Lab 4: Linear Model Selection and Regularization

First Name:Bill **Last Name**:Zhou **NetID**:wz325

Lab 4 is due March 8th (Tuesday) by 11:59pm on Gradescope. Submit your answers as one .pdf file, including all R code that you had to write to obtain your answers. Regrade requests should be submitted via Gradescope. Questions should be sent to Tao Jiang (`tj293@cornell.edu`).

If the question requires running R code, please attach your code snippet (screenshot/text) to each corresponding question

In this lab, we will cover linear model selection and regularization.

We will use the *Hitters* data set, which has 322 observations of major league players on 20 variables from the 1986 and 1987 seasons. We wish to predict a baseball player's Salary (available via the `Salary` column) on the basis of various statistics associated with performance. The dataset is available from the `ISLR` library.

```
> library(ISLR)
> fix(Hitters)
```

The first step is to preprocess the data and remove any observations with missing entries, which should leave you with a total of 263 observations.
To make sure we are not getting a horrible fit, use the studentized residuals (recall Lab 1) to **remove** any points characterized as outliers. Report the adjusted-R^2 score of a linear model fit using all predictors *before* and *after* you remove the outliers. You should observe a nontrivial improvement. **(R code required)**
In the sequel, **we will refer to the resulting data frame as** `df.post`.

```
 adjusted-R^2 before outliers removal:0.5106
 adjusted-R^2 after outliers removal:0.6109
#Code part
lm1.fit=lm(formula= Salary~.,data=Hitters)
summary(lm1.fit)
Res=rstudent(lm1.fit)
Res_new=abs(Res)<3
index_new=which(Res_new %in% TRUE)
df.post=Hitters[index_new]
lm2.fit=lm(formula =Salary~.,data=df.post )
summary(lm2.fit)
```

Examine the summary of your model fit **after** removing the outliers – you should (hopefully) see a very large p-value for the intercept. Fit a linear model **without** intercept, and report its adjusted R^2 score. Did the situation improve? **(R code required)**

```
p-value for intercept:0.6038
Adjuted R square=0.6019
After set intercept to 0
New Adjusted R square= 0.8402
#Code
lm3.fit=lm(formula =Salary~AtBat+Hits+HmRun+RBI+Walks+Years+CAtBat+CHits+
CHmRun+CRuns+CRBI+CWalks+League+Division+PutOuts+Assists+Errors+NewLeague
+0.,data=df.post  )
summary(lm3.fit)
```

1 Subset Selection Methods

We focus on three subset selection methods here: Best Subset Selection and Forward and Backward Stepwise Selection. These selections can be performed using `regsubsets()` with argument `method = "exhaustive"` (default) or `"forward"` or `"backward"` for the methods, respectively. The function `regsubsets()` are from `library(leaps)`
For example, the following code performs best-subset selection, with maximum number of predictors equal to `nvmax` = 19. Note that we are explicitly instructing R to not use an intercept

```
> library(leaps)
> regfit.full <- regsubsets(Salary~., data=df.post, nvmax=19, intercept=F)
> reg.summary <- summary(regfit.full)
```

Which predictors are selected for the best 5-variable model?

```
#1:CRBI
#2:Runs+CRBI
#3:Runs+CRBI+PutOuts
#4:Hits+CAtBat+CHits+CHmRun
#5:Runs+CAtBat+CHits+CHmRun+PutOuts
```

We now have 19 candidate models in our hands. In order to pick the “best” of them, we can compute their RSS, R^2 score, adjusted R^2 score, C_p score and BIC. **Without looking** at the model summary, can you explain which of these models will have:

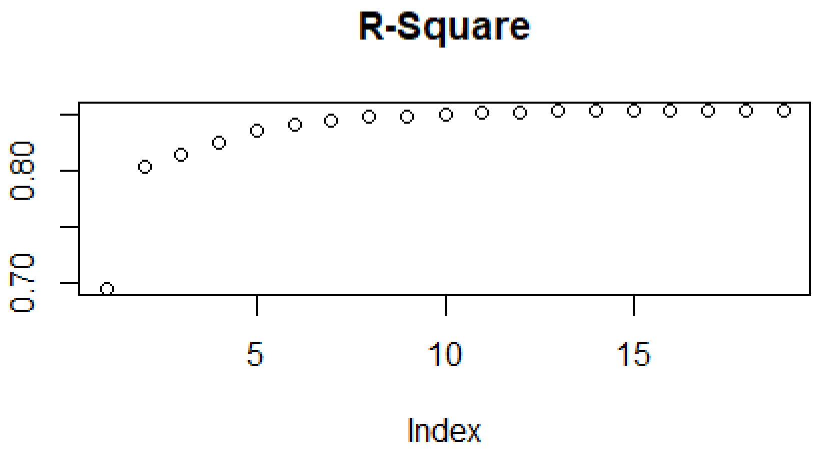
- the smallest RSS?
- the largest R^2 score?

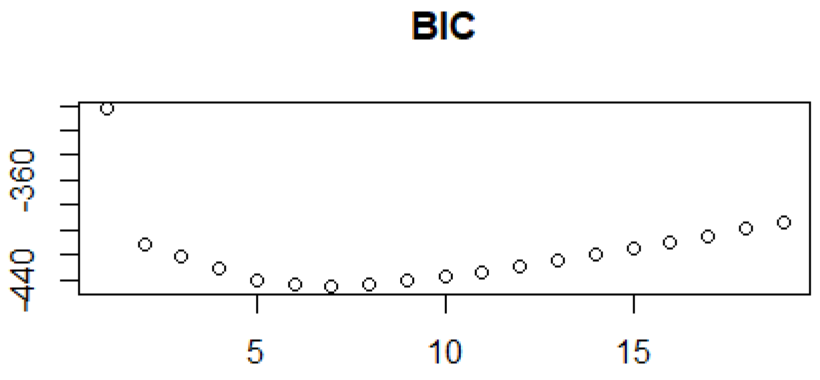
Is it a good idea to select either of these models without looking at the other statistics? Why or why not?

```
Smallest RSS: #5
Largest R^2:#5
```

```
More complex model tend to have less RSS and larger R-square
Generally I think it’s okay to make selection by the statistics we mentioned
above Since they are the key metrics to evaluate how well the models are
```

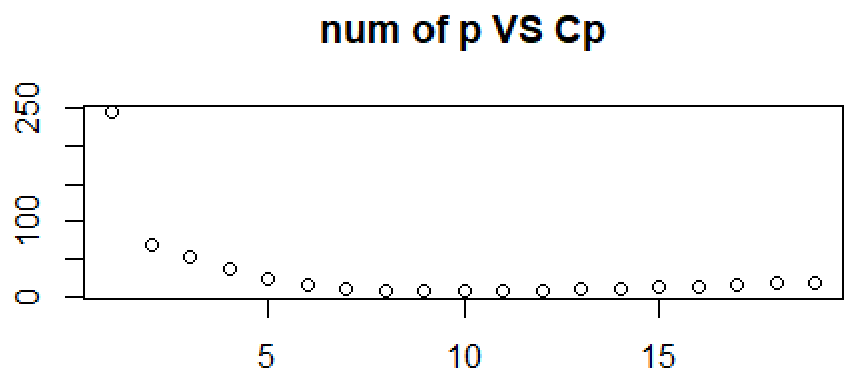
The summary object `reg.summary` contains all the aforementioned information. Plot the adjusted R^2 score and the BIC score of your models, and explain which model you would choose for each metric.





```
From R-square plot, I will select #19 since it has the largest R-Square
From BIC plot, I will select #& since it has the smallest BIC
```

In the following, plot the C_p against number of predictors in the model. What is the optimal number of predictors based on C_p?



```
According to the plot above
Optimal number of perdictors based on Cp is 10
```

In this example, the number of predictors ($p = 19$) is small enough so that using best-subset selection isn't super time consuming. Behind the scenes, the package may also use pruning steps, to avoid fitting a few models. In the sequel, we will also examine forward and backward stepwise selection. These can be implemented using `regsubsets()` as well (see `method` argument).

