lab1_assignment3

Martynas Lukosevicius, Alejo Perez Gomez, Shwetha Vandagadde Chandramouly

11/7/2020

Assignment 2

1.

Bayesian Model:
$\hat{y} \sim N(w_0 + w^t X, \sigma^2)$ where $w \sim N(0, \frac{\sigma^2}{\lambda})$

- w - weights
- X - features
- λ - regularization penalty
- σ - standard deviation

2.

Scaling data:

```
library(readr)
parkinsons <- read_csv("parkinsons.csv")
cleaned <- parkinsons[c(-1:-4, -6)]
parkinsons.scaled <- scale(cleaned)
set.seed(12345)
n <- dim(parkinsons.scaled)[1]
id=sample(1:n, floor(n*0.6))
train=parkinsons.scaled[id,]
test=parkinsons.scaled[-id,]
```

3.

Check Lecture 1d block 1 slide 16

As we will be optimizing σ and w, likelihood and prior should contain all σ, even if data is scaled (so it means that $\sigma \sim 1$):

$$log(posterior) = log(likelihood * prior) = log(likelihood) + log(prior)$$

a)

loglikelihood:

$$log(p(D|w)) = -\frac{n}{2} log(2\pi\sigma^2) - \sum_{i=1}^{n} \frac{(y_i - w^T X_i)^2}{2\sigma^2}$$

```
loglikelihood <- function(w, sigma){
n <- dim(train)[1]
part1 <- -(n/ 2) * log(2 * pi*(sigma^2))

y <- train[, 1]
x <- train[, -1]
res <- sum((y - (x %*% w))^2)

return(part1 - (res/(2*(sigma^2))))
}
```

b)

Ridge part $\sim$ log prior, where $\tau = \frac{\sigma^2}{\lambda}$, and p number of weights:

$$log(prior) = -\frac{p}{2}log(2\pi\tau) - \frac{\sum^p (w_i)^2}{2\tau}$$

function returns $-log(posterior)$

```
ridge <- function(x, lambda){
  w <- x[1:16]
  sigma <- x[17]
  p <- length(w)
  tau <- sigma^2 / lambda
  part1 <-  (-1/2) * log(2* pi * tau)
  part2 <- (w ^ 2) / (2* tau)
  ridge <- part1 - part2
  # ridge <- lambda * (w %*% w)
  return( -(loglikelihood(w,sigma) + sum(ridge)))
  #return(sum(ridge))
}
```

c)

function to predict weights (w) and σ

```
ridgeOpt <- function(lambda){
  x <- c(rep(1,16),1)
  a <- optim(x ,ridge, method = "BFGS", lambda = lambda)
  w <- a$par[1:16]
  sigma <- a$par[17]
  return(a)
}
```

d)

function to calculate degrees of freedom

```
DF <- function(lambda){
  m <- as.matrix(train[ ,-1])
  part1 <- t(m) %*% m + (lambda * diag(16))
  part2 <- m %*% solve(part1) %*% t(m)
  return(sum(diag(part2)))
}
```

4.

	MSE train	MSE test
lambda = 1	0.8747959	0.9332394
lambda = 100	0.8837090	0.9254054
lambda = 1000	0.8982641	0.9301081

$\lambda = 100$ is better than others because MSE for train set and for test set is lowest. MSE is good loss function because it comes from model's MLE.

5.

	AIC
lambda = 1	9559.988
lambda = 100	9634.181
lambda = 1000	10257.767

The optimal model is with lowest AIC score, in this case its a model with $\lambda = 1$. Hold out method requires to divide data into 3 parts: train,validation, test, which wont allow to use all data for training, its not the case with AIC, where its use on training + validation data

However im not sure about it....

Assignment 3

1.

Here we are assuming that fat can be modeled as linear regression with channels as features.

