Computing the LASSO

Ivan

2/6/2023

Question 1:

Part a

Will the lasso always select more variables than the adaptive lasso? It depends. If we are assuming orthogonal design and a "medium" λ across both estimators, then the lasso and adaptive lasso choose the same variables. If we have a "small" λ across both estimators, then the adaptive lasso penalizes heavier less irrelevant variables and decreases the bias for more relevant variables. For a large value of λ, the AL actually chooses **less** variables than the normal lasso

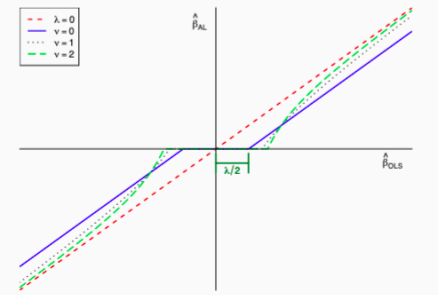


Figure 2: Thresholds for different ν, "small" λ

Part b

Do we always prefer the adaptive lasso over the lasso? Well, the adaptive lasso always gives us all oracle properties, while the normal lasso can never do this. But perhaps we are not interested in variable selection, but in prediction. In addition, as noted in Zhou (2006), when we have a low signal to noise ratio (relatively more noise than signal), then the lasso performs better than the adaptive lasso.

Part c

Do we prefer information criteria or cross validation? It depends on the use case. Using cross validation, we lose some information because we dedicate some portions to training and testing. But this could be much better in a prediction setting. Information criteria may be better when we have less samples to choose from.

Part d

$$y_i = \sum_{j=1}^{p} \beta_j x_{i,j} + \varepsilon_i$$

where $\beta_j = \alpha^j$ for some $0 < \alpha < 1$. Is β weakly sparse?

Then we know that α tends to zero at some rate, depending on the form of α. Suppose we had strong sparsity. Then we would have

$$||\beta_j||_0^0 = \sum_{j=1}^{k} |\beta_j|^0 = \sum_{j=1}^{k} \mathbb{1}(|\beta_j| > 0) = o\left(\sqrt{\frac{n}{\ln p}}\right)$$

This then means that the cardinality of α^j would need to be of smaller order than $\sqrt{\frac{n}{\ln p}}$. Not true. Now suppose weak sparsity.

$$||\beta_j||_1 = \sum_{j=1}^{p} |\beta_j| = o\left(\sqrt{\frac{n}{\ln p}}\right)$$

Then we have the sum of the absolute values should not be too large. This is the case when

$$\sum_{j=1}^{p} |\beta_j| = \sum_{j=1}^{p} |\alpha^j| = o\left(\sqrt{\frac{n}{\ln p}}\right) \Rightarrow \sqrt{\frac{\ln p}{n}} \sum_{j=1}^{p} |\alpha^j| \to 0$$

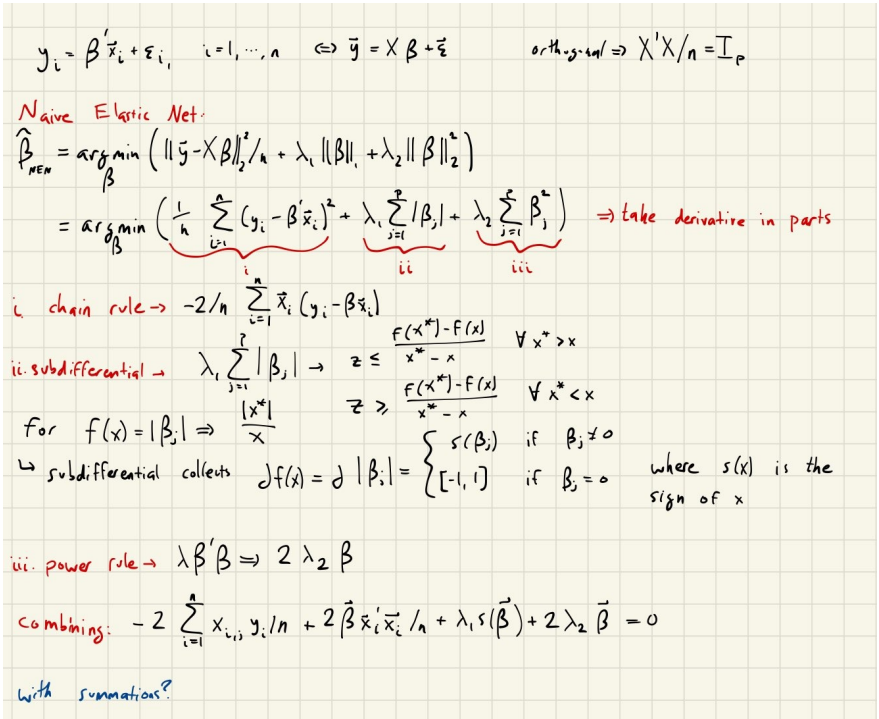
Because even as p increases, we have the sum converging to zero. Unlike in cardinality where we have a sum of ones.