We can also select models using the validation approach (such as K-fold cross validation). Problem 1 of the take-home questions asks you to implement 10-fold cross validation for forward stepwise selection method.

2 Ridge and Lasso

Ridge and lasso are two commonly-used regularization techniques in regression. When solving an optimization problem to minimize the squared loss, the two methods apply different penalties on regression coefficients.

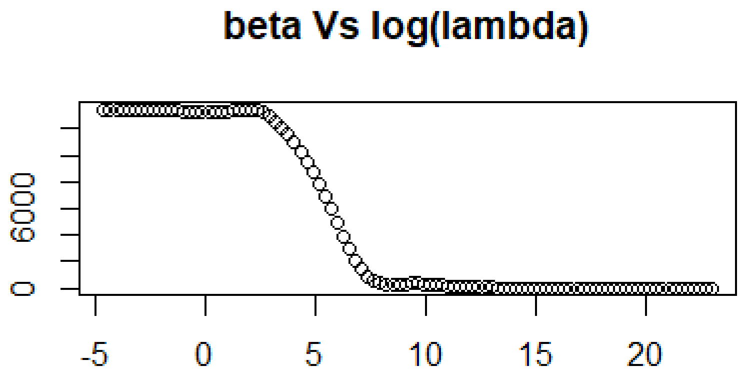
In fitting both methods, we use the well-written R library `glmnet`. The main function `glmnet()` has an argument `alpha` that controls whether the fit being ridge or lasso: `alpha = 0` corresponds to ridge and `alpha = 1` corresponds to lasso. The syntax of `glmnet()` is slightly different from what we have encountered so far. In particular, we must pass in an x matrix with predictors in columns and a y vector separately:

```
> x <- model.matrix(Salary~., df.post)[,-1]
> y <- df.post$Salary
```

Both ridge and lasso has a tuning parameter to tune. The user can either specify a sequence of tuning parameter himself/herself or let `glmnet()` to generate a sequence automatically. Let us first look at how to specify the tuning parameter sequence ourselves. Below we construct 100 equally-spaced penalty values on log-scale and fit a ridge regression with them.

```
> library(glmnet)
> grid <- 10^seq(10, -2, length=100) # lambda sequence
> ridge.mod <- glmnet(x, y, alpha=0, lambda=grid, intercept=FALSE)
> dim(coef(ridge.mod))
```

`ridge.mod$beta` is a matrix with the jth column being ridge coefficients ($\hat{\beta}^{\text{ridge}}_{\lambda_j}$) corresponding to the jth tuning parameter (λ_j). Plot the ℓ_2 norm of $\hat{\beta}^{\text{ridge}}_{\lambda}$ versus $\log(\lambda)$. What pattern do you observe and why does the pattern look like this?



```
As we can see from the plot above
When λ is tiny , |β|^2 is relative large
As λ increase, |β|^2 keeps diminishing
As λ goes to infinity,|β|^2 is around 0
```

With the ridge fit `ridge.mod`, we can predict at a new tuning parameter value. For example, to predict the response when $\lambda = 50$, given a new set of data `newx`, we can run

```
> # Predict at lambda = 50
> predict(ridge.mod, newx, s=50)
```

What is the drawback of manually specifying the tuning parameter λ?

