(list = ls())
set.seed(44)
N = 50
p = 5000
error_wrong = NULL
for (j in 1:50) {
  ## Define y as even split from two classes
  y = sample(c(rep(1,(N/2)),rep(2,(N/2))), N, replace=FALSE)
  ## Define X from N(0,1)
  X = matrix(rnorm(N*p),N,p)
  ## Find most correlated features
  split_X = split(X, rep(1:ncol(X), each = nrow(X)))
  cor_vals = unlist(map(split_X, function(z) abs(cor(z,y))))
  cor_df = as.data.frame(cor_vals)
  cor_df$index = 1:p
  top_features = cor_df[order(cor_vals,decreasing =TRUE),][1:100,2]
  X_wrong = X[,top_features]
  ## Performed K-fold cross-validation with K=5
  k_folds = sample(rep(1:5,(N/5)), N, replace=FALSE)
  error_wrong_j = NULL
  for (i in 1:5) {
    ## Get predicted values from KNN with k=1
    knn_pred <- knn(X_wrong[k_folds!=i,],X_wrong[k_folds==i,],y[k_folds!=i],k=1)
    conf_matrix <- table(knn_pred,y[k_folds==i])
    error_wrong_j <- c(error_wrong_j,1-(conf_matrix[1,1]+conf_matrix[2,2])/sum(conf_matrix))
  }
  mean(error_wrong_j)
  ## Append error to vector with errors for all simulations
  error_wrong = c(error_wrong,mean(error_wrong_j))
}
mean(error_wrong)
```

(c)

To perform cross-validation "the right way", the data was split into the K folds, and the strongest correlated features were identified K times, once for each excluded fold. The KNN classifier was then trained on these features excluding data from the fold of interest, and then evaluated using the data from that fold. Using this method for 50 simulations, the estimated error was 53.7%.

```
######
# 1c
######
## Perform CV error calculation the "wrong way"
rm(list = ls())
set.seed(44)
N = 50
p = 5000
error_right = NULL
```