We have a geometric sum that converges to a finite value

Question 2:

Part a

$$\hat{\beta}^* = \arg\min_{\beta} \left(||\mathbf{y} - \mathbf{X}\beta||_2^2/n + \lambda_1||\beta||_1 + \lambda_2||\beta||_2^2\right)$$



Part b

$$-2\sum_{i=1}^{n} x_{ij} y_i / n + 2\beta_j + \lambda_1 \, sign(\beta_j) + 2\lambda_2 \beta_j = 0 \qquad \text{(solve for } \beta\text{)}$$

$$2\beta_j (1+\lambda_2) + \lambda_1 s(\beta_j) = 2\sum_{i=1}^{n} x_{ij} y_i / n$$

$$2\beta_j (1+\lambda_2) = \underbrace{2\sum_{i=1}^{n} x_{ij} y_i / n}_{\text{OLS under orth.}} - \lambda_1 s(\beta_j)$$

$$2\beta_j (1+\lambda_2) = \hat{\beta}_{OLS,j} - \lambda_1 s(\beta_j) \Rightarrow \text{similar to our LASSO penalty:}$$

$$\begin{cases} x'y/n - \lambda/2 & \text{if } x'y/n > \lambda/2 \\ x'y/n + \lambda/2 & \text{if } x'y/n < -\lambda/2 \\ 0 & \text{if } -\lambda/2 \le x'y/n \le \lambda/2 \end{cases}$$

under orthogonal design:

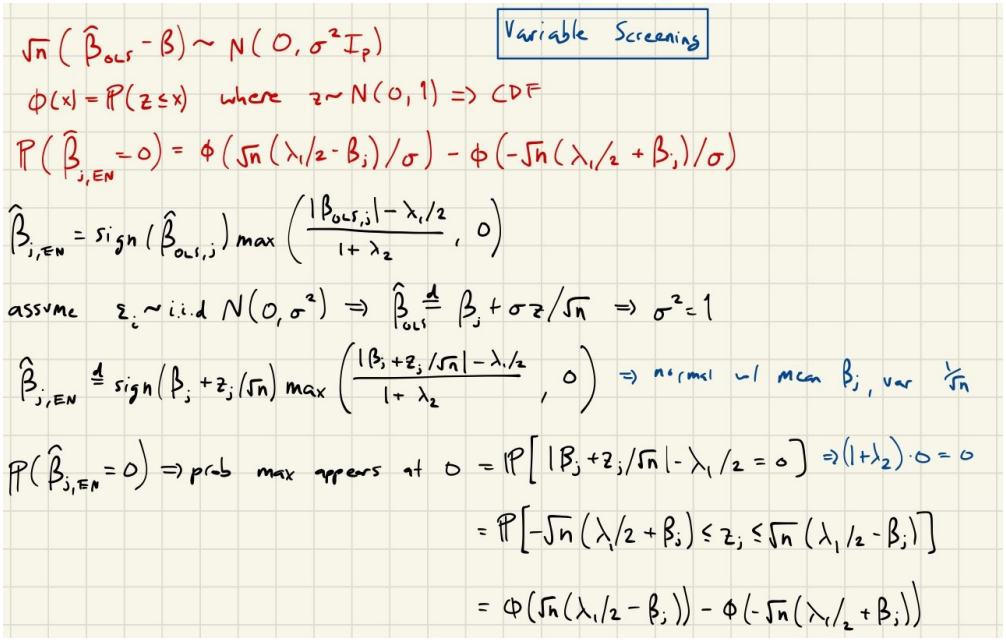
$$\beta_j = sign(\hat{\beta}_{OLS,j}) \max\left(\frac{|\beta_{OLS,j}| - \lambda_1/2}{1+\lambda_2}, 0 \right)$$

where $2(1+\lambda_2)$ go to other side

Part c

Why do we scale EN by $(1 + \lambda/2)$? We do this to improve **prediction performance**. This reduces the bias in the naive elastic net and offsets the double shrinkage. Double shrinkage does not help reduce the variance and introduces unnecessary extra bias. We end up with the lasso under orthogonality!

Part d



Part e

i. For the Naive elastic net, we are looking for consistent estimation of $\hat{\beta}_{NEN}$, such that for every β and $\varepsilon > 0$,

$$\lim_{n\to\infty} \mathbb{P}\left(\left|\hat{\beta}_{NEN} - \beta\right| < \varepsilon\right) = 1 \quad \text{or} \quad \lim_{n\to\infty} \mathbb{P}\left(\left|\hat{\beta}_{NEN} - \beta\right| \geq \varepsilon\right) = 0$$

For fixed p and $n \to \infty$, we need

$$\frac{|\hat{\beta}_{OLS}|}{1+\lambda_2} - \frac{\lambda_1/2}{1+\lambda_2} \xrightarrow{p} \beta$$

In other words,

$$\lambda_1 \to 0, \quad \lambda_2 \to 0$$

For both $p \to \infty$ and $n \to \infty$, we need assumptions on sparsity. Suppose we have weak sparsity, or in other words,

$$||\beta||_1 = \sum_{j=1}^{p} |\beta_j| = o\left(\sqrt{\frac{n}{\ln p}}\right)$$

Then our λ_1 needs to grow at a rate opposite of this, $\sqrt{\frac{\ln p}{n}}$. By similar logic, we can take the ℓ_2-norm as

$$||\beta||_2^2 = \sum_{j=1}^{p} \beta_j^2 = o\left(\frac{n}{\ln p}\right)$$

Then our λ_2 needs to grow at a rate opposite of this, $\frac{\ln p}{n}$.

For the normal elastic net, we multiply the whole parameter by $(1+\lambda_2)$, so in order to correct for this, our λ_1 now takes the form $\lambda_1 + \lambda_2\lambda_1$ and our λ_2 takes the form $\lambda_2 + \lambda_2^2$. The higher polynomial takes over, which in the first case is $\lambda_1\lambda_2$ and the second case is λ_2^2. We then have to cancel the terms to grow at an opposite rate, solving the system

