STAT 432 Homework-2

Sharvi Tomar (stomar2)

30/08/21

Contents

Question 1 (linear regression review)

Let's used the real estate data as an example. The data can be obtained from the course website.

a. Construct a new categorical variable called season into the real estate dataset. You should utilize the original variable date to perform this task and read the definition provided in our lecture notes. The season variable should be defined as: spring (Mar - May), summer (Jun - Aug), fall (Sep - Nov), and winter (Dec - Feb). Show a summary table to demonstrate that your variable conversion is correct.

b. Split your data into two parts: a testing data that contains 100 observations, and the rest as training data. For this question, you need to set a random seed while generating this split so that the result can be replicated. Use your UIN as the random seed. Report the mean price of your testing data and training data, respectively.

c. Use the training dataset to perform a linear regression. The goal is to model price with season, age, distance and stores. Then use this model to predict the testing data using the predict() function. Calculate the training data mean squared error (training error): Training Error=1ntrain i Train(yi−ŷ i)2 and prediction mean squared error (testing error) using the testing data, defined as: Testing Error=1ntest i Test(yi−ŷ i)2

d. For this last part, we will explicitly calculate the parameter estimates using the linear regression solution (for details, see our lecture notes): ˆ=(XTX)−1XTy To perform this calculation, you need to properly define the data matrix X and the outcome vector y from just your training data. One thing to be careful here is that the data matrix X should contain a column of 1 to represent the intercept term. Construct such a data matrix with season, age, distance and stores, while making sure that the season variable is using a dummy coding. Should your dummy variable be three columns or four columns if an intercept is already included? and Why? After obtaining the parameter estimates, validate your results by calculating the training error of your model, and compare it with the value obtained from the previous question.

```r
realestate = read.csv("realestate.csv", row.names = 1)
## Creating a var 'new' with decimal values from 'date' var
realestate$new=realestate$date - floor(realestate$date)
## Creating seasons
realestate$new[realestate$new >= .250 & realestate$new < .417] <- "spring"
```

```r
realestate$new[realestate$new >= .500 & realestate$new < .667] <- "summer"
realestate$new[realestate$new >= .750 & realestate$new < .917] <- "fall"
realestate$new[realestate$new >= .000 & realestate$new < .167] <-"winter"
## Changing data type to factor
realestate$new=as.factor(realestate$new)
## Renaming the 'new' var as 'season'
colnames(realestate)[8]<-"season"
## Table summary of 'season' var
table(realestate$season)
```

```
##
##   fall spring summer winter
##     96    119    100     99
```

```r
## Spliting data into testing data (100 observations) & training data (rest of the observations)
require(caTools)
```

```
## Loading required package: caTools
```

```r
set.seed(667346304)
sample = sample.split(realestate$price, SplitRatio = 100/nrow(realestate))
train = subset(realestate, sample == FALSE)
test  = subset(realestate, sample == TRUE)

## Reporting the mean price of testing data and training data
mean(train$price)
```

```
## [1] 37.95828
```

```r
mean(test$price)
```

```
## [1] 38.049
```

```r
## Use the training dataset to perform a linear regression
model=lm(price~season+age+distance+stores, data=train)
## Predicting the testing data
y_pred=predict(model,test)
y_pred
```

```
##        14        15        24        25        46        52        54        56
## 32.107079 41.026630 47.801885 35.624445 41.000054 27.360684 43.500536 25.800742
##        59        60        63        66        69        72        75        91
## 15.022659 44.346091 27.525388 42.288973 40.621788 34.652811 48.062035 39.930914
##        92        96        98        99       104       106       107       108
## 34.063228 43.432129 36.608445 44.014531 46.094392 45.636324 47.619411 34.652791
##       118       120       122       129       133       135       137       142
## 17.555387 46.052231 45.726717 39.435479 40.124724 43.439061 41.008921 37.724451
##       146       150       153       156       157       159       165       168
## 43.076815 45.416506 34.704326 18.128274 31.329919 44.488994 39.248878 44.238178
##       174       179       181       182       183       184       187       198
```

```
## 35.613113 41.162855 15.067860 48.995397 34.468189 19.607193 30.053506 43.741565
##       200       203       206       221       222       234       239       241
## 46.156833 30.389101 36.122206 43.858048 34.088995 42.418762 36.518453 34.000981
##       248       250       254       257       263       274       277       278
## 32.694898  2.634488 31.181214 41.650852 40.397513 40.837878 42.174549 30.190451
##       279       281       285       287       291       293       297       305
## 46.094392 49.377854 42.444017 48.697513 31.415253 42.941055 35.482370 33.526356
##       307       308       314       316       318       347       350       352
## 42.618706 20.924081 47.100682 32.578647 34.480810 33.648340 43.483665 30.903072
##       356       357       358       361       363       365       366       369
## 46.749426 37.615585 52.618208 48.754047 39.001393 39.709157 28.198059 38.571055
##       373       374       378       385       387       390       392       394
## 42.327351 43.462522 52.017951 14.087538 39.248878 42.090308 35.150055 38.838680
##       402       403       409       414
## 33.535277 37.126909 30.936984 54.377999
```

```
## Training data mean squared error
training_error<-mean((train$price - predict(model,train)) ^ 2)
training_error
```

```
## [1] 83.99173
```

```
## Testing data mean squared error
mean((test$price - y_pred) ^ 2)
```

```
## [1] 79.42543
```

```
## Calculating beta parameter ##

## Adding a column to represent the intercept term
intercept = rep(1,nrow(train))
train = cbind(intercept,train)

# Creating dummy coding for 'season' var
train$season_summer=ifelse(train$season=="summer",1,0)
train$season_fall=ifelse(train$season=="fall",1,0)
train$season_winter=ifelse(train$season=="winter",1,0)

# Dropping vars 'longitude', 'latitude', 'date', 'season'
train<-train[-c(2,6,7,9)]
# Re-ordering vars
train<-train[c(1,2,3,4,6,7,8,5)]

# Creating X matrix and y matrix
X=as.matrix(train[1:nrow(train),1:7])
y=as.matrix(train[1:nrow(train),8])

# Taking transpose of matrix X -- X_t
X_t <- t(X)
# Taking inverse of product of X_t with X -- prod_inverse
prod_inverse <- solve(X_t%*%X)
# Taking product of 'prod_inverse', X_t, and y -- beta_parameter
```

```r
beta_parameter = prod_inverse%*%X_t%*%y

# Calculating the training error of model using beta parameter
y_pred_param=X%*%beta_parameter
mean((train$price - y_pred_param) ^ 2)
```

```
## [1] 83.99173
```

```r
# Training data mean squared error
training_error
```

```
## [1] 83.99173
```

