UNIVERSITY OF MICHIGAN
DATA MINING
STATS415

ASSIGNMENT 6

Author: *Shu* ZHOU
ID: 19342932
Lab Section: 001

November 2, 2020

Stats 415 Assignment 6

19342932 Shu Zhou.

1. $y_i = x_i^T \beta^* + \varepsilon$

$AIC(M) = -2\log L(M) + 2d$

So, we first calculate the maximum likelihood function

From the p.d.f.: $f_i(x) = \frac{1}{\sqrt{2\pi}\,\sigma} e^{-\frac{1}{2\sigma^2}(x - \beta_1 x_{i2} - \cdots \beta_p x_{ip})^2)}$

Hence, the likelihood function is

$$\prod_{i=1}^{n} f_i Y_{(i)} = \left(\frac{1}{\sqrt{2\pi}\,\sigma}\right)^n e^{(-\frac{1}{2\sigma^2}|Y - X\beta|^2)} = L(M).$$

Then $AIC(M) = -2\log L(M) + 2d$

$$= -2\log\left[\left(\frac{1}{\sqrt{2\pi}\,\sigma}\right)^n e^{-\frac{1}{2\sigma^2}|Y - X\beta|^2}\right] + 2d$$

$$= 2d + n \times \left[(-2) \times -\tfrac{1}{2}\log 2\pi + -2 \times -\log\sigma\right] + \frac{1}{-2\sigma^2} \times (-2) \times \underset{\overset{\shortparallel}{|Y - X\beta|^2}}{SSE} + 2d$$

$$= n\left[\log 2\pi + 2\log\sigma\right] + \frac{SSE}{\sigma^2} + 2d.$$

$$= n\left[\log 2\pi + 2\log\sigma\right] + \frac{SSE}{\sigma^2} + 2p \text{ (\# of predictors)}$$

2. $\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})$ subject to $\sum_{j=1}^{p}|\beta_j| \le s$.

a. The number of included variable will steadily increases. Since we are restricting β_j in a smaller region. So the model becomes more flexible and more variables will be included

b. The training error will steadily decreases, according to the explanation in (a), the increase in the flexibility of a model will cause an decrease in training error.

c. The test error will decrease initially and then eventually starts increasing. So at first, our model is underfitting, so if we restrict the β_j coefficients, the test error will decrease. However, this restriction would eventually cause overfitting and increase the test error.

d. The variance of $\hat\beta$ will steadily increase. The model becomes more and more flexible, so the variance would steadily increas

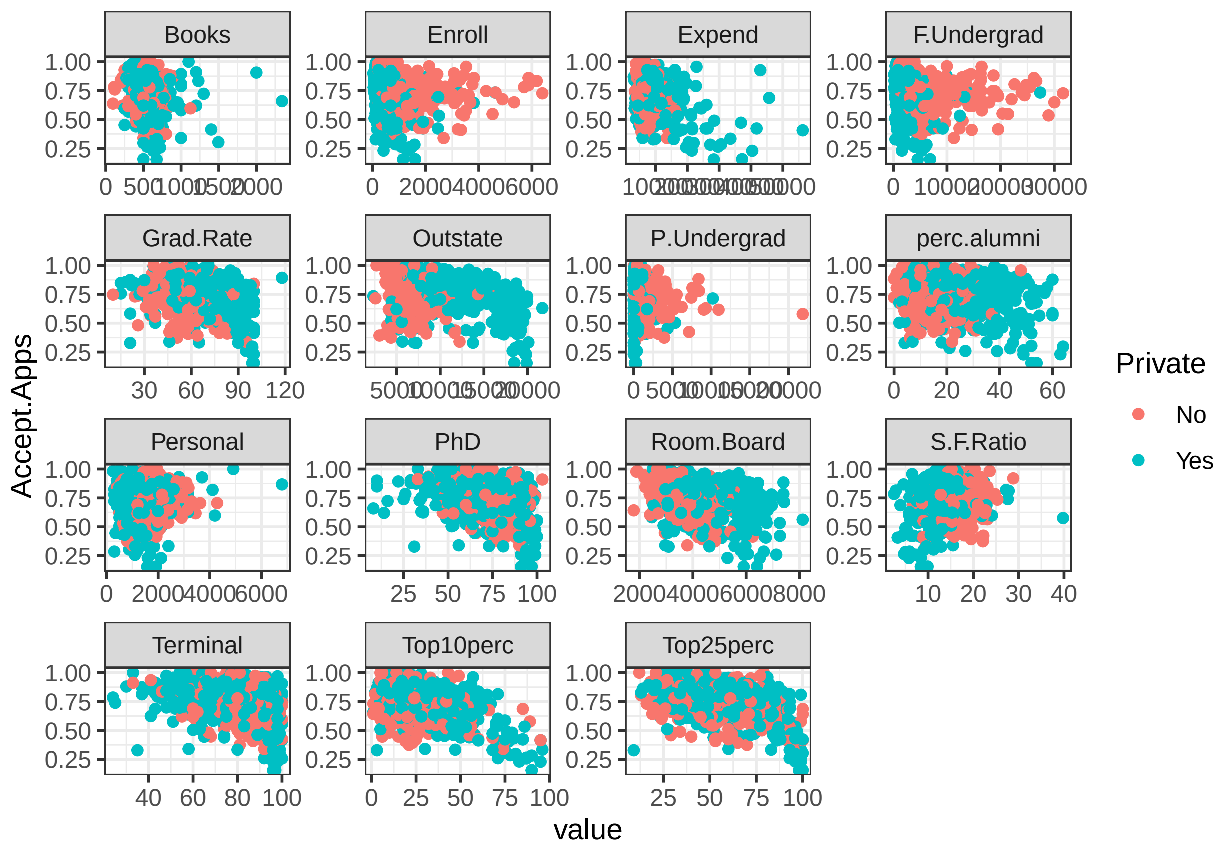
e. The squared bias of $\hat\beta$ will steadily decrease. Since we fit our model closer to their least-square estimate, our model became more and more unbiased.