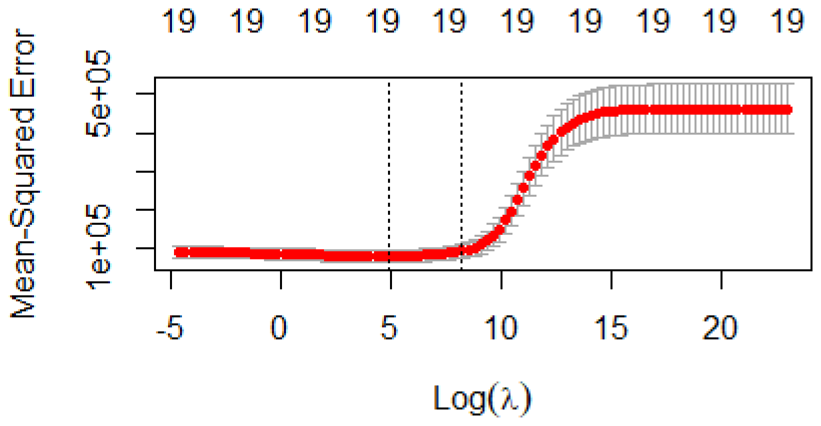
```
This might lead to fitting concentrate too much/too less on the regulazation
terms, affect the accuracy of the model.
```

Below we will let `glmnet()` use its own sequence of tuning parameters without providing one ourselves. By default, `glmnet()` constructs a sequence of 100 tuning parameters with the first one being the largest.

To select optimal tuning parameter, we use K-fold cross validation. The `glmnet` package has a convenient cross validation function called `cv.glmnet()`. It is important that model fit and cross validation are done on the same sequence of tuning parameters. So don’t forget to pass the tuning parameter in `cv.glmnet()`. In the following, we perform 10-fold cross validation after splitting training and test sets of equal sizes. For reproducible results, we set seed for random number generator before train-test split and before cross validaton.

```
> # Split the data into training data set and test data set.
> set.seed(1)
> train <- sample(1:nrow(x), nrow(x)/2)
> y.train <- y[train]
> y.test <- y[-train]
>
> # Model fit on the training set
> ridge.mod <- glmnet(x[train,], y[train], alpha=0, lambda=grid,
+ intercept=FALSE)
> # Use 10-fold cross-validation to choose lambda
> set.seed(1)
> cv.out <- cv.glmnet(x[train,], y[train], nfolds = 10, lambda=grid,
+ alpha=0, intercept=FALSE)
```

Plot the cross validation error versus the log of the tuning parameter. What is the value of the optimal tuning parameter? Using this optimal value, what is the mean squared prediction error on the *test* set?

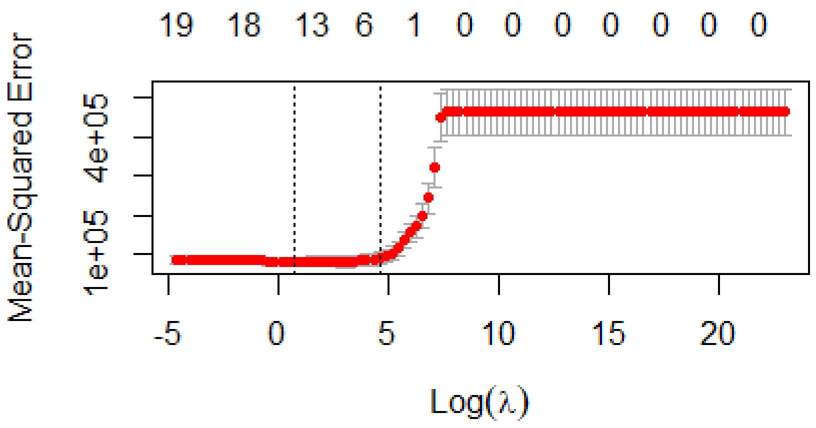


```
a bit hard to read from the plot directly, since lots of points squeezing
around the minimum,but we can call function to get
> cv.out $lambda.min
[1] 132.1941
Using λ=132.1941,we have testing MSE = 113186.6
```