```
for (j in 1:50) {
  y = sample(c(rep(1,(N/2)),rep(2,(N/2))), N, replace=FALSE)
  X = matrix(rnorm(N*p),N,p)
  k_folds = sample(rep(1:5,(N/5)), N, replace=FALSE)
  error_right_j = NULL
  ## Loop through folds, finding features and calculating error on ith fold
  for (i in 1:5) {
    X_k = X[k_folds!=i,]
    split_X = split(X_k, rep(1:ncol(X_k), each = nrow(X_k)))
    cor_vals = unlist(map(split_X, function(z) abs(cor(z,y[k_folds!=i]))))
    cor_df = as.data.frame(cor_vals)
    cor_df$index = 1:p
    top_features = cor_df[order(cor_vals,decreasing =TRUE),][1:100,2]
    X_right = X_k[,top_features]
    ## Perform KNN with k=1
    knn_pred <- knn(X_right,X[k_folds==i,top_features],y[k_folds!=i],k=1)
    conf_matrix <- table(knn_pred,y[k_folds==i])
    error_right_j <- c(error_right_j,1-(conf_matrix[1,1]+conf_matrix[2,2])/sum(conf_matrix))
  }
  mean(error_right_j)
  ## Append error to vector with errors for all simulations
  error_right = c(error_right,mean(error_right_j))
}
mean(error_right)
```

(d)

The "wrong way" approach was incorrect because it considers all of the data, including the data that will ultimately be used to evaluate the model, when selecting the most correlated features. Splitting data into test and train sets and using cross-validation are intended to mimic the performance of a model on an separate, independent data set, and the "wrong way" clearly violates this notion. This is evident by the small error rate (1.4%) when compared to the "right way" (53.7%) and the true error rate of the data-generating function (50%).

Question 2

The goal is to prove that the difference between

$$Err_{in} = \frac{1}{N}\sum_{i=1}^{N} E_{Y_i^0}\left(Y_i^0 - \hat{f}(x_i)\right)^2$$

$$\overline{err} = \frac{1}{N}\sum_{i=1}^{N} \left(y_i - \hat{f}(x_i)\right)^2$$

has the expectation

$$\frac{2}{N}\sum_{i=1}^{N} \mathrm{Cov}\left(y_i, \hat{f}(x_i)\right)$$

We will first add and subtract terms from within each expression

$$Err_{in} = \frac{1}{N}\sum_{i=1}^{N} E_{Y_i^0}\big((Y_i^0 - f(x_i)) + (f(x_i) - E(\hat{f}(x_i))) + (E(\hat{f}(x_i)) - \hat{f}(x_i))\big)^2$$

$$\overline{err} = \frac{1}{N}\sum_{i=1}^{N} \big((y_i - f(x_i)) + (f(x_i) - E(\hat{f}(x_i))) + (E(\hat{f}(x_i)) - \hat{f}(x_i))\big)^2$$

Now considering the expanded terms:

$$\begin{aligned}
Err_{in} &= \frac{1}{N}\sum_{i=1}^{N} E_{Y_i^0}\big((Y_i^0 - f(x_i)) + (f(x_i) - E(\hat{f}(x_i))) + (E(\hat{f}(x_i)) - \hat{f}(x_i))\big)^2 \\
&= \frac{1}{N}\Bigg[\sum_{i=1}^{N} E_{Y_i^0}[(Y_i^0 - f(x_i))^2] + \sum_{i=1}^{N}(f(x_i) - E(\hat{f}(x_i)))^2 + \sum_{i=1}^{N}(E(\hat{f}(x_i)) - \hat{f}(x_i))^2 \\
&+ 2\sum_{i=1}^{N} E_{Y_i^0}[(Y_i^0 - f(x_i))(f(x_i) - E(\hat{f}(x_i)))] + 2\sum_{i=1}^{N}(f(x_i) - E(\hat{f}(x_i)))(E(\hat{f}(x_i)) - \hat{f}(x_i)) \\
&+ 2\sum_{i=1}^{N} E_{Y_i^0}[(Y_i^0 - f(x_i))(E(\hat{f}(x_i)) - \hat{f}(x_i))]\Bigg] \\
\overline{err} &= \frac{1}{N}\sum_{i=1}^{N} \big((y_i - f(x_i)) + (f(x_i) - E(\hat{f}(x_i))) + (E(\hat{f}(x_i)) - \hat{f}(x_i))\big)^2 \\
&= \frac{1}{N}\Bigg[\sum_{i=1}^{N}(y_i - f(x_i))^2 + \sum_{i=1}^{N}(f(x_i) - E(\hat{f}(x_i)))^2 + \sum_{i=1}^{N}(E(\hat{f}(x_i)) - \hat{f}(x_i))^2 \\
&+ 2\sum_{i=1}^{N}(y_i - f(x_i))(f(x_i) - E(\hat{f}(x_i))) + 2\sum_{i=1}^{N}(f(x_i) - E(\hat{f}(x_i)))(E(\hat{f}(x_i)) - \hat{f}(x_i)) \\
&+ 2\sum_{i=1}^{N}(y_i - f(x_i))(E(\hat{f}(x_i)) - \hat{f}(x_i))\Bigg]
\end{aligned}$$

Take the difference between the two terms:

$$\begin{aligned}
Err_{in} - \overline{err} &= \frac{1}{N}\Bigg[\sum_{i=1}^{N} E_{Y_i^0}[(Y_i^0 - f(x_i))^2] + 2\sum_{i=1}^{N} E_{Y_i^0}[(Y_i^0 - f(x_i))(f(x_i) - E(\hat{f}(x_i))] \\
&+ 2\sum_{i=1}^{N} E_{Y_i^0}[(Y_i^0 - f(x_i))(E(\hat{f}(x_i)) - \hat{f}(x_i))] - \sum_{i=1}^{N}(y_i - f(x_i))^2 - 2\sum_{i=1}^{N}(y_i - f(x_i))(f(x_i) - E(\hat{f}(x_i))) \\
&- 2\sum_{i=1}^{N}(y_i - f(x_i))(E(\hat{f}(x_i)) - \hat{f}(x_i))\Bigg] \\
&= \frac{1}{N}\Bigg[\Bigg(\sum_{i=1}^{N} E_{Y_i^0}[(Y_i^0 - f(x_i))^2] - \sum_{i=1}^{N}(y_i - f(x_i))^2\Bigg) + \Bigg(2\sum_{i=1}^{N} E_{Y_i^0}[(Y_i^0 - f(x_i))(f(x_i) - E(\hat{f}(x_i)))] \\
&- 2\sum_{i=1}^{N}(y_i - f(x_i))(f(x_i) - E(\hat{f}(x_i))\Bigg) + \Bigg(2\sum_{i=1}^{N} E_{Y_i^0}[(Y_i^0 - f(x_i))(E(\hat{f}(x_i)) - \hat{f}(x_i))] \\
&- 2\sum_{i=1}^{N}(y_i - f(x_i))(E(\hat{f}(x_i)) - \hat{f}(x_i))\Bigg)\Bigg]
\end{aligned}$$

Taking the expectation of the first term:

$$E\Big(\sum_{i=1}^{N} E_{Y_i^0}[(Y_i^0 - f(x_i))^2] - \sum_{i=1}^{N}(y_i - f(x_i))^2\Big) = E\Big(\sum_{i=1}^{N} E_{Y_i^0}[(Y_i^0 - f(x_i))^2]\Big) - E\Big(\sum_{i=1}^{N}(y_i - f(x_i))^2\Big)$$
$$= N\sigma_\epsilon^2 - N\sigma_\epsilon^2$$
$$= 0$$

Taking the expectation of the second term

$$E\Big(2\sum_{i=1}^{N} E_{Y_i^0}[(Y_i^0 - f(x_i))(f(x_i) - E(\hat{f}(x_i)))] - 2\sum_{i=1}^{N}(y_i - f(x_i))(f(x_i) - E(\hat{f}(x_i)))\Big)$$
$$= E\Big(2\sum_{i=1}^{N} E_{Y_i^0}[(Y_i^0 - f(x_i))(f(x_i) - E(\hat{f}(x_i)))]\Big) - E\Big(2\sum_{i=1}^{N}(y_i - f(x_i))(f(x_i) - E(\hat{f}(x_i)))\Big)$$
$$= \Big(2\sum_{i=1}^{N} E_{Y_i^0}[(f(x_i) - f(x_i))(f(x_i) - E(\hat{f}(x_i)))]\Big) - \Big(2\sum_{i=1}^{N}(f(x_i) - f(x_i))(f(x_i) - E(\hat{f}(x_i)))\Big)$$
$$= 0$$

Taking the expectation of the third term:

$$E\Big(2\sum_{i=1}^{N} E_{Y_i^0}[(Y_i^0 - f(x_i))(E(\hat{f}(x_i)) - \hat{f}(x_i))] - 2\sum_{i=1}^{N}(y_i - f(x_i))(E(\hat{f}(x_i)) - \hat{f}(x_i))\Big)$$
$$= E\Big(2\sum_{i=1}^{N} E_{Y_i^0}[(Y_i^0 - f(x_i))(E(\hat{f}(x_i)) - \hat{f}(x_i))]\Big) - E\Big(2\sum_{i=1}^{N}(y_i - f(x_i))(E(\hat{f}(x_i)) - \hat{f}(x_i))\Big)$$
$$= \Big(2\sum_{i=1}^{N} E_{Y_i^0}[(f(x_i) - f(x_i))(E(\hat{f}(x_i)) - \hat{f}(x_i))]\Big) - E\Big(2\sum_{i=1}^{N}(y_i - f(x_i))(E(\hat{f}(x_i)) - \hat{f}(x_i))\Big)$$
$$= 0 - E\Big(2\sum_{i=1}^{N}(y_i - f(x_i))(E(\hat{f}(x_i)) - \hat{f}(x_i))\Big)$$

This term evaluates to the covariance between the sample and the expectation, as it captures patterns in deviance from the mean for each.

$$Err_{in} - \overline{err} = \frac{-2}{N}\sum_{i=1}^{N} \mathrm{Cov}\big(y_i, \hat{f}(x_i)\big)$$

As we are considering the difference, we can switch the order of the subtraction to obtain

$$\overline{err} - Err_{in} = \frac{2}{N}\sum_{i=1}^{N} \mathrm{Cov}\big(y_i, \hat{f}(x_i)\big)$$

Question 3

(a)

The code below takes inputs, N, p, and K and outputs the following quantity:

$$\sum_{i=1}^{N} y_i \hat{y}_i$$