Q3.

(a)

```r
library(ISLR)
data("College")
College$Accept.Apps<-College$Accept/College$Apps
College<-College[,c(-2,-3)]

#Scatterplot for variables except for private, private shown as color
College %>%
  gather(-Accept.Apps, -Private,  key = "var", value = "value") %>%
  ggplot(aes(x = value, y = Accept.Apps, color = Private)) +
    geom_point() +
    facet_wrap(~ var, scales = "free") +
    theme_bw()
```




```r
#Split the dataset
set.seed(234)
inTrain <- createDataPartition(College$Accept.Apps, p = 0.7, list = FALSE)
training <- College[inTrain,]
testing <- College[-inTrain,]
```

(b) Fit a linear model using least squares on the training set, and report the training and test error obtained, with Accept/Apps as the response variable and all the other variables except Accept and Apps as predictors.

For testing error

```r
linear_model <- lm(Accept.Apps~. ,data=training)
testPrediction <- predict(linear_model, testing)
test_MSE<-mean((testPrediction - testing$Accept.Apps)^2)
test_MSE
```

```
## [1] 0.01350159
```

```r
##Hence, the testing error is 0.01350159
```

For training error

```r
trainPrediction <- predict(linear_model, training)
train_MSE<-mean((trainPrediction - training$Accept.Apps)^2)
train_MSE
```

```
## [1] 0.01422197
```

```r
##Hence, the testing error is 0.01422197
```

(c)

```r
library(leaps)
```

```
## Warning: package 'leaps' was built under R version 4.0.3
```

```r
regfit.full = regsubsets(Accept.Apps~. , data = College, nvmax=ncol(College)-1)
regfit.Summary = summary(regfit.full)
names(regfit.Summary)
```

```
## [1] "which"  "rsq"    "rss"    "adjr2"  "cp"     "bic"    "outmat" "obj"
```

```r
regfit.Summary$rsq
```

```
##  [1] 0.2291301 0.2566215 0.2978282 0.3075737 0.3153663 0.3228179 0.3303928
##  [8] 0.3371194 0.3451705 0.3525302 0.3542739 0.3549540 0.3556430 0.3563261
## [15] 0.3565993 0.3566201
```

#Adjusted R-square

```r
best_adjr2 = which.max(regfit.Summary$adjr2)
best_adjr2 #11
```

```
## [1] 11
```

```r
# We can use the coef() function to see which predictors made the cut
coef(regfit.full, 11)
```

```
##   (Intercept)    PrivateYes        Enroll     Top10perc   P.Undergrad
##  1.072220e+00  7.355125e-02  2.759506e-05 -3.158082e-03 -1.061382e-05
##      Outstate    Room.Board         Books     S.F.Ratio   perc.alumni
##  5.321032e-06 -2.509935e-05 -8.008527e-05 -4.224509e-03  6.559062e-04
##        Expend     Grad.Rate
## -7.195160e-06 -1.409409e-03
```

For testing error

```r
linear_model1 <- lm(Accept.Apps ~ Private+Enroll+Top10perc+P.Undergrad+Outstate+Room.Board+ Books   +S.

testPrediction1 <- predict(linear_model1, testing)
test_MSE1<-mean((testPrediction1 - testing$Accept.Apps)^2)
test_MSE1
```

```
## [1] 0.01327613
```

```
##Hence, the testing error is 0.01327613
```

For training error

```
trainPrediction1 <- predict(linear_model1, training)
train_MSE1<-mean((trainPrediction1 - training$Accept.Apps)^2)
train_MSE1
```

```
## [1] 0.01433002
```

```
##Hence, the testing error is 0.01433002
```