The regular matrix inverse, (X´X)-1, of the X´X matrix only exists as long as there is no exact linear relationship among the columns of the X matrix. If any column of the X matrix can be expressed as an exact linear combination of any of the remaining columns of X, then perfect multicollinearity is said to exist and the determinant of the X´X matrix is equal to zero. This means that the inverse, (X´X)-1 will not exist since calculating the inverse involves a division by the determinant, which, in the case of perfect multicollinearity, means dividing by zero.

If we include 4 dummy variables for 'season', then the intercept column of X matrix would be an exact linear combination of the season_summer(Su), season_spring(Sp) and 'season_winter(Sw)' and 'season_fall(Sf)' (had we included it) columns -> Su + Sp + Sw + Sf = 1 implying the intercept column is equal to the sum of the 4 columns. Hence, would be a case of perfect multicollinearity so we should take 3 dummy variables when taking intercept value.

Question 2 (model selection)

For this question, use the original six variables defined in the realestate data, and treat all of them as continuous variables. However, you should keep your training/testing split. Fit models using the training data, and when validating, use the testing data.

a. Calculate the Marrows' Cp criterion using the full model, i.e., with all variables included. Compare this result with a model that contains only age, distance and stores. Which is the better model based on this criterion? Compare their corresponding testing errors. Does that match your expectation? If yes, explain why you expect this to happen. If not, what could be the causes?

b. Use the best subset selection to obtain the best model of each model size. Perform the following:

- Report the matrix that indicates the best model with each model size.
- Use the AIC and BIC criteria to compare these different models and select the best one respectively. Use a plot to intuitively demonstrate the comparison of different model sizes.
- Report the best model for each criteria. Are they the same?
- Based on the selected variables of these two best models, calculate and report their respective prediction errors on the testing data.
- Which one is better? Is this what you expected? If yes, explain why you expect this to happen. If not, what could be the causes?

c. Use a step-wise regression with AIC to select the best model. Clearly state:

- What is your initial model?

- What is the upper/lower limit of the model?
- Are you doing forward or backward?