$$\lambda_1\lambda_2 = \sqrt{\frac{n}{\ln p}}$$

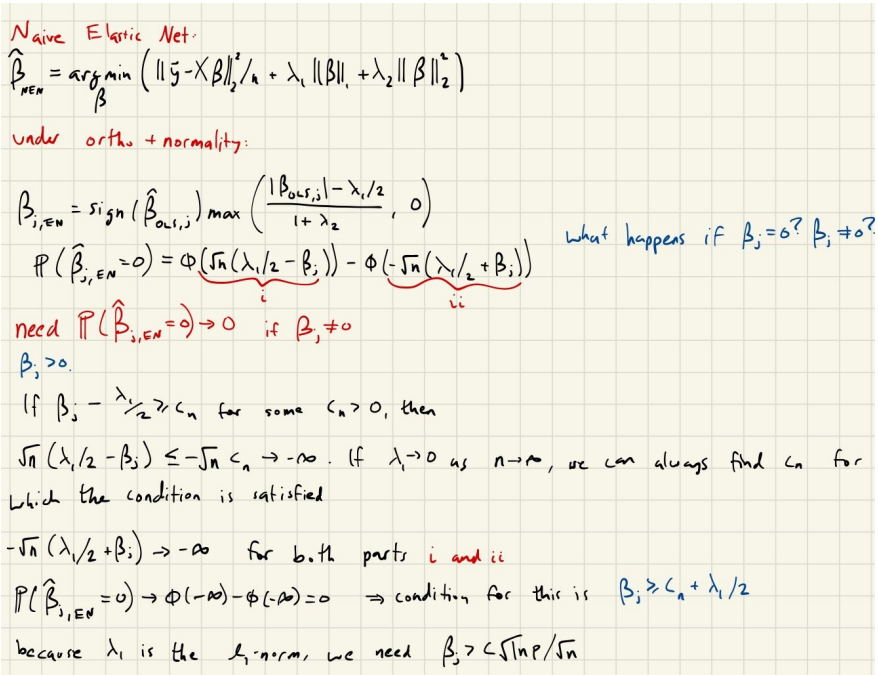
$$\lambda_2^2 = \frac{n}{\ln p}$$

$$\lambda_2 = \sqrt{\frac{n}{\ln p}}, \quad \lambda_1 = 1$$

So we only need our λ_2 to grow at a rate of $\sqrt{\frac{n}{\ln p}}$ and our λ_1 can grow at a constant rate.

- ℓ_2-norm going to zero doesn't mean our l1 norm also goes to zero
- p increases to infinity, we need $\sqrt{p} * \ell_2$-norm to also to go zero for our ℓ_1-norm to go to zero
- Lasso is consistent as long as penalty goes to zero as n to infinity

Part f



λ_1 would then need to go to zero, weak ℓ_1 penalty

Part g

$\beta = 0$. Need $P(\hat{\beta}_{j,EN} = 0) \to 1$ as $n \to \infty$

$$P(\hat{\beta}_{j,EN} = 0) = \Phi(\sqrt{n}(\lambda_1/2 - \beta_j)) - \Phi(-\sqrt{n}(\lambda_1/2 + \beta_j))$$

$$= \Phi(\sqrt{n}\,\lambda_1/2) - \Phi(-\sqrt{n}\,\lambda_1/2)$$

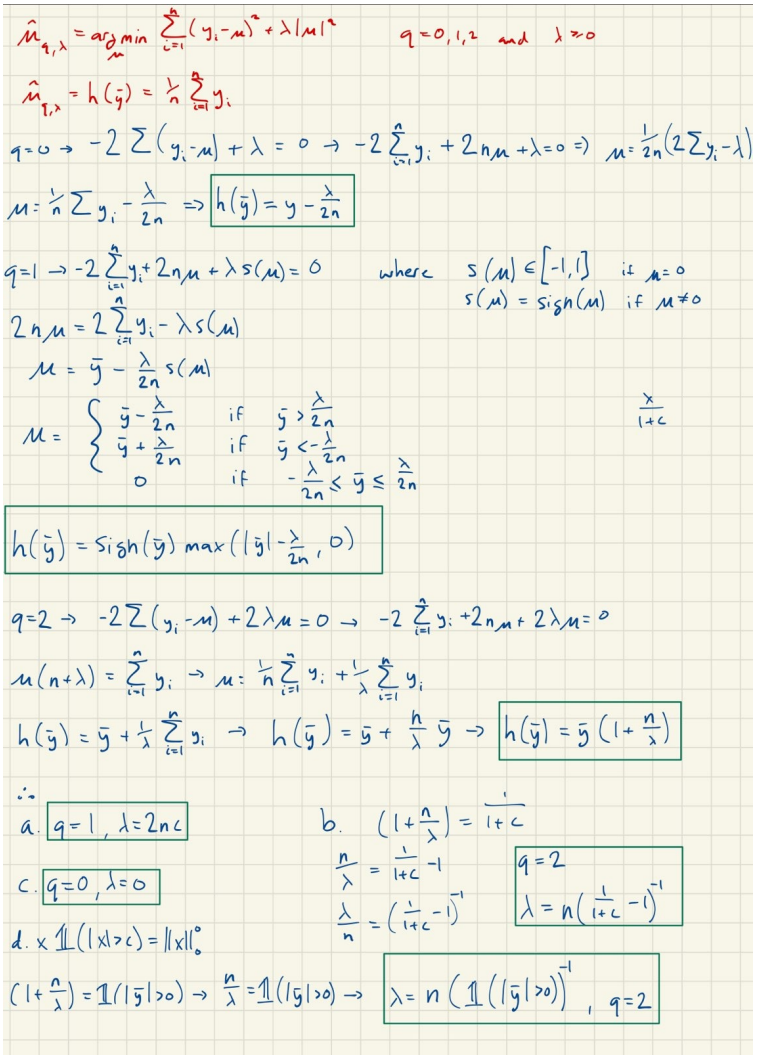
if $\sqrt{n}\lambda_1 \to \infty$, we have

$$P(\hat{\beta}_{j,EN} = 0) \to \Phi(\infty) - \Phi(-\infty) = 1 - 0 = 1$$

strong λ_1 penalty

Question 3

Split cases where $q = 0$ because 0^0 is UNDEFINED. Need to check which one is smallest. This gives us a **hard threshold** condition. We do not shrink anymore. We don't incur a bias, directly end up with the sample mean. Disadvantage is instability, and is computationally complex. Check for all different variables, which gives lowest RSS.