Fitting and cross validating Lasso can be done similarly. In the following, we perform 10-fold cross validation for lasso on the same training set.

```
> # Model fit on the training set
> lasso.mod <- glmnet(x[train,], y[train], alpha=1, lambda=grid,
+ intercept=FALSE)
> # Use 10-fold cross-validation to choose lambda
> cv.out <- cv.glmnet(x[train,], y[train], alpha=1, lambda=grid,
+ intercept=FALSE)
```

Plot the cross validation error versus the log of the tuning parameter:



What is the value of optimal tuning parameter? What is the mean squared prediction error on the test set? How many predictors have their coefficients equal to 0 at the optimal tuning parameter?

```
> cv.out$lambda.min
[1] 2.009233
For Lasso
Using λ=2.009233,we have testing MSE = 113186.6
17  predictors have their coefficients equal to 0
```

Take-Home Problems

1. Subset selection

For this problem, assume that you are given a sample of n points with p predictors.

1. Suppose that you are running best subset selection for model selection. How many models will you have to fit in total?

```
let m=min(n,p)
total num of model=2^m
```

2. In the same setting as the previous question, let $\mathcal{M}_i$ denote the best model using i predictors. Let $\text{err}_{\text{train}}(\mathcal{M}_i)$ denote its training MSE. Is is possible to have $\text{err}_{\text{train}}(\mathcal{M}_i) < \text{err}_{\text{train}}(\mathcal{M}_{i+j})$, for some $j > 0$? Why or why not?

```
Yes, if we increase the number of predictors, then select the best
model of that number of predictors, the model with more predictors
are likely to have less training error compare with the model with
fewer predictors
```

3. Consider two models using i predictors: $\mathcal{M}_i^{\text{best-subset}}$, found using best-subset selection, and the model $\mathcal{M}_i^{\text{forward}}$, which is found using forward stepwise selection. Which of the two models has **lower** training error?

```
Mi_best-subset has lower training error than Mi_forward
(or equals to training error of Mi_forward)
Since for best subset algo, we enumerate all possibility, thus
Mi_best-subset has the smallest error among all model with i
predictors,which Mi_forward is also included
```

4. Now, suppose that you are using stepsize selection to find a good model. Which of the two variants would you prefer to use (forward vs. backward) if $p > n$, and why?

```
Forward method for sure, because all the model has more than n predictors
makes no sense, it will lead to overfitting, so starting with num of
predictor=0 is computationally more cost effective for this case
```

5. Read pages 248-250 from the ISLR book (Section 6.5.3) about using cross-validation combined with best subset selection. Because `regsubsets` does not provide a `predict` function, we will have to provide one. To avoid getting into the subtleties of the function provided in `ISLR`, we will define the following variant:

```
predict.regsubsets <- function(regfit, newdata, id, ...) {
    mat <- model.matrix(formula(Salary~. + 0), newdata)
    coefi <- coef(regfit, id=id)
    xvars <- names(coefi)
    as.matrix(mat[, xvars]) %*% coefi
}
```

To use this function for prediction using the best subset of p predictors and given input data `dfIn`, we simply call

```
        predict(regfit.full, dfIn, p)
```

Your objective here: use K-fold cross-validation with $K = 5$ and $K = 10$ to compute the number of predictors achieving the best validation error. Use the code in page 250 of ISLR as a guide, but remember to use our version of `predict`, and provide `intercept=F` as an argument to `regsubsets`. You may want to set the seed to a fixed value as well. **(R code required)**

```
k=10
folds=sample (1:k,nrow(x),replace=TRUE)
cv.errors =matrix (NA,k,19, dimnames =list(NULL , paste (1:19) ))

for(j in 1:k){
  best.fit=regsubsets (Salary~.,data=x[train,] [folds!=j,],nvmax=19)
  for(i in 1:19){
     pred=predict.regsubsets (best.fit ,x[train,] [folds ==j,],id=i)
     cv.errors[j,i]= mean( ( Hitters$Salary[ folds==j]-pred)^2)
     }
  }
```