Is your result the same as question b)? Provide a brief discussion about their similarity or dissimilarity and the reason for that.

```r
realestate2 = read.csv("realestate.csv", row.names = 1)
set.seed(667346304)
sample2 = sample.split(realestate2$price, SplitRatio = 100/414)
train2 = subset(realestate2, sample2 == FALSE)
test2  = subset(realestate2, sample2 == TRUE)
```

```r
model_full=lm(price~., data=train2)
model_sub=lm(price~age+distance+stores, data=train2)

# Calculating the Cp criterion for the full model
p_full = 7                                                # number of variables (including intercept)
n = nrow(train2)
RSS_full = sum(residuals(model_full)^2)                   # obtain residual sum of squares
Cp_full = RSS_full + 2*p_full*summary(model_full)$sigma^2   # use the formula to calculate the Cp crite

# Calculating the Cp criterion for the sub model
p_sub = 4                                                 # number of variables (including intercept)
n = nrow(train2)
RSS_sub = sum(residuals(model_sub)^2)                     # obtain residual sum of squares
Cp_sub = RSS_sub + 2*p_sub*summary(model_sub)$sigma^2       # use the formula to calculate the Cp

Cp_full
```

```
## [1] 25758.16
```

```r
Cp_sub
```

```
## [1] 28383.69
```

The better model based on Mallow's Cp criterion is the one with lower Cp value i.e. full model i.e., with all variables included.

```r
# Testing errors for the full model and sub model
mean((test2$price - predict(model_full, test2)) ^ 2)
```

```
## [1] 74.17226
```

```r
mean((test2$price - predict(model_sub, test2)) ^ 2)
```

```
## [1] 74.69912
```

The testing errors of model_full is less than testing errors for model_sub which has a greater Cp value. Since the model (selected as per Cp criterion) gives small training errors hence, we expect it to give small testing errors (provided the model is not overfit–which has been handled by Cp by penalizing for the number of variables added to the model).

```r
library(leaps)
RSSleaps = regsubsets(x = as.matrix(realestate2[, -7]), y = realestate2[, 7])
summary(RSSleaps, matrix=T)
```

```
## Subset selection object
## 6 Variables  (and intercept)
##           Forced in Forced out
## date          FALSE      FALSE
## age           FALSE      FALSE
## distance      FALSE      FALSE
## stores        FALSE      FALSE
## latitude      FALSE      FALSE
## longitude     FALSE      FALSE
## 1 subsets of each size up to 6
## Selection Algorithm: exhaustive
##          date age distance stores latitude longitude
## 1  ( 1 ) " "  " " "*"      " "    " "      " "
## 2  ( 1 ) " "  " " "*"      "*"    " "      " "
## 3  ( 1 ) " "  "*" "*"      "*"    " "      " "
## 4  ( 1 ) " "  "*" "*"      "*"    "*"      " "
## 5  ( 1 ) "*"  "*" "*"      "*"    "*"      " "
## 6  ( 1 ) "*"  "*" "*"      "*"    "*"      "*"
```

```r
library(leaps)
sumleaps = summary(RSSleaps, matrix = T)
modelsize=apply(sumleaps$which,1,sum)
AIC = n*log(sumleaps$rss/n) + 2*modelsize;
BIC = n*log(sumleaps$rss/n) + modelsize*log(n);
cbind("Our BIC" = BIC, "Our AIC"=AIC)
```

```
##    Our BIC  Our AIC
## 1 1547.104 1539.605
## 2 1527.224 1515.976
## 3 1503.917 1488.920
## 4 1488.387 1469.640
## 5 1485.851 1463.355
## 6 1491.550 1465.304
```

The best model as per BIC and AIC criteria is same and is the one with 5 variables 'date', 'age', 'distance', 'stores', 'latitude' and an intercept additionally.