Question 4

`lambda.1se` can be used in order to impose stricter penalization. We are interested in characteristics of MLB players and salaries. Salary will be our response variable. Load the data and perform some preprocessing

```
set.seed(20230209)
library(ISLR2)
library(glmnet)
x <- model.matrix(Salary ~ ., data = na.omit(Hitters))[,-1]
y <- na.omit(Hitters$Salary)
```

`model.matrix()` changes letter values to numerical dummies, and removes the salary column from the matrix. Use glmnet to perform lasso regression and select tuning parameter using 10-fold cross validation. Lambda can be a value from 10^{-2} to 10^{10} Error metric is the MSE.

```
cv.lasso <- cv.glmnet(x,y,alpha=1)
best.lasso.lambda <- cv.lasso$lambda.min
lasso.model <- glmnet(x,y,alpha=1, lambda = best.lasso.lambda)
coef(lasso.model)
```

```
## 20 x 1 sparse Matrix of class "dgCMatrix"
```

```
##                       s0
## (Intercept)  129.3771245
## AtBat         -1.6336069
## Hits           5.8538014
## HmRun          .
## Runs           .
## RBI            .
## Walks          4.8802497
## Years         -9.7441000
## CAtBat         .
## CHits          .
## CHmRun         0.5836731
## CRuns          0.6967581
## CRBI           0.3696566
## CWalks        -0.5672148
## LeagueN       32.7904223
## DivisionW   -119.0549255
## PutOuts        0.2750933
## Assists        0.1878597
## Errors        -2.1368191
## NewLeagueN     .
```

Here, we see `HmRun`, `Runs`, `RBI`, `CAtBat`, `CHits`, `NewLeagueN` are all shrunk to zero. The most influential seems to be new league, which is 1 if it is "N". What we notice here is that redundant information is kicked out, i.e. `NewLeagueN` and `LeagueN` measure the same, but `LeagueN` is kept in. This may change with Elastic Net. Additionally, `Hits` is more influential as well as `Walks` in determining salary.

```
cv.ridge <- cv.glmnet(x,y,alpha=0)
best.ridge.lambda <- cv.ridge$lambda.min
ridge.model <- glmnet(x,y,alpha=0,lambda = best.ridge.lambda)
coef(ridge.model)
```

```
## 20 x 1 sparse Matrix of class "dgCMatrix"
##                       s0
## (Intercept)  10.30727421
## AtBat         0.04647470
## Hits          0.96408426
## HmRun         0.27275855
## Runs          1.10089484
## RBI           0.87588934
## Walks         1.75303796
## Years         0.51023294
## CAtBat        0.01124563
## CHits         0.06270228
## CHmRun        0.43863298
## CRuns         0.12463737
## CRBI          0.13256493
## CWalks        0.03683191
## LeagueN      25.76072952
## DivisionW   -88.35997290
## PutOuts       0.18482993
## Assists       0.03843922
## Errors       -1.68521140
## NewLeagueN    7.91754772
```

Here, the ridge regression keeps all coefficients instead of kicking some out.

```
cv.EN <- cv.glmnet(x,y,alpha=0.5)
best.EN.lambda <- cv.EN$lambda.min
EN.model <- glmnet(x,y,alpha=0.5,lambda = best.EN.lambda)
coef(EN.model)
```

```
## 20 x 1 sparse Matrix of class "dgCMatrix"
##                         s0
## (Intercept)  118.11441375
## AtBat         -1.41192070
## Hits           5.29333901
## HmRun          .
## Runs           .
## RBI            .
## Walks          4.56951709
## Years         -9.60464346
## CAtBat         .
## CHits          0.01269951
## CHmRun         0.56150425
## CRuns          0.61931715
## CRBI           0.37188270
## CWalks        -0.49700267
## LeagueN       33.31080936
## DivisionW   -120.67635468
## PutOuts        0.27233102
## Assists        0.17737562
## Errors        -2.37806842
## NewLeagueN     .
```

Elastic Net kicks out more variables than the ridge regression, but not more than lasso. This could be because there is quite a bit of correlation between each of the parameters. Here, we see that `Walks` influences `Salary` quite a bit and `Hits` is also a big indicator for the salary. Next we do adaptive lasso with Ridge Regression to create the weights vector.

```
w3 <- 1/abs(matrix(coef(cv.ridge, s=cv.ridge$lambda.min)[, 1][2:(ncol(x)+1)]))^1 ## Using gamma = 1. Se
cv.AL <- cv.glmnet(x,y,alpha = 1, penalty.factor=w3)
best.AL.lambda <- cv.AL$lambda.min
AL.model <- glmnet(x,y,alpha=1, lambda = best.AL.lambda, penalty.factor = w3)
coef(AL.model)
```

```
## 20 x 1 sparse Matrix of class "dgCMatrix"
##                         s0
## (Intercept)  -17.88883007
## AtBat          .
## Hits           2.94543310
## HmRun         -0.60350039
## Runs          -1.02007325
## RBI           -0.05324016
## Walks          2.41165429
## Years          .
## CAtBat         .
## CHits          .
## CHmRun         1.11802114
## CRuns          0.30855434
## CRBI           0.04805990
## CWalks         .
```

```
## LeagueN       76.01436664
## DivisionW   -131.34517769
## PutOuts        0.22849880
## Assists        .
## Errors        -2.82085043
## NewLeagueN   -28.44695844
```

Although hard to see, we have multiple values that are "kicked out", or have marginal effect, and others, such as `Years` and `DivisionW` and `LeagueN` which stay influential. This may be the best model yet.