```
######
# 3a
######
knn_sum <- function(N=100,p=50,K=3){
  X = matrix(runif(N*p,min=0,max=1),N,p) # Generate X from Unif(0,1)
  y = rnorm(N)
  pred_vals = FNN::knn.reg(train=X,test=X,y=y,k=K)$pred # Get predicted values from KNN
  return(y%*%pred_vals)
}
knn_sum(100,50,4)
```

(b)

We are looking for an estimator of the true degrees of freedom for K-nearest neighbors regression, given by

$$\frac{1}{\sigma^2} \sum_{i=1}^{N} \text{Cov}(y_i, \hat{y}_i)$$

The expression can be equivalently written

$$\frac{1}{\sigma^2} \sum_{i=1}^{N} \text{Cov}(y_i, \hat{y}_i) = \frac{1}{\sigma^2} \sum_{i=1}^{N} \big(E(y_i \hat{y}_i) - E(y_i) E(\hat{y}_i) \big)$$

As $\vec{y}$ is drawn from a N(0,1) distribution, $E(y_i) = 0$ and $\sigma^2 = 1$. Thus, the expression evaluates to

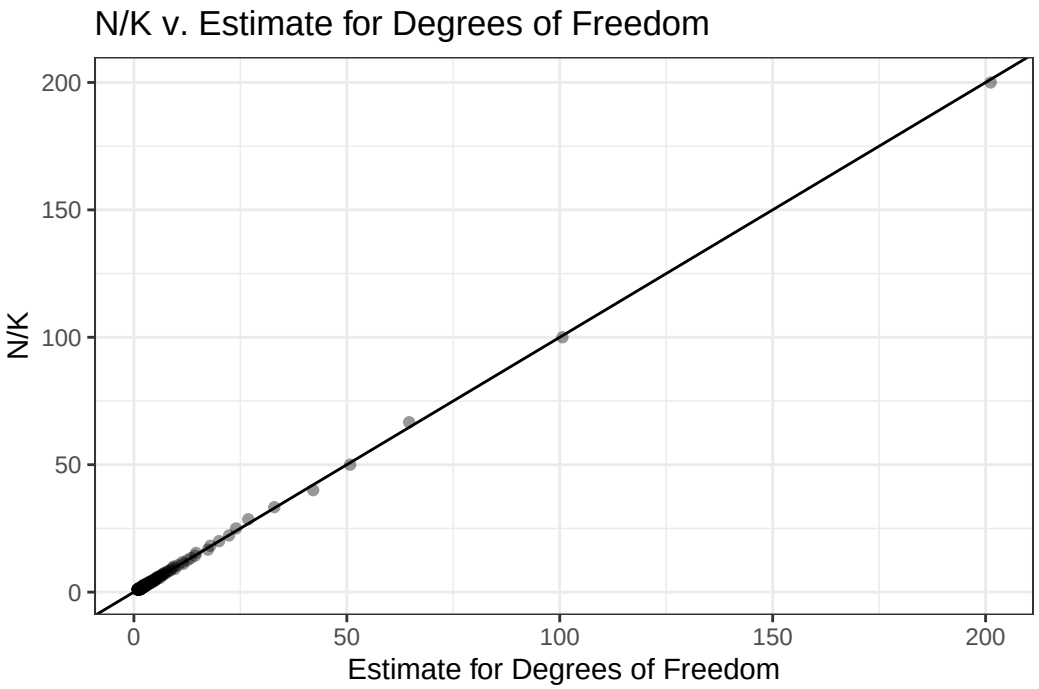
$$\sum_{i=1}^{N} E(y_i \hat{y}_i)$$

If we approximate the expectation by taking the average of B=500 runs of the functions, we can estimate the expression with

$$\frac{1}{B} \sum_{b=1}^{N} \sum_{i=1}^{N} y_{ib} \hat{y}_{ib}$$

(c)

The plot comparing N/K against the estimate discussed in part (b) is shown in the graph below.

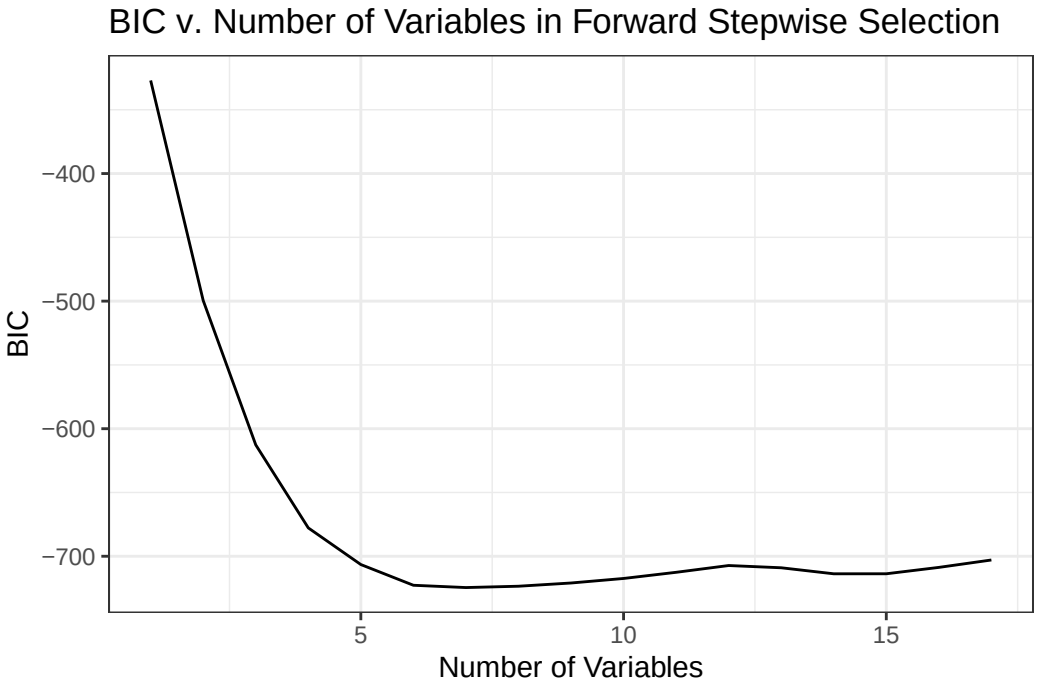


The graph shows that the points lie approximately on a straight line of slope 1 going through the origin. This indicates that the values are approximately equal at values of K from 1 to 200 and that N/K is also a viable estimate for the degrees of freedom for K-nearest neighbors regression.

Question 4

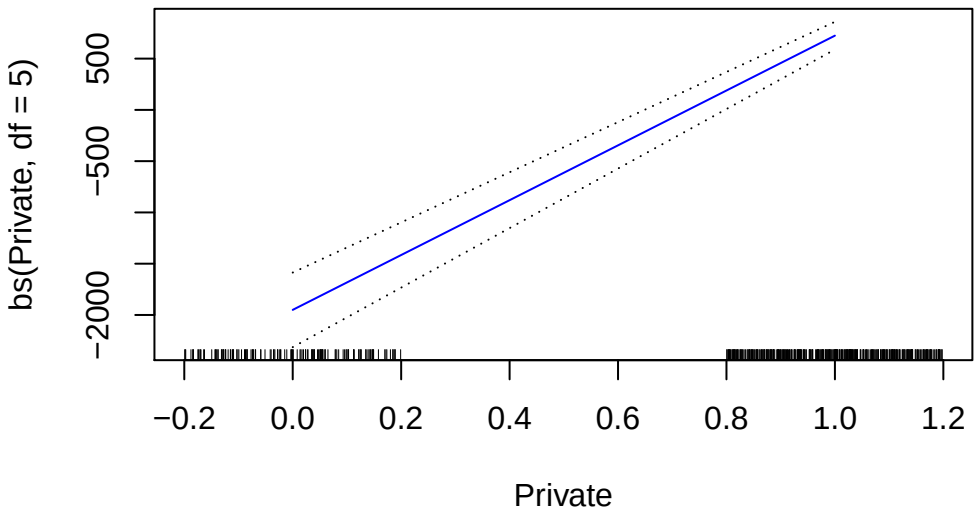
(a)

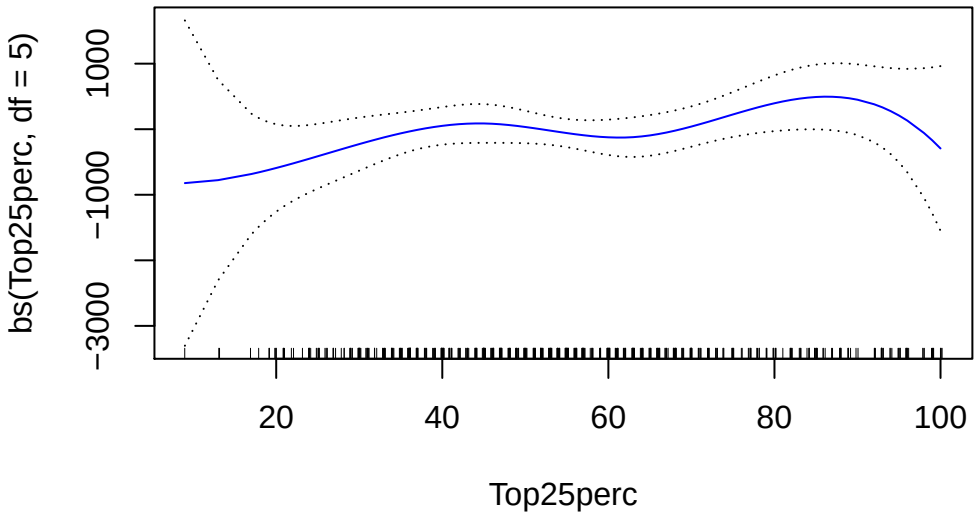
In order to identify the subset of features that will be used to train the GAM model, forward stepwise selection was performed on the training set using the `regsubsets` function. This process identified the model containing seven variables to have the lowest BIC. This metric was chosen because it includes a penalty term on the number of parameters estimated by the model. The seven variables selected were Private, Top25perc, Room.Board, Terminal, perc.alumni, Expend, and Grad.Rate.

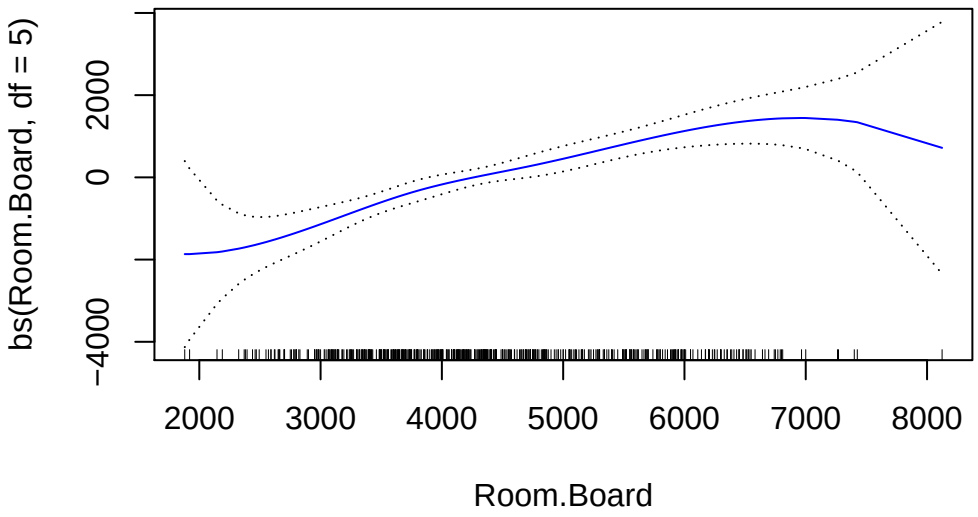


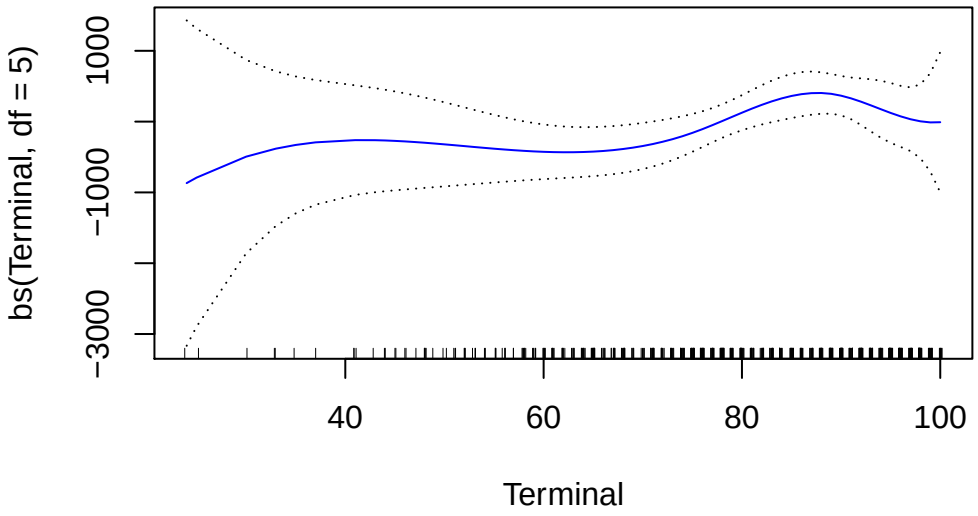
(b)

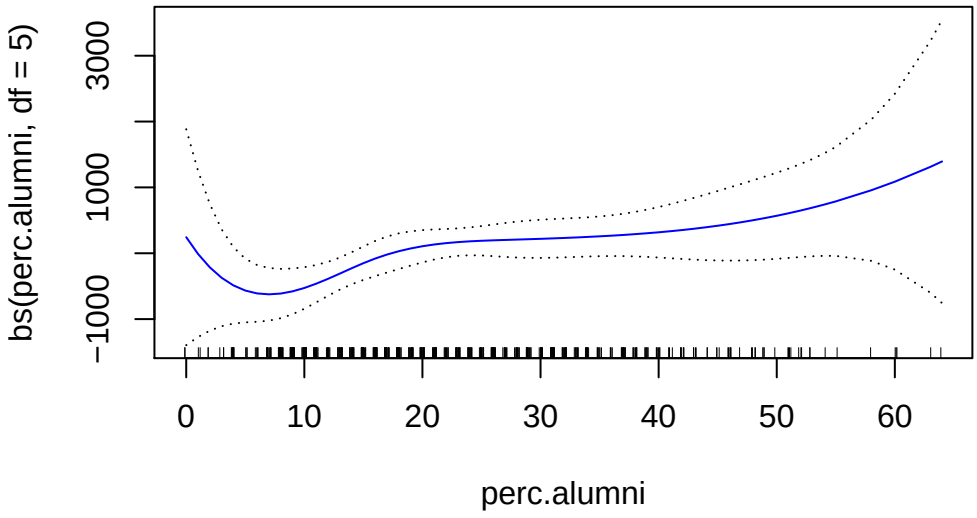