AIC

```
best_cp = which.min(regfit.Summary$cp)
best_cp #11
```

```
## [1] 11
```

```
coef(regfit.full, 11)
```

```
##   (Intercept)    PrivateYes        Enroll     Top10perc   P.Undergrad
##  1.072220e+00  7.355125e-02  2.759506e-05 -3.158082e-03 -1.061382e-05
##      Outstate    Room.Board         Books      S.F.Ratio   perc.alumni
##  5.321032e-06 -2.509935e-05 -8.008527e-05 -4.224509e-03  6.559062e-04
##        Expend     Grad.Rate
## -7.195160e-06 -1.409409e-03
```

For testing error

```
linear_model2 <- lm(Accept.Apps ~ Private+Enroll+Top10perc+P.Undergrad+Outstate+Room.Board+ Books   +S.I

testPrediction2 <- predict(linear_model2, testing)
test_MSE2<-mean((testPrediction2 - testing$Accept.Apps)^2)
test_MSE2
```

```
## [1] 0.01327613
```

```
##Hence, the testing error is 0.01327613
```

For training error

```
trainPrediction2 <- predict(linear_model2, training)
train_MSE2<-mean((trainPrediction2 - training$Accept.Apps)^2)
train_MSE2
```

```
## [1] 0.01433002
```

```
##Hence, the testing error is 0.01433002
```

BIC

```
best_bic = which.min(regfit.Summary$bic)
best_bic
```

```
## [1] 10
```

```r
best_bic #10
```

```
## [1] 10
```

```r
coef(regfit.full, 10)
```

```
##   (Intercept)    PrivateYes        Enroll      Top10perc   P.Undergrad
##  1.078867e+00  7.523625e-02  2.690366e-05 -3.068681e-03 -1.077537e-05
##      Outstate    Room.Board         Books      S.F.Ratio        Expend
##  5.992015e-06 -2.634089e-05 -8.232725e-05 -4.355397e-03 -7.172649e-06
##     Grad.Rate
## -1.309782e-03
```

For testing error

```r
linear_model3 <- lm(Accept.Apps ~ Private+Enroll+Top10perc+P.Undergrad+Outstate+Room.Board+ Books   +S.I

testPrediction3 <- predict(linear_model3, testing)
test_MSE3<-mean((testPrediction3 - testing$Accept.Apps)^2)
test_MSE3
```

```
## [1] 0.01315906
```

```r
##Hence, the testing error is 0.01315906
```

For training error

```r
trainPrediction3 <- predict(linear_model3, training)
train_MSE3<-mean((trainPrediction3 - training$Accept.Apps)^2)
train_MSE3
```

```
## [1] 0.01441521
```

```r
##Hence, the testing error is  0.01441521
```

(d) #Candidate model in (c) The model with smaller test error is the model founded by BIC with 10 variables included.

```r
y_train <- training$Accept.Apps
y_test <- testing$Accept.Apps

one_hot_encoding <- dummyVars(Accept.Apps ~ Private+Enroll+Top10perc+P.Undergrad+Outstate+Room.Board+ Bo
x_train <- predict(one_hot_encoding, training)
x_test <- predict(one_hot_encoding, testing)


linear_fit <- train(x = x_train, y = y_train,
                   method = 'lm',
                   trControl = trainControl(method = 'cv', number = 5),
                  )
```

```
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading
```

```
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading
```

```
(linear_info_test <- postResample(predict(linear_fit, x_test), y_test))
```

```
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

##       RMSE   Rsquared        MAE
## 0.11471296 0.30483774 0.09073556
```

```
test_MSE4<-linear_info_test[1]^2
test_MSE4 #0.01315906
```

```
##       RMSE
## 0.01315906
```

```
(linear_info_train <- postResample(predict(linear_fit, x_train), y_train))
```

```
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

##       RMSE   Rsquared        MAE
## 0.12006336 0.36615176 0.09392975
```

```
train_MSE4<-linear_info_train[1]^2
train_MSE4 #0.01441521
```

```
##       RMSE
## 0.01441521
```

```
college.glm <- glm(Accept.Apps ~ Private+Enroll+Top10perc+P.Undergrad+Outstate+Room.Board+ Books+S.F.Ra
cv.err = cv.glm(College ,college.glm , K = 5)$delta
cv.err
```

```
## [1] 0.01450695 0.01444841
```

Hence, the test error and train error obtained from cross validation is the same as the value we calculated in (c).