Now we use `HDeconometrics` to perform a lasso regression and select a tuning parameter using AIC and BIC

```
library(HDeconometrics)
bic.lasso <- ic.glmnet(x,y,crit = "bic", alpha = 1)
coef(bic.lasso)
```

```
## (Intercept)       AtBat        Hits       HmRun        Runs         RBI
##  24.8990715   0.0000000   1.8487366   0.0000000   0.0000000   0.0000000
##       Walks       Years      CAtBat       CHits      CHmRun       CRuns
##   2.1945815   0.0000000   0.0000000   0.0000000   0.0000000   0.2059445
##        CRBI      CWalks     LeagueN   DivisionW     PutOuts     Assists
##   0.4092047   0.0000000   0.0000000 -99.7364356   0.2155667   0.0000000
##      Errors  NewLeagueN
##   0.0000000   0.0000000
```

```
aic.lasso <- ic.glmnet(x,y,crit = "aic", alpha = 1)
coef(aic.lasso)
```

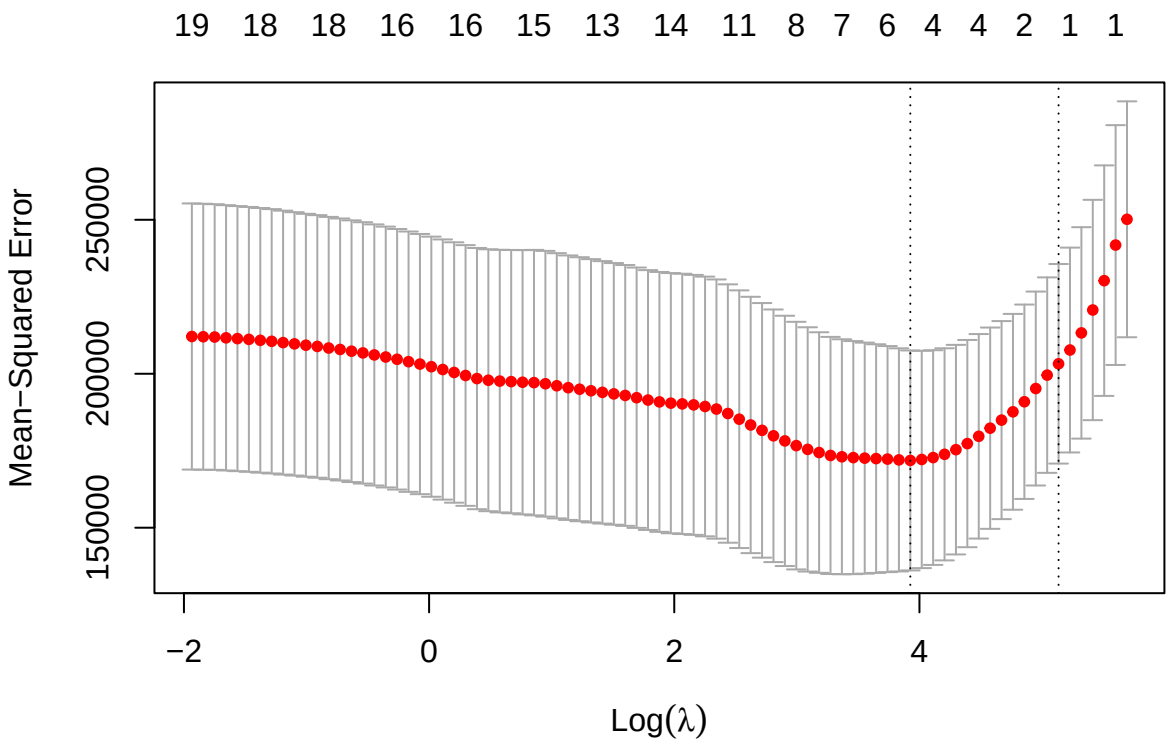
```
##  (Intercept)        AtBat         Hits        HmRun         Runs          RBI
##  129.4155571   -1.6130155    5.8058915    0.0000000    0.0000000    0.0000000
##        Walks        Years       CAtBat        CHits       CHmRun        CRuns
##    4.8469340   -9.9724045    0.0000000    0.0000000    0.5374550    0.6811938
##         CRBI       CWalks      LeagueN    DivisionW      PutOuts      Assists
##    0.3903563   -0.5560144   32.4646094 -119.3480842    0.2741895    0.1855978
##       Errors   NewLeagueN
##   -2.1650837    0.0000000
```

Here, we see that BIC is more sparse. The ones chosen by BIC are also chosen by cross validation. AIC does not shrink as many variables to zero. In fact, AIC chooses the same values as cross validation. The only difference is in the coefficients slightly.