A GAM was fit using out-of-state tuition as the response and the seven variables from part (a) as the predictors. A cubic spline with 5 degrees of freedom was used as for each $f_1, \ldots, f_p$. The plots of the estimating function along with standard errors are shown below. The plot of the fitted function for `Private` is a straight line, which is expected due to the fact that this is a binary variable. The curves for `Top25perc`, `Room.Board`, `perc.alumni`, and `Grad.Rate` have fits with reasonable standard errors over most values of the covariate, but the standard errors increase greatly at the low and high ends of the observed values. The curves for `Terminal` and `Expend` only see this large error increase on one side of the curve.

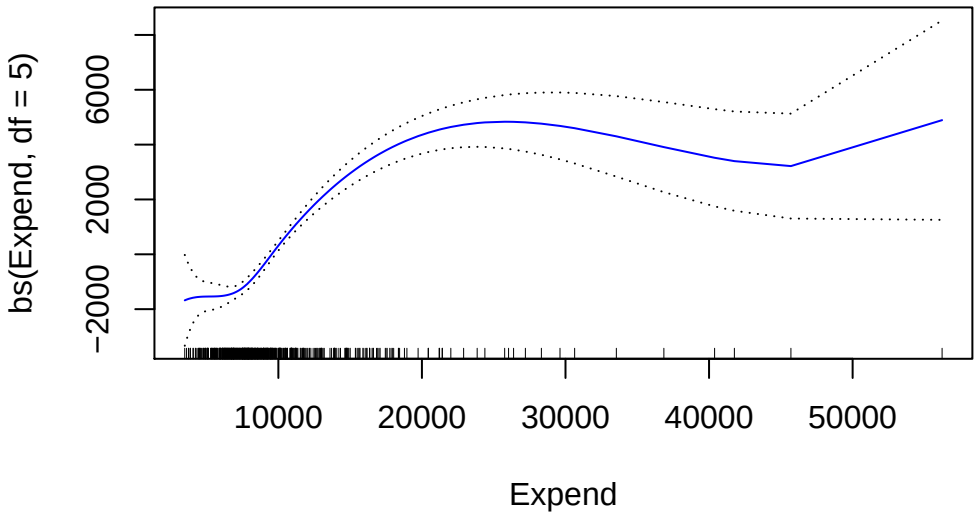

bs(Private, df = 5)
−2000
−500
500
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
Private

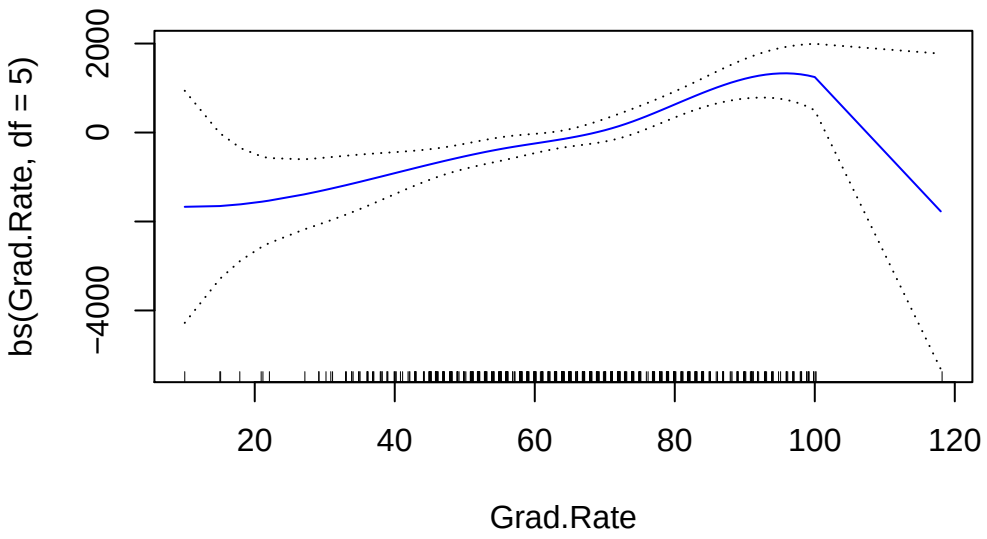

bs(Top25perc, df = 5)
−3000
−1000
1000
20
40
60
80
100
Top25perc


bs(Room.Board, df = 5)
−4000
0
2000
2000
3000
4000
5000
6000
7000
8000
Room.Board


bs(Terminal, df = 5)
−3000
−1000
1000
40
60
80
100
Terminal


bs(perc.alumni, df = 5)
−1000
1000
3000
0
10
20
30
40
50
60
perc.alumni


bs(Expend, df = 5)
−2000
2000
6000
10000
20000
30000
40000
50000
Expend



(c)

When evaluating the GAM on the test set, the mean squared error obtained was 3823830 dollars^2, compared to a training MSE of 3282671 dollars^2. This corresponds to an RMSE of \$1955.46, which is reasonable performance given that the mean value of the out-of-state tuition variable was \$10,476.49. The average of the relative error was 1.4%.

(d)

By using the `summary` function for the GAM, it is possible to view the ANOVA results for each variable and function. The p-values shown below correspond to a null hypothesis of a linear relationship versus the alternative hypothesis of a non-linear relationship. As all functions have a p-value below 0.05, there is strong evidence for a non-linear relationship for all variables considered.

Term	Degrees of Freedom	MSE	Pr(>F)
Private	1	2816659633	$< 2.2 * 10^{-16}$
Top25perc	5	440836638	$< 2.2 * 10^{-16}$
Room.Board	5	241413844	$< 2.2 * 10^{-16}$
Terminal	5	58523341	$2.3 * 10^{-15}$
perc.alumni	5	44329957	$1.2 * 10^{-11}$
Expend	5	117365358	$< 2.2 * 10^{-16}$
Grad.Rate	5	25794902	$1.0 * 10^{-6}$

Question 5

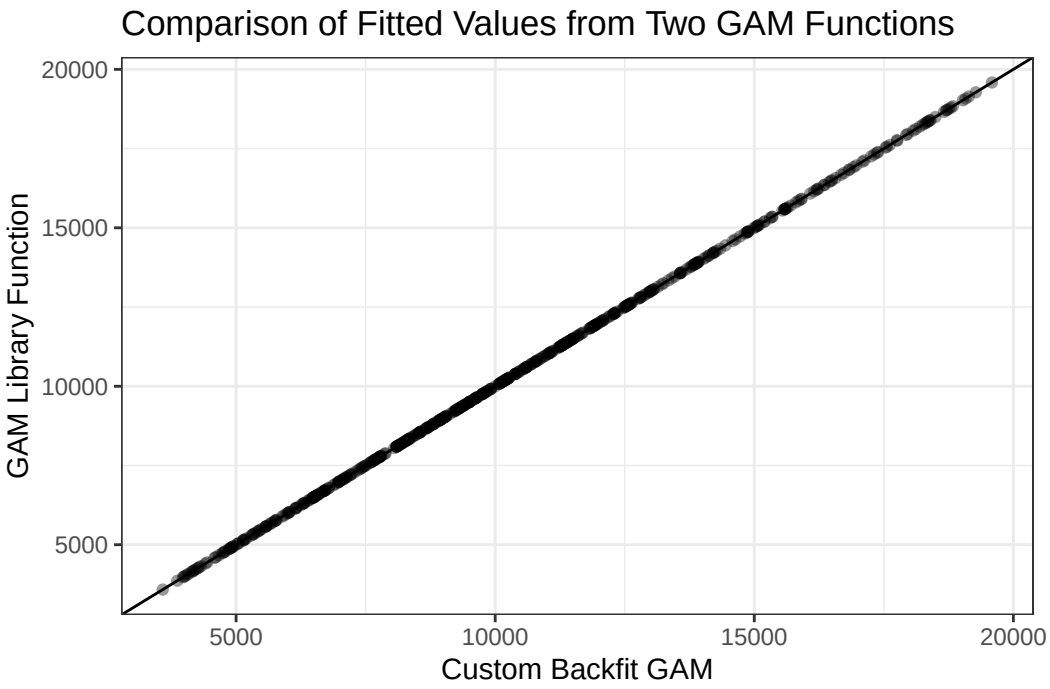
(a)

The code below creates a function to fit a GAM using backfitting.

