APPLIED STATISTICS

2023-12-04

GROUP MEMBERS:


```
"*SWETHA YANAMANDHALLA (811294573)*"

"*SAI LAXMI PRIYANKA GANNAVARAPU (811283553)*"

"*AJAYCHARY KANDUKURI (811294510)*"
```


CONTRIBUTIONS(GRADE :0-5)

#SAI LAXMI PRIYANKA GANNAVARAPU (811283553) : 5

#AJAYCHARY KANDUKURI (811294510):5

#1)

```r
library(alr4)
```

```
## Loading required package: car

## Loading required package: carData

## Loading required package: effects

## lattice theme set by effectsTheme()
## See ?effectsTheme for details.
```

```r
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following object is masked from 'package:car':
##
##     recode
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
data_frame = Downer[ c("calving", "daysrec", "ck", "ast", "urea", "pcv")]
data_frame = na.omit(df)
data_frame
```

```
## function (x, df1, df2, ncp, log = FALSE)
## {
##     if (missing(ncp))
##         .Call(C_df, x, df1, df2, log)
##     else .Call(C_dnf, x, df1, df2, ncp, log)
## }
## <bytecode: 0x0000024093ffa950>
## <environment: namespace:stats>
```

```
library(glmnet)
```

```
## Loading required package: Matrix

## Loaded glmnet 4.1-8
```

```
set.seed(123)
train_index = sample(seq_len(nrow(Downer)), 0.8 * nrow(Downer))
train_data = Downer[train_index, ]
test_data = Downer[-train_index, ]

logistic_model = glm(outcome ~ calving + daysrec + ck + ast + urea + pcv,
                      data = train_data,
                      family = binomial)

predictions = predict(logistic_model, newdata = test_data, type = "response")
threshold = 0.5
predicted_classes = ifelse(predictions > threshold, 1, 0)
confusion_matrix = table(test_data$outcome, predicted_classes)
confusion_matrix
```

```
##          predicted_classes
##             0  1
##   died     25  5
##   survived  4  7
```

```
accuracy = sum(diag(confusion_matrix)) / sum(confusion_matrix)
accuracy
```

```
## [1] 0.7804878
```