6. Repeat the experiment using forward and backward stepwise selection. Do your results differ significantly? List the predictors selected using forward and backward stepwise selection, and attach your .R code.

```
#R Code
forward.full <- regsubsets(Salary~., data=df.post,nvmax=19, intercept=F,method
= "forward")
forward.summary <- summary(backward.full)
backward.full <- regsubsets(Salary~., data=df.post,nvmax=19, intercept=F,method
= "backward")
backward.summary <- summary(backward.full)

forward.errors =matrix (NA,k,19, dimnames =list(NULL , paste (1:19) ))
backward.errors =matrix (NA,k,19, dimnames =list(NULL , paste (1:19) ))
for(j in 1:k){
  for(i in 1:19){
    forward.pred=predict.regsubsets (forward.full ,df.post [folds ==j,],id=i)
    backward.pred=predict.regsubsets (backward.full ,df.post [folds ==j,],id=i)
    forward.errors[j,i]= mean( ( Hitters$Salary[ folds==j]-forward.pred)^2)
    backward.errors[j,i]= mean( ( Hitters$Salary[ folds==j]-backward.pred)^2)
  }
}
forward.mse=rowSum(forward.errors)/k
backward.mse=rowSum(backward.errors)/k


 Best model for forward/backward

 Backward:

> coef(backward.full,13)
      AtBat        Hits         RBI       Walks
 -1.2550831   5.6754125  -1.7366685   5.2713458
      Years      CAtBat       CHits      CHmRun
 11.8170546  -0.3136236   0.9575454   2.6201527
      CRuns      CWalks   DivisionW     PutOuts
  0.6994203  -0.5464384 -72.6149587   0.2676866
    Assists
  0.1253129

 Forward:
> coef(forward.full,14)
      AtBat        Hits        Runs         RBI
 -1.1909963   6.5398487  -2.0242976  -1.7653880
      Walks       Years      CAtBat       CHits
  5.7061526   9.8101382  -0.2702140   0.5965148
     CHmRun       CRuns        CRBI      CWalks
  1.8446339   1.0532108   0.3032507  -0.6234319
  DivisionW     PutOuts
-74.2805963   0.2575952
```

```
As we can see, most terms are the same, but still some difference between them
```

2. Regularization

In the following, recall the definitions of the ℓ_p norms for $p \in \{1, 2\}$ for a vector in $\mathbb{R}^n$:

$$\|u\|_1 := \sum_{i=1}^{n} |u_i|, \quad \|u\|_2 := \sqrt{\sum_{i=1}^{n} u_i^2} \tag{1}$$

1. Write down the optimization problems for the Lasso and ridge regression in *matrix/vector* notation using the following notations: $y \in \mathbb{R}^n$ is the vector of responses, $X \in \mathbb{R}^{n\times p}$ is the predictor matrix, $\beta \in \mathbb{R}^p$ is the vector of regression coefficients, $\beta_0 \in \mathbb{R}$ is the intercept, and $\mathbf{1}_n$ is the all-ones vector in n dimensions.

$$min(\sum_{i=1}^{n}(y-\beta X)+\lambda*\sum_{i=1}^{n}|u_i|), (Ridge) \quad min(\sum_{i=1}^{n}(y-\beta X)+\lambda*\sqrt{\sum_{i=1}^{n}u_i^2})(Lasso) \tag{2}$$

2. Suppose we estimate the coefficients of a linear regression model using the **constrained** version of the lasso, i.e.

$$\hat{\beta}_\lambda := \arg\min_\beta \left\{ \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 \quad \text{subject to} \quad \|\beta\|_1 \le \lambda \right\}.$$

Assume we start with $\lambda = 0$ and gradually increase it, to obtain a set of solutions $\hat{\beta}_\lambda$. Which of the following is true about the **training** error (RSS) of the solutions, and why?