```
######
# 5a
######
## Define X and y from training set
y = train_df$Outstate
X = train_df[,-1]
custom_backfit <- function(y,X,num_knots = NULL) {
  ## Set default value for knots to be 2 if not specified
  if (is.null(num_knots)) num_knots = rep(2,ncol(X)) # if no knots passed, use 2 for all
  ## Initialize values
  alpha = mean(y) # initialize alpha
  func_list = matrix(0,nrow(X),ncol(X)) # initialize values of functions
  mse = mean(y^2)
  mse_diff = mse
  ## Perform updating using fitted values from each function
  while (mse_diff > 0) {
    for (j in 1:ncol(X)) {
      lm_j = lm((y-alpha-rowSums(func_list[,-j]))~bs(X[,j],df=num_knots[j]+3))
      func_list[,j] = lm_j$fitted.values-mean(lm_j$fitted.values)
    }
    ## Create convergence criteria with MSE
    mse_prev = mse
    mse = mean((y-rowSums(func_list)-alpha)^2)
    mse_diff = mse_prev - mse
  }
  print(mse)
  return(rowSums(func_list)+alpha)
}
```

(b)

The backfitting algorithm created in part (a) was fit using the training data from question 4, part (b). The values for the two GAM models are shown below. As the values lie on the straight line of slope 1 through the origin, the fitted values between the two models are identical (to an order of $1 * 10^{-5}$).



Question 6

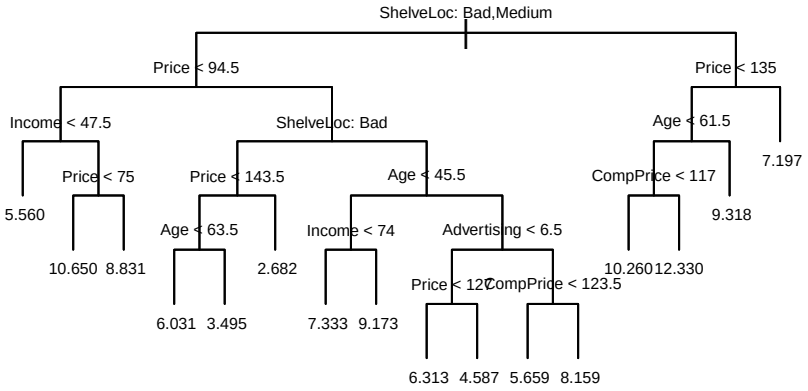
(a)

The code below splits the data into a training set and a test set.