#Full_model in (b)

```
y_train <- training$Accept.Apps
y_test <- testing$Accept.Apps

one_hot_encoding <- dummyVars(Accept.Apps ~. , data = training)
x_train <- predict(one_hot_encoding, training)
x_test <- predict(one_hot_encoding, testing)


linear_fit <- train(x = x_train, y = y_train,
                   method = 'lm',
                   trControl = trainControl(method = 'cv', number = 5),
                  )
```

```
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading
```

```
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading
```

```
(linear_info_test <- postResample(predict(linear_fit, x_test), y_test))
```

```
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

##       RMSE   Rsquared        MAE
## 0.11619634 0.28910308 0.09202104
```

```
test_MSE5<-linear_info_test[1]^2
test_MSE5 #0.01350159
```

```
##       RMSE
## 0.01350159
```

```
(linear_info_train <- postResample(predict(linear_fit, x_train), y_train))
```

```
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

##       RMSE   Rsquared        MAE
## 0.11925591 0.37464865 0.09349292
```

```
train_MSE5<-linear_info_train[1]^2
train_MSE5 #0.01422197
```

```
##       RMSE
## 0.01422197
```

```
college.glm <- glm(Accept.Apps ~., data = College)
cv.err = cv.glm(College ,college.glm , K = 5)$delta
cv.err
```

```
## [1] 0.01454782 0.01447501
```

Hence, the test error and train error obtained from cross validation is the same as the value we calculated in (b). **(e)**

```
library(glmnet)
```

```
## Warning: package 'glmnet' was built under R version 4.0.3

## Loading required package: Matrix

##
## Attaching package: 'Matrix'

## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
```

```
## Loaded glmnet 4.0-2
```

```r
set.seed(1)
grid = 10^seq(10, -2, length=100)
ridge.mod = glmnet(x_train, y_train, alpha=0, lambda=grid)

cv.out = cv.glmnet(x_train, y_train, alpha=0, lambda = grid)

bestlam = cv.out$lambda.min
bestlam
```

```
## [1] 0.01
```

```r
# training MSE
ridge.pred_train = predict(ridge.mod, s=bestlam, newx=x_train)
train_MSE_ridge<-mean((ridge.pred_train-y_train)^2)
train_MSE_ridge
```

```
## [1] 0.01432566
```

```r
# test MSE
ridge.pred_test = predict(ridge.mod, s=bestlam, newx=x_test)
test_MSE_ridge<-mean((ridge.pred_test-y_test)^2)
test_MSE_ridge
```

```
## [1] 0.01334943
```

```r
#cross_validation error
cv.out = cv.glmnet(x_train, y_train, alpha=0, lambda = grid, nfolds = 10)
lambda.grid = cv.out$lambda # grid of lambdas used by cv.glmnet()
mses = cv.out$cvm # mean crossvalidated error (MSE) for each lambda (averaged over the 10 folds)
cv_error = mses[which(lambda.grid == bestlam)] # this is the crossvalidated error (MSE)
print(cv_error)
```

```
## [1] 0.01538698
```

(f)

```r
lasso.mod = glmnet(x_train, y_train, alpha=1, lambda=grid)
set.seed(1)
cv.out = cv.glmnet(x_train, y_train, alpha=1)
# best lambda
bestlam = cv.out$lambda.min
bestlam
```

```
## [1] 0.0009319414
```

```r
# training error
lasso.pred_train = predict(lasso.mod,s=bestlam,newx=x_train)
train_MSE_lasso<-mean((lasso.pred_train-y_train)^2)
train_MSE_lasso
```

```
## [1] 0.01554418
```

```r
# test error
lasso.pred_test = predict(lasso.mod,s=bestlam,newx=x_test)
test_MSE_lasso<-mean((lasso.pred_test-y_test)^2)
test_MSE_lasso
```

```
## [1] 0.01405796
```

```
cv.out = cv.glmnet(x_train, y_train, alpha=1, lambda = grid, nfolds = 10)
lambda.grid = cv.out$lambda =
mses = cv.out$cvm =
cv_error = mses[which(lambda.grid == bestlam)]
print(cv_error)
```

```
## numeric(0)
```

(g)

```
train_MSE_best_reduced<-train_MSE3
test_MSE_best_reduced<-test_MSE3


models = c("Best reduced OLS", "Ridge Regression", "Lasso")
train_err = c(
train_MSE_best_reduced,
train_MSE_ridge,
train_MSE_lasso)

test_err = c(
test_MSE_best_reduced,
test_MSE_ridge,
test_MSE_lasso
)
results = data.frame(
models,
train_err,
test_err
)
colnames(results) = c("Model", "Train MSE", "Test MSE")
print(results)
```

```
##              Model  Train MSE   Test MSE
## 1 Best reduced OLS 0.01441521 0.01315906
## 2 Ridge Regression 0.01432566 0.01334943
## 3            Lasso 0.01554418 0.01405796
```

In all, we can predict the acceptance rate with approximately 0.0135 testing MSE. The testing MSE with ridge regression is the lowest, best reduced OLS model is the second lowest and the Lasso regression is the highest. In this dataset, I would choose ridge regression, since it reports the lowest training and test error.