Underlying probalistic model is :— correction : see ols model in slide 1d

$$\hat{y} = \beta_0 + \sum_{i=1}^{100} \beta_i * x_i + \epsilon, \epsilon \sim \mathsf{N}(0, \sigma^2)$$

y = *dependent variable* x_i = features (channel1 to channel100) β_0 = *y-intercept* β_i = slope coefficient for each feature ϵ = model error term

Fitting linear regression model to training data

```
X_train = as.matrix(train[2:101])
Y_train = as.matrix(train$Fat)
fit_train = lm(Fat ~ . , data = train[2:102]) #fitting linear regression to training data
y_hat_train = predict(fit_train)
```

Error for training and test data :

```
## MSE for training data is  0.005709117
```

```
## MAE for training data is  0.05503807
```

```
##
## MSE for testing data  722.4294
```

```
## MAE for training data is  12.20608
```

We can see that error(MSE and MAE) for the train data was low , but when we try to fit the model to test data error increases. This is because of the overfitting on training data due to large number of features in our model. Quality of fit : Overfit. Prediction quality is not good for test data , this is evident by observing increase in MSE and MAE in test data.

Over all quality of the model is not good , as there is large number of features here , regularization is required.

2.

In this case as the number of input features are high, high degree of polynomials lead to overfitting.

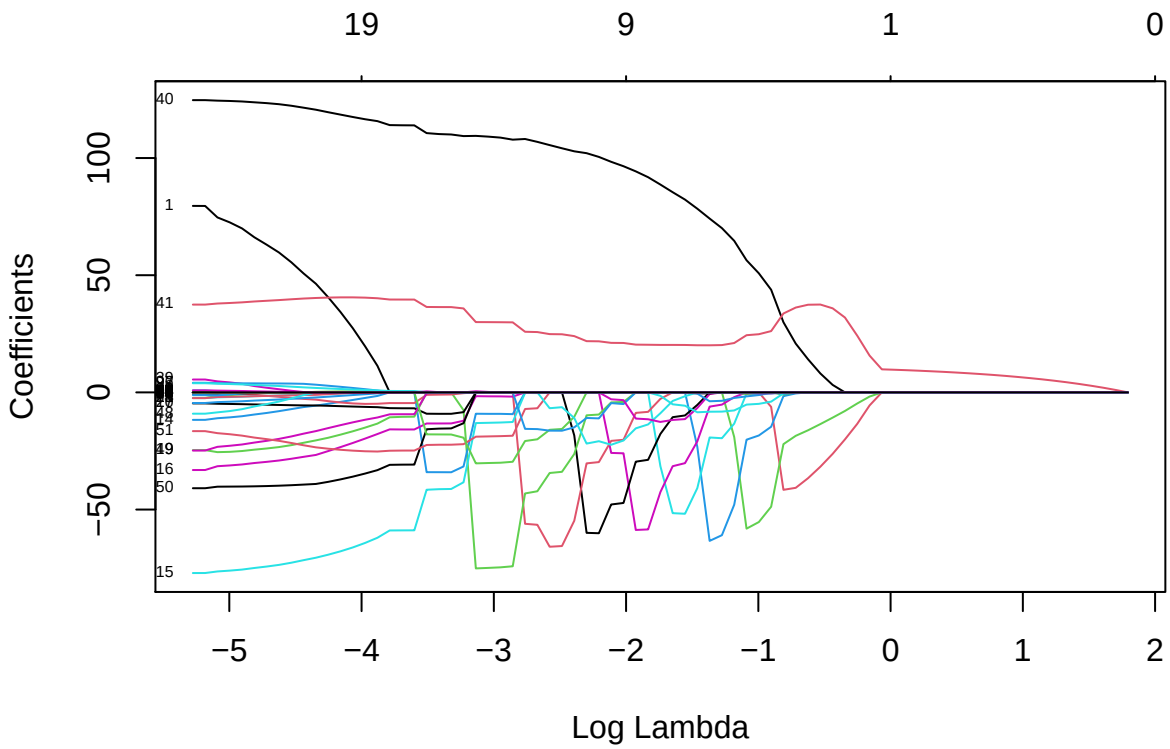
In LASSO , we try to choose the value of β such that the following loss function is minimized

$$\sum_{i=1}^{n}\left(Y_i - \beta_0 - \sum_{j=1}^{p}\beta_j X_{ji}\right)^2 + \lambda\sum_{j=1}^{p}|\beta_j|$$