#2)

```r
library(MASS)
```

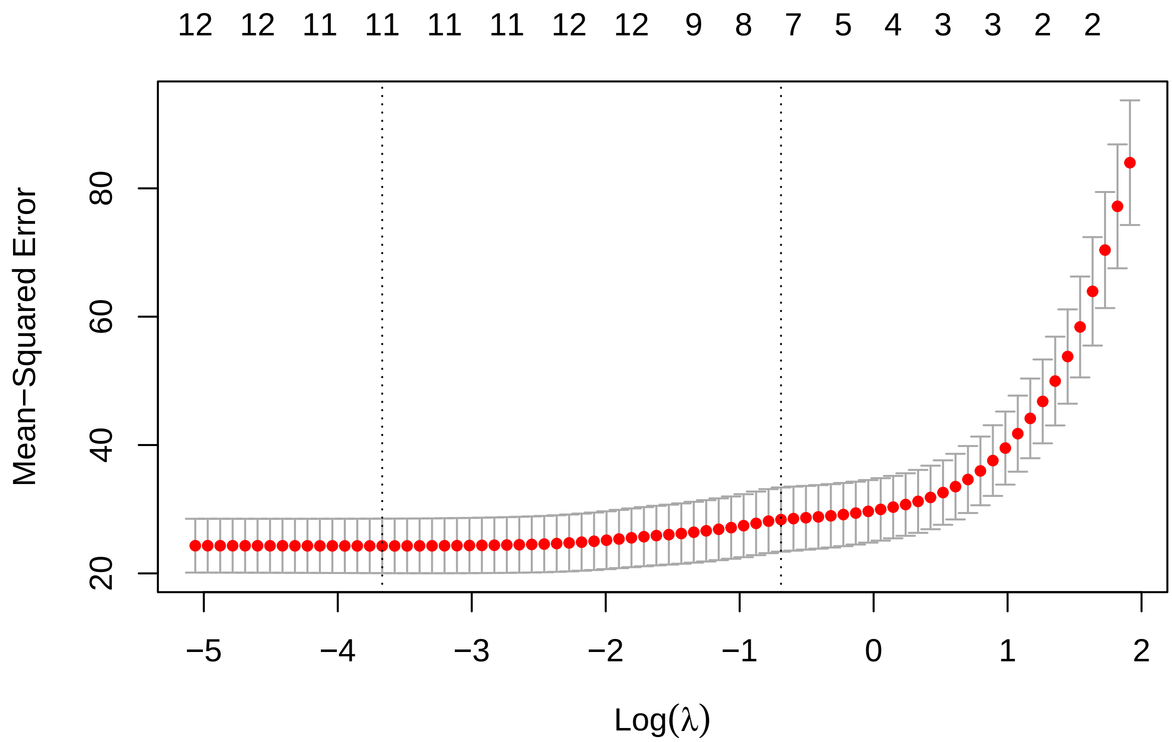
```
##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select
```

```r
data(Boston)
head(Boston)
```

```
##      crim zn indus chas   nox    rm  age    dis rad tax ptratio  black lstat
## 1 0.00632 18  2.31    0 0.538 6.575 65.2 4.0900   1 296    15.3 396.90  4.98
## 2 0.02731  0  7.07    0 0.469 6.421 78.9 4.9671   2 242    17.8 396.90  9.14
## 3 0.02729  0  7.07    0 0.469 7.185 61.1 4.9671   2 242    17.8 392.83  4.03
## 4 0.03237  0  2.18    0 0.458 6.998 45.8 6.0622   3 222    18.7 394.63  2.94
## 5 0.06905  0  2.18    0 0.458 7.147 54.2 6.0622   3 222    18.7 396.90  5.33
## 6 0.02985  0  2.18    0 0.458 6.430 58.7 6.0622   3 222    18.7 394.12  5.21
##   medv
## 1 24.0
## 2 21.6
## 3 34.7
## 4 33.4
## 5 36.2
## 6 28.7
```

```r
library(glmnet)
set.seed(123)
predictors = colnames(Boston)[colnames(Boston) != "medv"]
x = scale(as.matrix(Boston[, predictors]))
y = Boston$medv
lasso_model = cv.glmnet(x, y, alpha = 1)
plot(lasso_model)
```




```r
optimal_lambda = lasso_model$lambda.min
optimal_lambda
```

```
## [1] 0.02551743
```

```r
final_model = glmnet(x, y, alpha = 1, lambda = optimal_lambda)
coef(final_model)
```

```
## 14 x 1 sparse Matrix of class "dgCMatrix"
##                     s0
## (Intercept) 22.5328063
## crim        -0.8584071
## zn           0.9778999
## indus        .
## chas         0.6819302
## nox         -1.9019750
## rm           2.7114053
## age          .
## dis         -2.9595128
## rad          2.2581441
## tax         -1.7022331
## ptratio     -2.0180834
## black        0.8269258
## lstat       -3.7297304
```

```r
lasso_coefficients = coef(final_model)
selected_features = rownames(lasso_coefficients)[lasso_coefficients[, 1] != 0]

predictions = predict(final_model, newx = x, s = optimal_lambda)
mse = mean((predictions - y)^2)
mse
```

```
## [1] 21.92794
```

```r
selected_coefficients = coef(final_model, s = optimal_lambda)
selected_coefficients
```

```
## 14 x 1 sparse Matrix of class "dgCMatrix"
##                     s1
## (Intercept) 22.5328063
## crim        -0.8584071
## zn           0.9778999
## indus        .
## chas         0.6819302
## nox         -1.9019750
## rm           2.7114053
## age          .
## dis         -2.9595128
## rad          2.2581441
## tax         -1.7022331
## ptratio     -2.0180834
## black        0.8269258
## lstat       -3.7297304
```

```r
selected_features = rownames(selected_coefficients)[selected_coefficients[, 1] != 0]
feature_importance = lasso_coefficients[rownames(lasso_coefficients) %in% selected_features, ]
feature_importance
```

```
## (Intercept)        crim          zn        chas         nox          rm
##  22.5328063  -0.8584071   0.9778999   0.6819302  -1.9019750   2.7114053
##         dis         rad         tax     ptratio       black       lstat
##  -2.9595128   2.2581441  -1.7022331  -2.0180834   0.8269258  -3.7297304
```

INTERPRETATION: Selected features like low crime rates, proximity to the Charles River, and increased residential land positively impact predicted house prices in Boston suburbs, while factors such as higher nitric oxide levels and longer commutes contribute to lower predicted prices.

#3)

```r
data(faithful)
faithful_data = faithful
X = faithful_data$waiting
y = faithful_data$eruptions
r_squared_values = vector("numeric", 4)
for (degree in 1:4) {
  poly_model = lm(y ~ poly(X, degree, raw = TRUE))
```

```r
  folds = cut(seq(1, length(y)), breaks = 10, labels = FALSE)
  cv_r_squared = sapply(1:10, function(i) {
    val_index = which(folds == i, arr.ind = TRUE)
    val_data = data.frame(X = X[val_index], y = y[val_index])
    train_data = data.frame(X = X[-val_index], y = y[-val_index])
    model = lm(y ~ poly(X, degree, raw = TRUE), data = train_data)
    predictions = predict(model, newdata = val_data)
    1 - sum((val_data$y - predictions)^2) / sum((val_data$y - mean(val_data$y))^2)
  })
  avg_r_squared = mean(cv_r_squared)
  r_squared_values[degree] = avg_r_squared
}
best_degree = which.max(r_squared_values)
best_avg_r_squared = max(r_squared_values)
best_degree
```

```
## [1] 4
```

```r
best_avg_r_squared
```

```
## [1] 0.8654005
```