```
######
# 6a
######
rm(list = ls())
set.seed(145)
train_obs = sample.int(n=nrow(Carseats), size=floor(.6*nrow(Carseats)), replace=F)
train_df = Carseats[train_obs,]
test_df = Carseats[-train_obs,]
```

(b)

The large regression tree was fit to the data and contained 16 terminal nodes. Below is the plot of the large tree. It follows convention that left corresponds to TRUE and right to FALSE. One example estimate would be that a car seat with a "Bad" or "Medium" shelving location, price between \$75.00 and \$94.50, and community income level above \$47.5k is estimated to sell 8.8 thousands units.

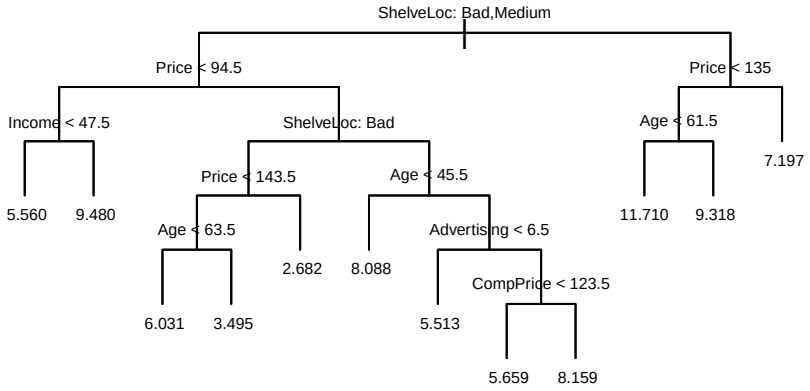


(c)

The test error rate (sum of squared error, known as deviance in the tree library for regression trees) corresponding to the large tree was 668.4 (thousand dollars^2) compared to a training error rate of 615.6 (thousand dollars^2).

(d)

Using cross-validation, the optimal number of terminal nodes was determined to be 12. This level of tree complexity corresponded to a cross-validation error rate (deviance) of 1098.8 (thousand dollars^2). The pruned tree is plotted below.



(e)

The test error rate (SSE/deviance) corresponding to the pruned tree was 650.0 (thousand dollars^2). This is smaller than the error rate of the large tree, 668.4 (thousand dollars^2), and corresponds to a simpler model.

Appendix