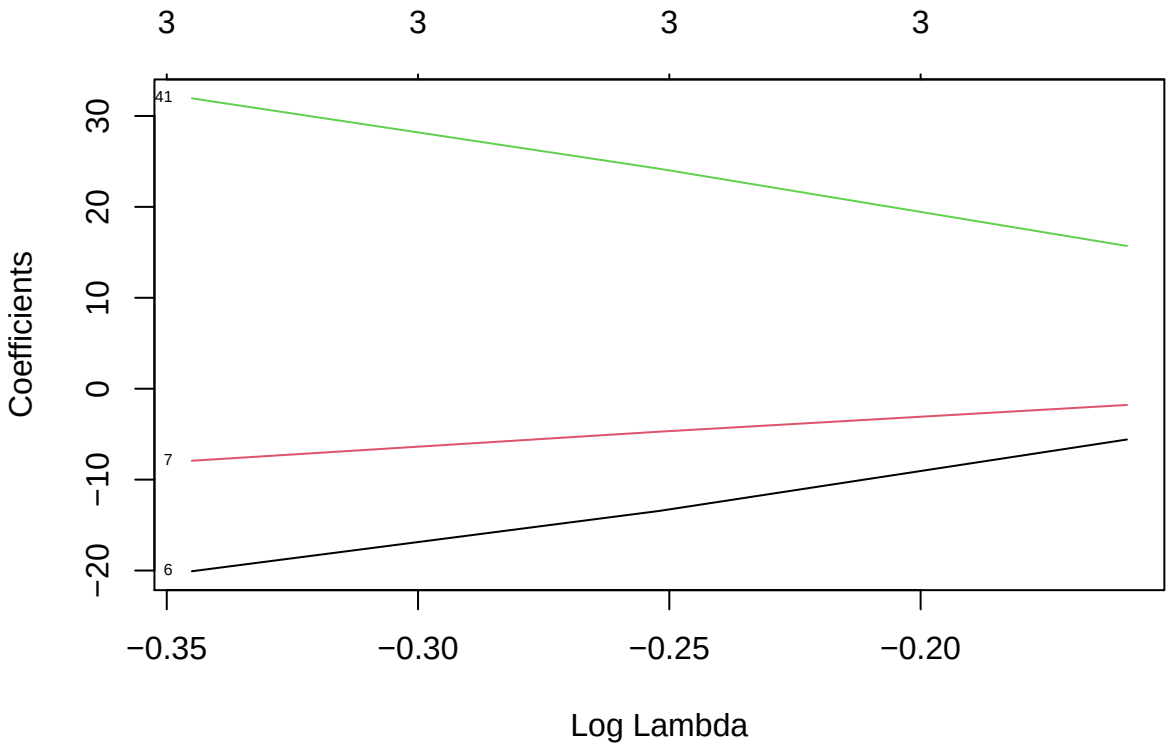
3.

Fitting LASSO regression model to training data

```
covariates = as.matrix(train[ ,2:101])
response = as.matrix(train[ ,102])
model_lasso = glmnet((covariates), response, alpha = 1, family = "gaussian")
plot(model_lasso, xvar = "lambda", label = T)
```