(a) as we increase λ, the training error starts decreasing until a critical value $\lambda^\star$, after which it increases again

(b) as we increase λ, the training error can never increase

```
(a) is true, of course the training error starts decreasing at first,
both in (a)and (b), however, as λ goes to infinity,  training error
will only decrease,since the constraint of betas are lesser and lesser,
this is a good news for training error,but might not be a good new for
the model, might lead to overfitting when λ is infinity.
```

3. Suppose that we now use the familiar form of ridge regression, i.e. for a fixed λ we estimate

$$\hat{\beta}_\lambda := \arg\min_\beta \left\{ \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 + \lambda\|\beta\|^2 \right\} \tag{3}$$

What happens to the **bias** of the resulting linear model as we grow λ, starting from 0, and why?

```
When λ=0, this is a regression without regularization
When λ is infinity ,we expect to see a zero model( all parameter=0)
Thus the bias will go down with λ until hitting the critical point
then keep bursting up
```

What do you predict will happen to the **test** error as you grow λ, if your model for $\lambda = 0$ has very small training error but large test error? Explain.

```
Testing error will goes down then goes up, same as model bias
For the case λ=0, without penalty terms, we are likely to encounter overfitting
Causing smaller error with huge testing error
```

4. Suppose that you run linear regression using `lm()` and, when you examine the summary of the model, observe that almost all variables exhibit small p-values (assume $p \ll n$). On the other hand, your test error is significantly higher compared to the training error. Which of the two regularization methods (ridge regression vs. Lasso) would you prefer to use, and why?

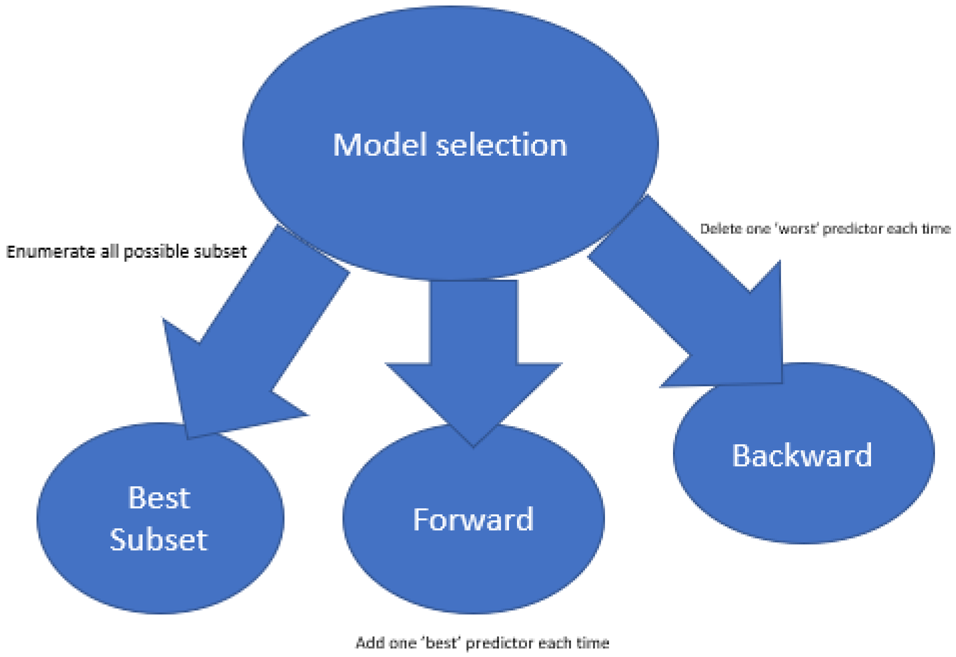
```
Lasso
Since all the variables have small p-values, thus we would like to treat them
equality when apply for the penalty term, in that sense, Lasso seems a better
option
```