```
## Load libraries and set working directory
setwd("/Users/bstan/Documents/UW/Courses/BIOST 527")
rm(list = ls())
library(tidyverse)
library(tidyr)
library(tinytex)
library(class)
library(MASS)
library(ISLR)
library(leaps)
library(gam)
library(tree)

######
# 1a
######
## Create dataset for simulation
set.seed(44)
N = 50
p = 5000
```

```
y = sample(c(rep(1,(N/2)),rep(2,(N/2))), N, replace=FALSE)
X = matrix(rnorm(N*p),N,p)
k_folds = sample(rep(1:5,(N/5)), N, replace=FALSE)

######
# 1b
######
## Perform CV error calculation the "wrong way"
rm(list = ls())
set.seed(44)
N = 50
p = 5000
error_wrong = NULL
for (j in 1:50) {
  ## Define y as even split from two classes
  y = sample(c(rep(1,(N/2)),rep(2,(N/2))), N, replace=FALSE)
  ## Define X from N(0,1)
  X = matrix(rnorm(N*p),N,p)
  ## Find most correlated features
  split_X = split(X, rep(1:ncol(X), each = nrow(X)))
  cor_vals = unlist(map(split_X, function(z) abs(cor(z,y))))
  cor_df = as.data.frame(cor_vals)
  cor_df$index = 1:p
  top_features = cor_df[order(cor_vals,decreasing =TRUE),][1:100,2]
  X_wrong = X[,top_features]
  ## Performed K-fold cross-validation with K=5
  k_folds = sample(rep(1:5,(N/5)), N, replace=FALSE)
  error_wrong_j = NULL
  for (i in 1:5) {
    ## Get predicted values from KNN with k=1
    knn_pred <- knn(X_wrong[k_folds!=i,],X_wrong[k_folds==i,],y[k_folds!=i],k=1)
    conf_matrix <- table(knn_pred,y[k_folds==i])
    error_wrong_j <- c(error_wrong_j,1-(conf_matrix[1,1]+conf_matrix[2,2])/sum(conf_matrix))
  }
  mean(error_wrong_j)
  ## Append error to vector with errors for all simulations
  error_wrong = c(error_wrong,mean(error_wrong_j))
}
mean(error_wrong)

######
# 1c
######
## Perform CV error calculation the "wrong way"
rm(list = ls())
set.seed(44)
N = 50
p = 5000
error_right = NULL
for (j in 1:50) {
  y = sample(c(rep(1,(N/2)),rep(2,(N/2))), N, replace=FALSE)
  X = matrix(rnorm(N*p),N,p)
  k_folds = sample(rep(1:5,(N/5)), N, replace=FALSE)
```

```
  error_right_j = NULL
  ## Loop through folds, finding features and calculating error on ith fold
  for (i in 1:5) {
    X_k = X[k_folds!=i,]
    split_X = split(X_k, rep(1:ncol(X_k), each = nrow(X_k)))
    cor_vals = unlist(map(split_X, function(z) abs(cor(z,y[k_folds!=i]))))
    cor_df = as.data.frame(cor_vals)
    cor_df$index = 1:p
    top_features = cor_df[order(cor_vals,decreasing =TRUE),][1:100,2]
    X_right = X_k[,top_features]
    ## Perform KNN with k=1
    knn_pred <- knn(X_right,X[k_folds==i,top_features],y[k_folds!=i],k=1)
    conf_matrix <- table(knn_pred,y[k_folds==i])
    error_right_j <- c(error_right_j,1-(conf_matrix[1,1]+conf_matrix[2,2])/sum(conf_matrix))
  }
  mean(error_right_j)
  ## Append error to vector with errors for all simulations
  error_right = c(error_right,mean(error_right_j))
}
mean(error_right)