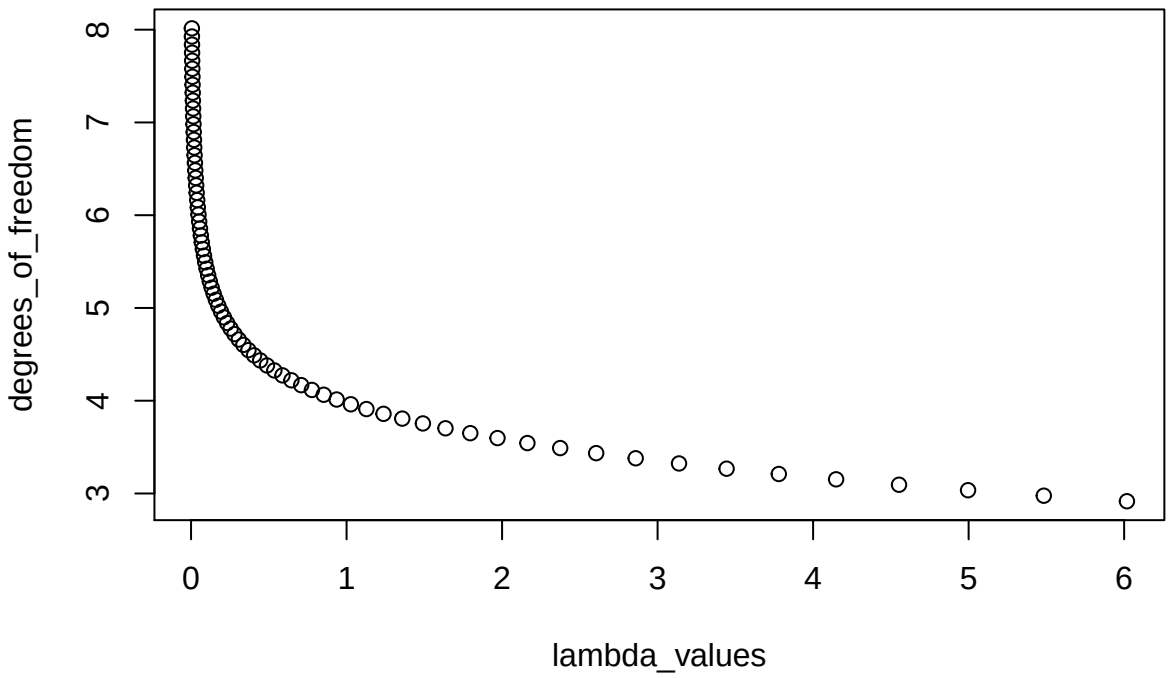


To choose lambda that has only 3 features , we can perform cross validation and check for lambda values which has only 3 variables.



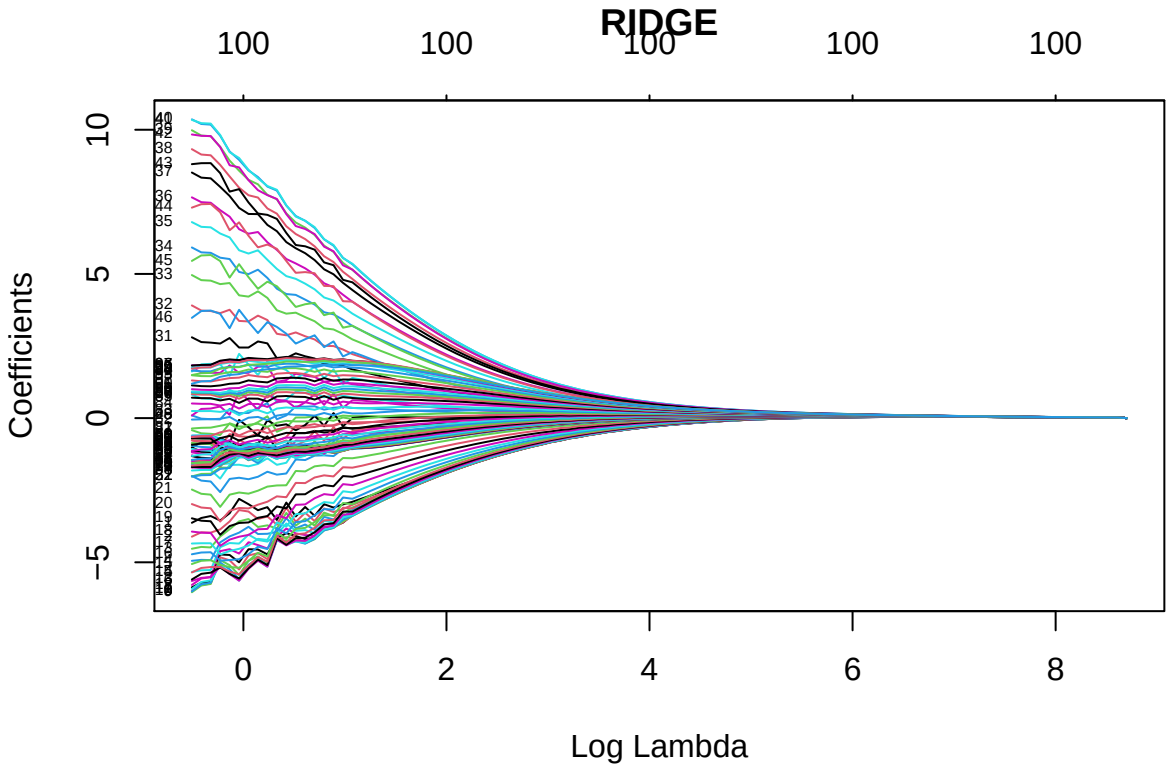

```
## Lambda values that give 3 variables are : 0.8530452 0.777263 0.7082131
```