Now we see how well we can predict. The difference here is we ignore portions of the data set instead of above where we included everything

```
train <- sample(1:nrow(x), nrow(x)/2)
test <- (-train)
y.test <- y[test]

cv.out <- cv.glmnet(x[train,],y[train],alpha=1)
plot(cv.out)
```




```
bestlam <- cv.out$lambda.min
lasso.mod <- glmnet(x[train,],y[train],alpha = 1, lambda = bestlam)
lasso.pred <- predict(lasso.mod, s = bestlam, newx = x[test,])
mean((lasso.pred - y.test)^2)
```

```
## [1] 85911.68
```

```
ridge.cv.out <- cv.glmnet(x[train,],y[train],alpha=0)  # Ridge
ridge.bestlam <- ridge.cv.out$lambda.min
ridge.mod <- glmnet(x[train,],y[train],alpha = 0, lambda = ridge.bestlam)
ridge.pred <- predict(ridge.mod, s = ridge.bestlam, newx = x[test,])
mean((ridge.pred - y.test)^2)
```

```
## [1] 80993.69
```

```
EN.cv.out <- cv.glmnet(x[train,],y[train],alpha=0.5)  # EN
EN.bestlam <- EN.cv.out$lambda.min
EN.mod <- glmnet(x[train,],y[train],alpha = 0.5, lambda = EN.bestlam)
EN.pred <- predict(EN.mod, s = EN.bestlam, newx = x[test,])
mean((EN.pred - y.test)^2)
```

```
## [1] 81108.17
```

```
w3 <- 1/abs(matrix(coef(cv.ridge, s=cv.ridge$lambda.min)[, 1][2:(ncol(x)+1)]))^1  # Use same weight as
AL.cv.out <- cv.glmnet(x[train,],y[train],alpha=1,penalty.factor=w3)  # AL
AL.bestlam <- AL.cv.out$lambda.min
AL.mod <- glmnet(x[train,],y[train],alpha = 1, penalty.factor = w3)
AL.pred <- predict(AL.mod, s = AL.bestlam, newx = x[test,])
mean((AL.pred - y.test)^2)
```

```
## [1] 92350.35
```

Our best model for prediction in this case is the Ridge Regression, with a mean squared prediction error of

80993.28.

Question 5

Load and clean the data

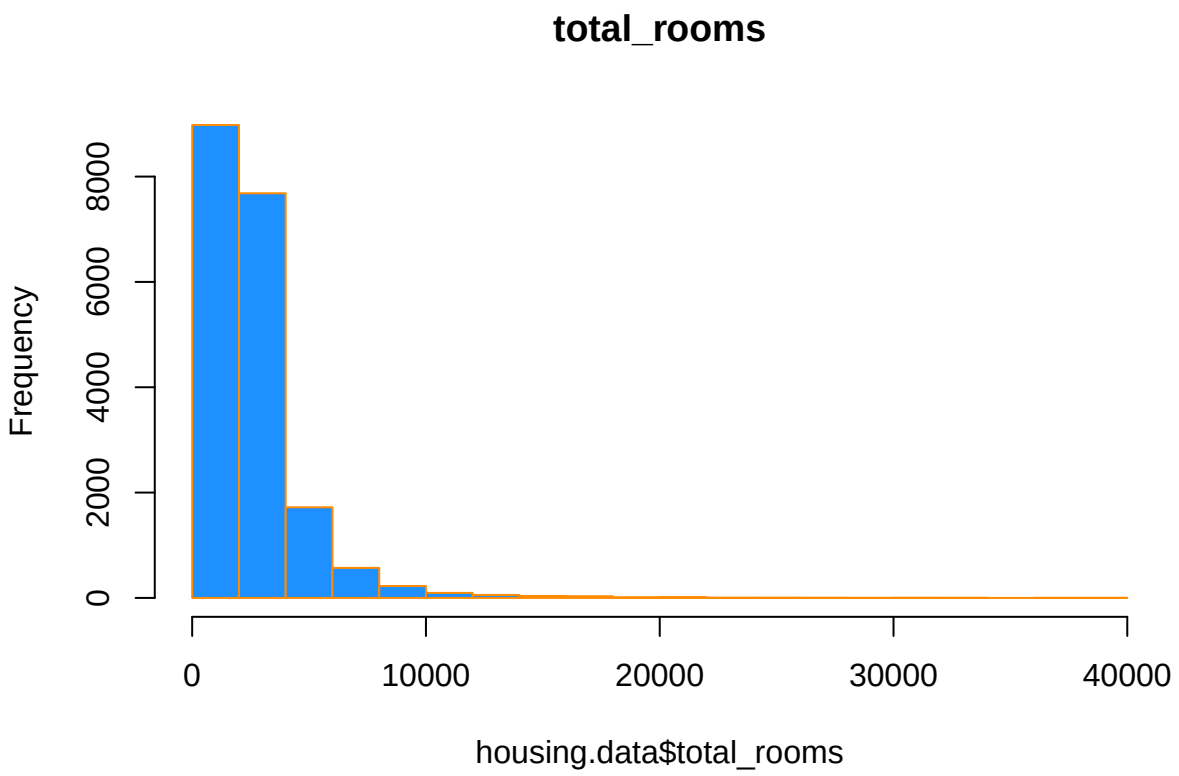
```
housing.data <- read.csv("/Users/Ivan/Desktop/Areas/UM/CLASSES/YEAR_2/period_4/BIg_Data/computing_the_la
housing.data$ocean_proximity = as.factor(housing.data$ocean_proximity)  # Change from text-based to fac
housing.data <- housing.data[housing.data$ocean_proximity != "ISLAND", ]  # Remove "ISLAND" as it could
housing.data <- na.omit(housing.data)  # Delete observations with missing values
housing.data <- housing.data[housing.data$median_house_value < 500000, ]  # Remove value larger than 50
```

Variables which are best treated as categorical:

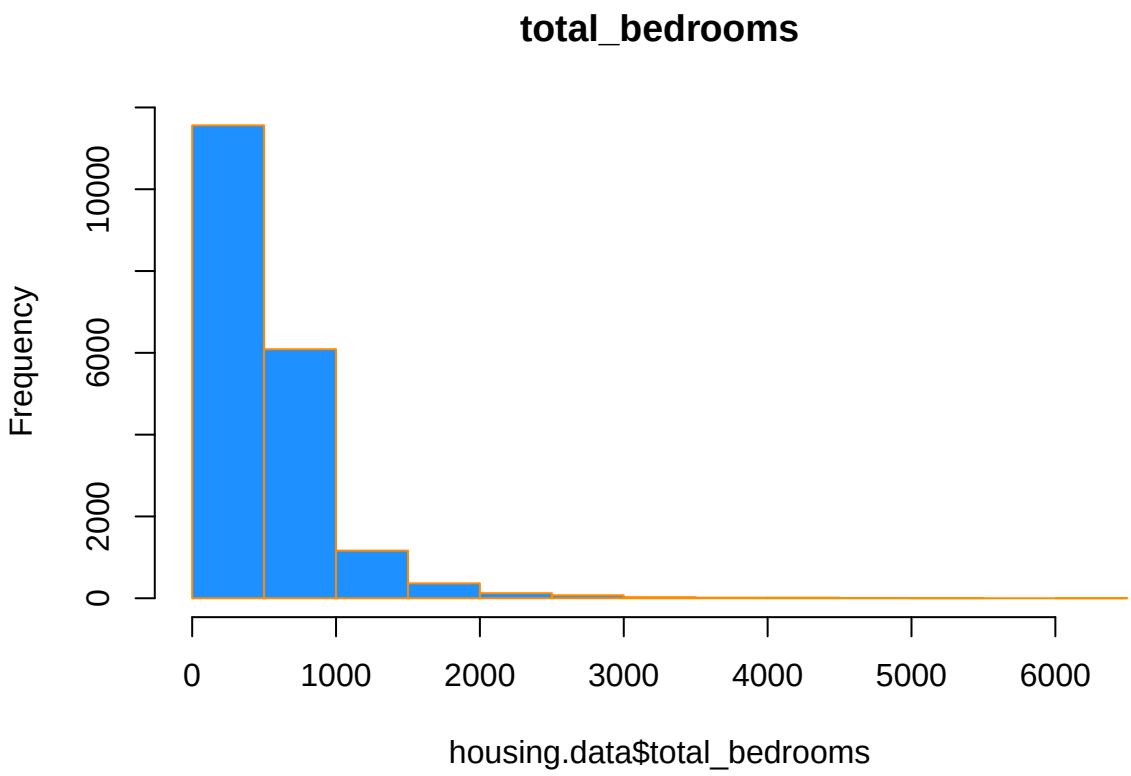
- Number of rooms
- Population
- Number of bedrooms

These are chosen because they are the most right skewed given a histogram of counts.

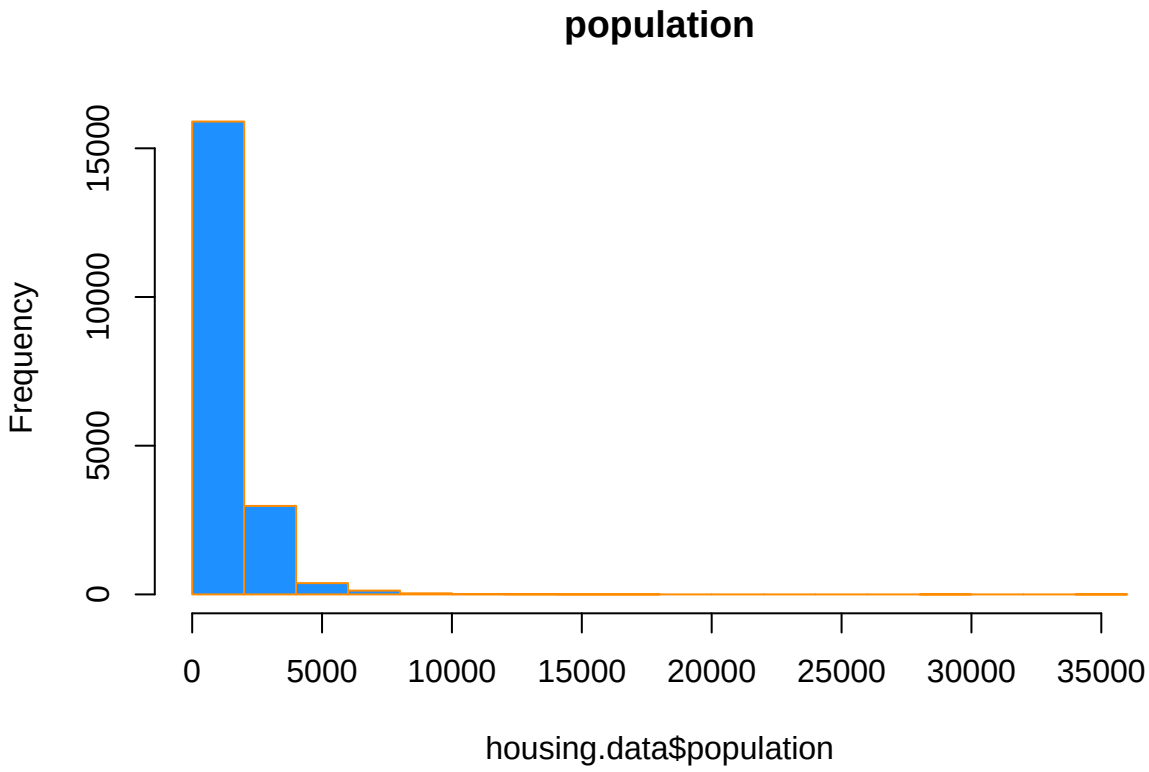
```
hist(housing.data$total_rooms, breaks = 20, main = "total_rooms", border="darkorange", col="dodgerblue"
```




```
hist(housing.data$total_bedrooms, breaks = 20, main = "total_bedrooms", border="darkorange", col="dodge
```




```
hist(housing.data$population, breaks = 20, main = "population", border="darkorange", col="dodgerblue")
```