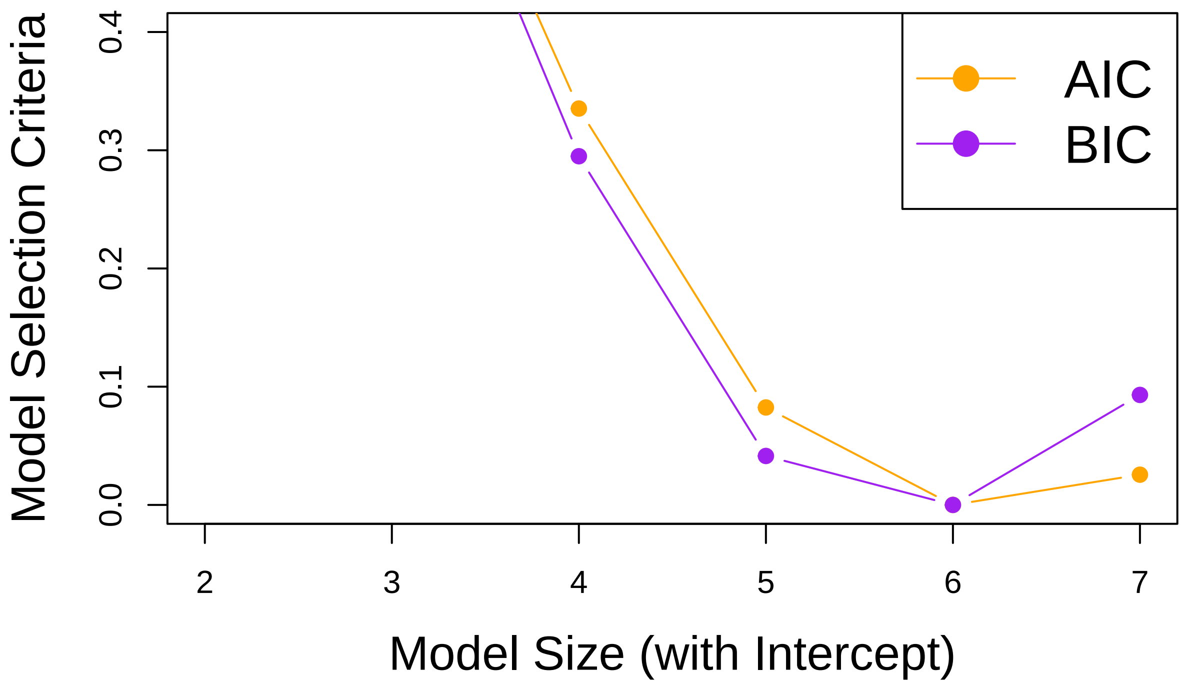
```r
inrange <- function(x) { (x - min(x)) / (max(x) - min(x)) }

    BIC = inrange(BIC)
    AIC = inrange(AIC)

    plot(range(modelsize), c(0, 0.4), type="n",
         xlab="Model Size (with Intercept)",
         ylab="Model Selection Criteria", cex.lab = 1.5)

    points(modelsize, AIC, col = "orange", type = "b", pch = 19)
    points(modelsize, BIC, col = "purple", type = "b", pch = 19)
```

```r
    legend("topright", legend=c("AIC", "BIC"),
           col=c("orange", "purple"),
           lty = rep(1, 3), pch = 19, cex = 1.7)
```



The best model for each criteria is the model with size=6 (including intercept).

```r
## Best model 1-- and its testing error
best_model1=lm(price~date+age+latitude+stores+distance, data=train2)
y_pred_best1=predict(best_model1, test2)
mean((test2$price - y_pred_best1) ^ 2)
```

```
## [1] 73.90245
```

Since the best model as per BIC value and as per AIC value is the same one, we can't make comparison between them.

```r
step(lm(price~1, data=train2), scope=list(upper=model_full, lower=~1), direction="forward",
     k = 2, trace=0)
```

```
##
## Call:
## lm(formula = price ~ distance + age + latitude + stores + date,
##     data = train2)
##
## Coefficients:
```

```
## (Intercept)     distance          age     latitude       stores         date
##  -1.832e+04   -4.225e-03   -2.852e-01    2.429e+02    1.024e+00    6.111e+00
```

Initial model is the intercept model.

The upper limit is the full model with all 6 variables: 'date', 'age', 'distance', 'stores', 'latitude' and 'longitude'. The lower limit of the model is the intercept model.

Doing forward movement i.e. forward selection- testing the addition of each variable, adding the variable (if any) whose inclusion gives the most statistically significant improvement of the fit, and repeating this process until none improves the model to a statistically significant extent.

The best model obtained from b) is same as obtained with step-wise regression.

However, this may not always be the case.The results of the step-wise regression approach depend on initial model and 'direction' set to it.

The best subset selection is better because it considers all possible candidates, which step-wise regression may stuck at a sub-optimal model, while adding and subtracting any variable do not benefit further. Hence, the results of step-wise regression may be unstable. On the other hand, best subset selection not really feasible for high-dimensional problems because of the computational cost.