4. Dependence of degrees of freedom on panalty factor



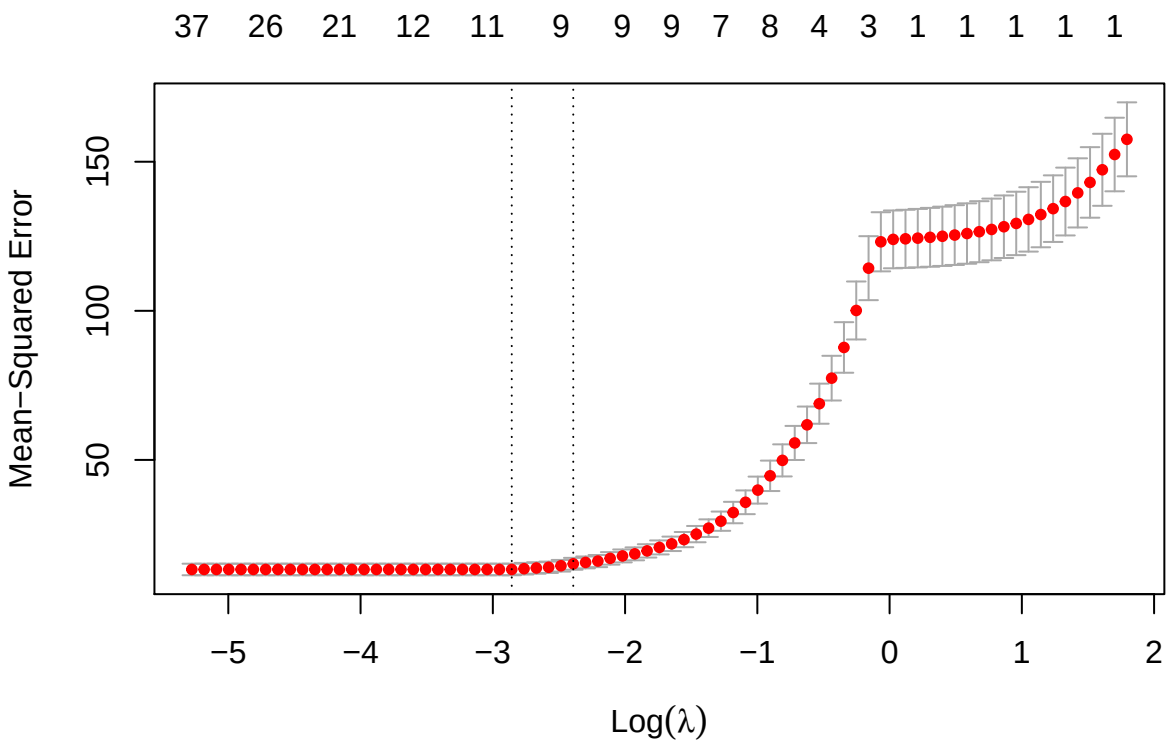
Here as expected we can see that degrees of freedom decreases as penalty factor increases. When lambda is 0 , df is infinity and when df is 0, lambda is infinity.