Dummies are split into low, average, and high, with dummies created for only low and average to prevent collinearity. Thresholds are chosen from the first third tercile

```
tercile.rooms <- quantile(housing.data$total_rooms, c(0:3/3))
housing.data$lowrooms <- ifelse(housing.data$total_rooms < tercile.rooms[2],1,0)
housing.data$midrooms <- ifelse(housing.data$total_rooms >= tercile.rooms[2] & housing.data$total_rooms

tercile.bedrooms <- quantile(housing.data$total_bedrooms, c(0:3/3))
housing.data$lowbedrooms <- ifelse(housing.data$total_bedrooms < tercile.bedrooms[2],1,0)
housing.data$midbedrooms <- ifelse(housing.data$total_bedrooms >= tercile.bedrooms[2] & housing.data$to

tercile.pop <- quantile(housing.data$population, c(0:3/3))
housing.data$lowpopulation <- ifelse(housing.data$population < tercile.pop[2],1,0)
housing.data$midpopulation <- ifelse(housing.data$population >= tercile.pop[2] & housing.data$populatio
```

We also add interactions and higher powers. They are shown in the following formula

```
ff <- median_house_value ~ . + .^2 + I(housing_median_age^2)
housing.X <- model.matrix(ff,housing.data)[,-1]
housing.y <- housing.data$median_house_value
dim(housing.X)
```

```
## [1] 19443   166
```

We now have 166 variables and 19443 observations. We do our normal analysis as follows

```
train.h <- sample(1:nrow(housing.X), nrow(housing.X)/2)
test.h <- (-train.h)
y.test.h <- housing.y[test.h]

cv.housing.lasso <- cv.glmnet(housing.X[train.h,],housing.y[train.h],alpha=1)
housing.lambda.lasso <- cv.housing.lasso$lambda.min
housing.lasso <- glmnet(housing.X[train.h,],housing.y[train.h],alpha=1, lambda = housing.lambda.lasso)
housing.lasso.pred <- predict(housing.lasso, s = housing.lambda.lasso, newx = housing.X[test.h,])
sqrt(mean((housing.lasso.pred - y.test.h)^2))
```

```
## [1] 55905.33
```

```
cv.housing.ridge <- cv.glmnet(housing.X[train.h,],housing.y[train.h],alpha=0)
housing.lambda.ridge <- cv.housing.ridge$lambda.min
housing.ridge <- glmnet(housing.X[train.h,],housing.y[train.h],alpha=0, lambda = housing.lambda.ridge)
housing.ridge.pred <- predict(housing.ridge, s = housing.lambda.ridge, newx = housing.X[test.h,])
sqrt(mean((housing.ridge.pred - y.test.h)^2))
```

```
## [1] 57585.01
```

```
cv.housing.EN <- cv.glmnet(housing.X[train.h,],housing.y[train.h],alpha=0.5)
housing.lambda.EN <- cv.housing.EN$lambda.min
housing.EN <- glmnet(housing.X[train.h,],housing.y[train.h],alpha=0.5, lambda = housing.lambda.EN)
housing.EN.pred <- predict(housing.EN, s = housing.lambda.EN, newx = housing.X[test.h,])
sqrt(mean((housing.EN.pred - y.test.h)^2))
```

```
## [1] 55889.82
```

```
w3.h <- 1/abs(matrix(coef(cv.housing.ridge, s=cv.housing.ridge$lambda.min)[, 1][2:(ncol(housing.X)+1)])
cv.housing.AL <- cv.glmnet(housing.X[train.h,],housing.y[train.h],alpha=1,penalty.factor=w3.h)  # AL
housing.lambda.AL <- cv.housing.AL$lambda.min
housing.AL <- glmnet(housing.X[train.h,],housing.y[train.h],alpha = 1, penalty.factor = w3.h)
housing.AL.pred <- predict(housing.AL, s = housing.lambda.AL, newx = housing.X[test.h,])
sqrt(mean((housing.AL.pred - y.test.h)^2))
```

```
## [1] 59696.33
```

```
# Group Lasso
library(gglasso)

# Split groups based on names: total 16 groups
GL.groups <- numeric(ncol(housing.X))
col.names <- colnames(housing.X)
group_names <- c("long", "lati", "housing", "total_r", "total_b",
                 "pop", "househ", "median", "ocean", "lowrooms",
                 "midrooms", "lowb", "midb", "lowp", "midp")
group_values <- 1:length(group_names)

for (i in 1:length(group_names)) {
  GL.groups[grep(group_names[i], col.names)] <- group_values[i]
}

# Take only 500 samples of housing because gglasso is slow
sample.housing <- housing.data[sample(nrow(housing.data), size = 500),]

# Split the data matrices once again and organize the grouping columns
sample.housing.X <- model.matrix(ff,sample.housing)[,-1]
sample.housing.X <- sample.housing.X[,GL.groups]
sample.housing.y <- sample.housing$median_house_value

# Split into train and test
sample.train <- sample(1:nrow(sample.housing.X), nrow(sample.housing.X)/2)
sample.test <- (-sample.train)
sample.y.test <- sample.housing.y[sample.test]

# Set vector of groups
GL.groups <- sort(GL.groups)-1
groups.c <- GL.groups
groups.c[groups.c == 1] <- 2
groups.c[groups.c == 0] <- 1

# Perform group lasso
cv.housing.GL <- cv.gglasso(sample.housing.X[sample.train,],sample.housing.y[sample.train], group = grou
housing.lambda.GL <- cv.housing.GL$lambda.min
housing.GL <- gglasso(sample.housing.X[sample.train,],sample.housing.y[sample.train], lambda = housing.l
housing.GL.pred <- predict(housing.GL, s = housing.lambda.GL, newx = sample.housing.X[sample.test,])
sqrt(mean((housing.GL.pred - sample.y.test)^2))
```

```
## [1] 76126.78
```

So which variables matter most for predicting housing prices?

- You fix other variables and change one
- Gives a marginal effect estimate
- Create counterfactual data, can be done with a function
- Take means, and then just vary the age
- Regress house price on age, people living in the area