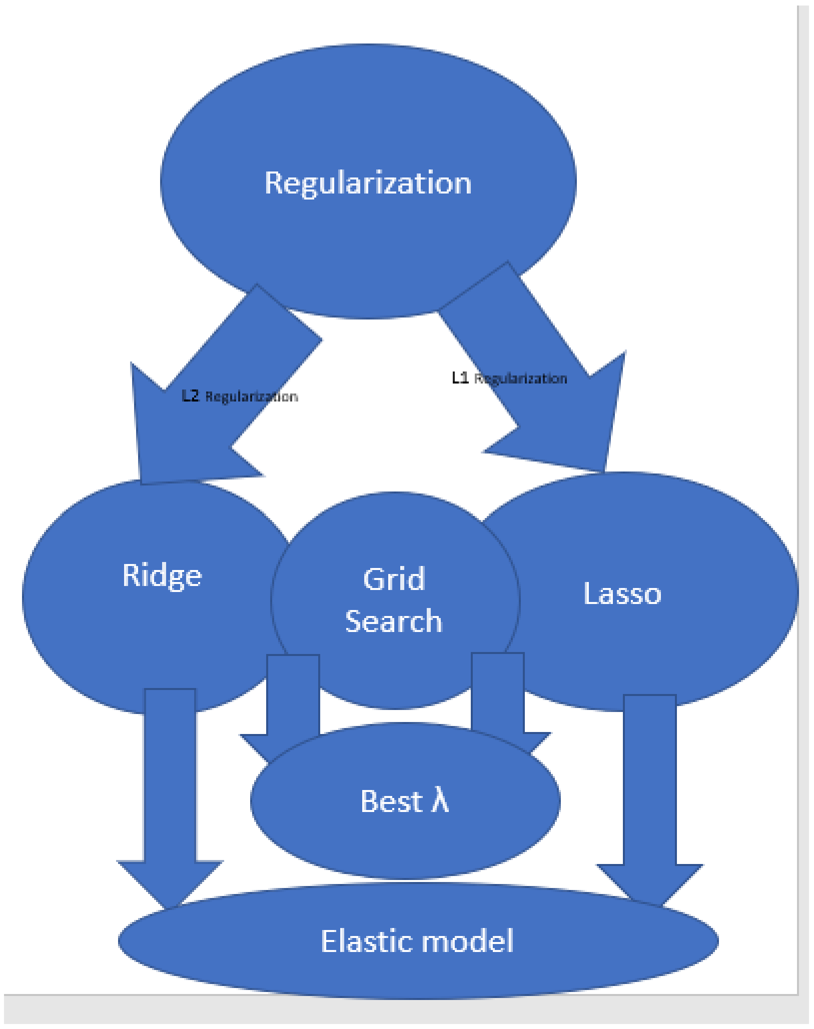
5. To inspect the Lasso's tendency to promote sparse solutions, we will create a set of synthetic data following the model below:

$$y := X\beta + \varepsilon, \ \varepsilon_i \sim \mathcal{N}(0, 1) \tag{4}$$

- Generate such a set of data, containing $n = 500$ samples and $p = 600$ predictors. Use `rnorm` to generate X and ε, and the method of your choice to generate β (e.g. `runif`). Then, set half of β's coordinates to 0 (you can set the first 300 coordinates to 0 for simplicity).
- Split the data into 60% training data and 40% test data, and use the Lasso with 5-fold and 10-fold cross validation to select the value of λ. Plot the MSE as a function of $\log(\lambda)$ and interpret the results. How many elements of β are set to 0 for the "best" value of λ found?
- Use the best value of λ found to generate a prediction over the test set, and report its MSE.
- Could you use `lm` to fit a linear model here? Explain why or why not.

Concept map. Make a **concept map** for the topics covered by this lab. While there is no single right answer, you should demonstrate that you understand the relationships between various topics in the lab.


Model selection
Enumerate all possible subset
Delete one 'worst' predictor each time
Best Subset
Forward
Backward
Add one 'best' predictor each time


Regularization
L2 Regularization
L1 Regularization
Ridge
Grid Search
Lasso
Best λ
Elastic model