######
# 3a
######
knn_sum <- function(N=100,p=50,K=3){
  X = matrix(runif(N*p,min=0,max=1),N,p) # Generate X from Unif(0,1)
  y = rnorm(N)
  pred_vals = FNN::knn.reg(train=X,test=X,y=y,k=K)$pred # Get predicted values from KNN
  return(y%*%pred_vals)
}
knn_sum(100,50,4)

######
# 3c
######
nk_list = NULL
est_list = NULL
## Loop through values of K perfoming estimation
for (K in 1:200) {
  nk_list = c(nk_list,200/K)
  dof_est = mean(replicate(100,knn_sum(N=200,p=4,K=K)))
  est_list = c(est_list,dof_est)
}
## Plot results
g1 <- ggplot() +
  geom_point(aes(y=nk_list, x=est_list), alpha = 0.4) +
  labs(x='Estimate for Degrees of Freedom', y='N/K') +
  ggtitle("N/K v. Estimate for Degrees of Freedom") +
  theme_bw()
print(g1+geom_abline(slope = 1,intercept = 0))

######
# 4a
```

```
######
rm(list = ls())
set.seed(44)
## Reencode Private
College$Private = ifelse(College$Private == "Yes",1,0)
## Create test and train sets
train_obs = sample.int(n=nrow(College), size=floor(.7*nrow(College)), replace=F)
train_df = College[train_obs,]
test_df = College[-train_obs,]
## Perform forward subset selection
fwd_subset = regsubsets(Outstate~.,
                        data=train_df,
                        method="forward",
                        nvmax=ncol(College))
subset_summary = summary(fwd_subset)
#print(which.min(subset_summary$bic)) # "7"
## Plot results
ggplot() +
  geom_line(aes(y=subset_summary$bic, x=1:17)) +
  labs(x='Number of Variables', y='BIC') +
  ggtitle("BIC v. Number of Variables in Forward Stepwise Selection") +
  theme_bw()

######
# 4b
######
## Select subset of columns from train and test sets
train_df = train_df %>%
  dplyr::select(Outstate,
                Private,
                Top25perc,
                Room.Board,
                Terminal,
                perc.alumni,
                Expend,
                Grad.Rate)
test_df = test_df %>%
  dplyr::select(Outstate,
                Private,
                Top25perc,
                Room.Board,
                Terminal,
                perc.alumni,
                Expend,
                Grad.Rate)

my_plot_hook <- function(x, options)
  paste("\n", knitr::hook_plot_tex(x, options), "\n")
knitr::knit_hooks$set(plot = my_plot_hook)

## Fit GAM with cubic spline functions
college_gam = gam(Outstate~bs(Private,df=5)+bs(Top25perc,df=5)+bs(Room.Board,df=5)+
    bs(Terminal,df=5)+bs(perc.alumni,df=5)+bs(Expend,df=5)+
```

```
    bs(Grad.Rate,df=5),
    data=train_df)
plot(college_gam, se=TRUE,col="blue")

######
# 4c
######
## Fit model to test data and compute error
preds = predict(college_gam,newdata=test_df)
error = test_df$Outstate-preds
error_df = data.frame(actual = test_df$Outstate, predicted = preds, error = error)
error_df$pct_error = error_df$error/error_df$actual
mean(college_gam$residuals^2) # train MSE
mean(error^2) # test MSE
mean(error_df$pct_error)

######
# 4d
######
## View summary
sum_gam = summary(college_gam)
sum_gam$parametric.anova

######
# 5a
######
## Define X and y from training set
y = train_df$Outstate
X = train_df[,-1]
custom_backfit <- function(y,X,num_knots = NULL) {
  ## Set default value for knots to be 2 if not specified
  if (is.null(num_knots)) num_knots = rep(2,ncol(X)) # if no knots passed, use 2 for all
  ## Initialize values
  alpha = mean(y) # initialize alpha
  func_list = matrix(0,nrow(X),ncol(X)) # initialize values of functions
  mse = mean(y^2)
  mse_diff = mse
  ## Perform updating using fitted values from each function
  while (mse_diff > 0) {
    for (j in 1:ncol(X)) {
      lm_j = lm((y-alpha-rowSums(func_list[,-j]))~bs(X[,j],df=num_knots[j]+3))
      func_list[,j] = lm_j$fitted.values-mean(lm_j$fitted.values)
    }
    ## Create convergence criteria with MSE
    mse_prev = mse
    mse = mean((y-rowSums(func_list)-alpha)^2)
    mse_diff = mse_prev - mse
  }
  print(mse)
  return(rowSums(func_list)+alpha)
}

######
```

```
# 5b
######
## Call function and create dataframe with comparison between models
cust_bf_results = custom_backfit(y,X)
comp_df = data.frame(gam = college_gam$fitted.values, custom = cust_bf_results)
comp_df$diff = comp_df$gam-comp_df$custom
## Plot values
g2 <- ggplot() +
  geom_point(aes(y=comp_df$gam, x=comp_df$custom), alpha = 0.4) +
  labs(x='Custom Backfit GAM', y='GAM Library Function') +
  ggtitle("Comparison of Fitted Values from Two GAM Functions") +
  theme_bw()
print(g2+geom_abline(slope = 1,intercept = 0))

######
# 6a
######
rm(list = ls())
set.seed(145)
train_obs = sample.int(n=nrow(Carseats), size=floor(.6*nrow(Carseats)), replace=F)
train_df = Carseats[train_obs,]
test_df = Carseats[-train_obs,]

######
# 6b
######
tree.carseats =tree(Sales~.,train_df)
#summary(tree.carseats)
plot(tree.carseats,type="uniform")
text(tree.carseats, pretty=0, cex=0.5)

######
# 6c
######
pred_train = predict(tree.carseats,train_df)
sum((train_df$Sales-pred_train)^2)

pred_test = predict(tree.carseats,test_df)
sum((test_df$Sales-pred_test)^2)

######
# 6d
######
set.seed(56) #137 works
cv.carseats = cv.tree(tree.carseats)
#cv.carseats
#which(cv.carseats$dev==min(cv.carseats$dev)) # 12 has lowest error

prune.carseats = prune.tree(tree.carseats,best=12)
plot(prune.carseats,type="uniform")
text(prune.carseats, pretty=0, cex=0.5)

######
```

```r
# 6e
######
pred_test = predict(prune.carseats,test_df)
sum((test_df$Sales-pred_test)^2)
```