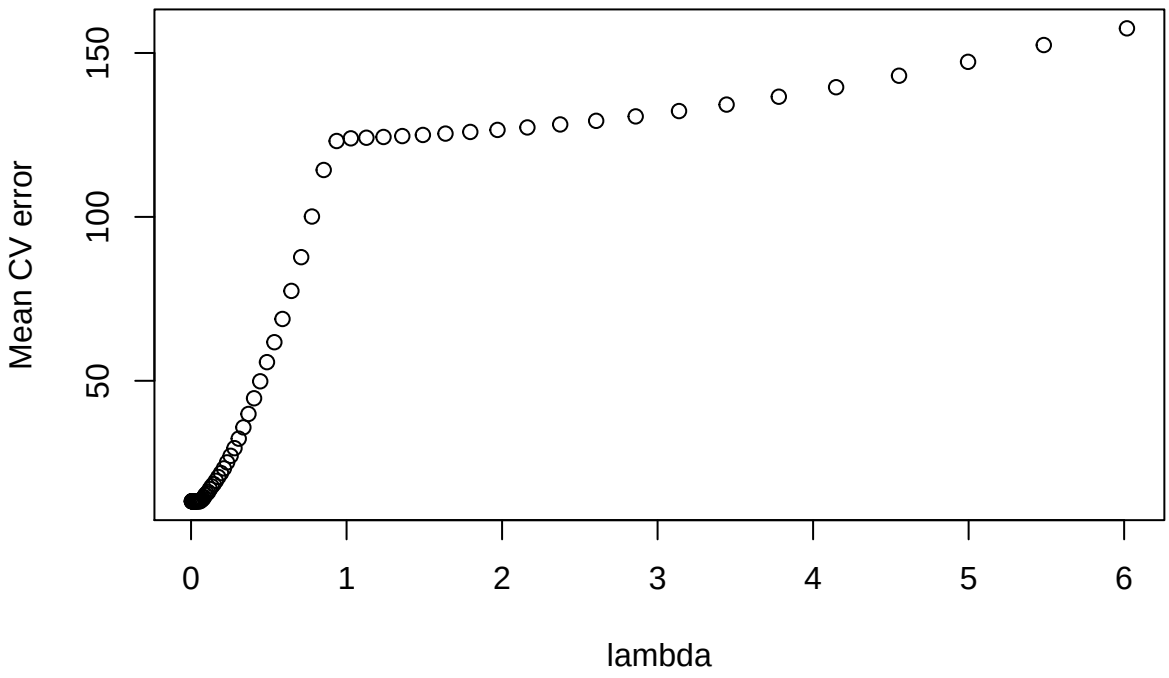
5. Fitting ridge model



In both lasso and ridge we can see that the value of coefficents decreases with increase in lambda values, But only in case of lasso , coeffients converge to 0 with increase in lambda.

6. To find optimal lambda, we use cross validation.


37 26 21 12 11 9 9 9 7 8 4 3 1 1 1 1 1
Mean−Squared Error
50
100
150
−5
−4
−3
−2
−1
0
1
2
Log(λ)




```
## Optimal lambda is : 0.05744535
```

Relationship between log(lambda) and MSE: As the lambda increases, the value of log(lambda) also increases, and we can see from the plot above that the MSE also increases with increasing lambda. Best lambda is given as zero for the above model , this makes sense as we can see from cvm(mean cross-validated error) vrs lambda plot , the error is minimum when lambda is zero.

Fitting the model for 0 and log_lambda = -2 for comparision.

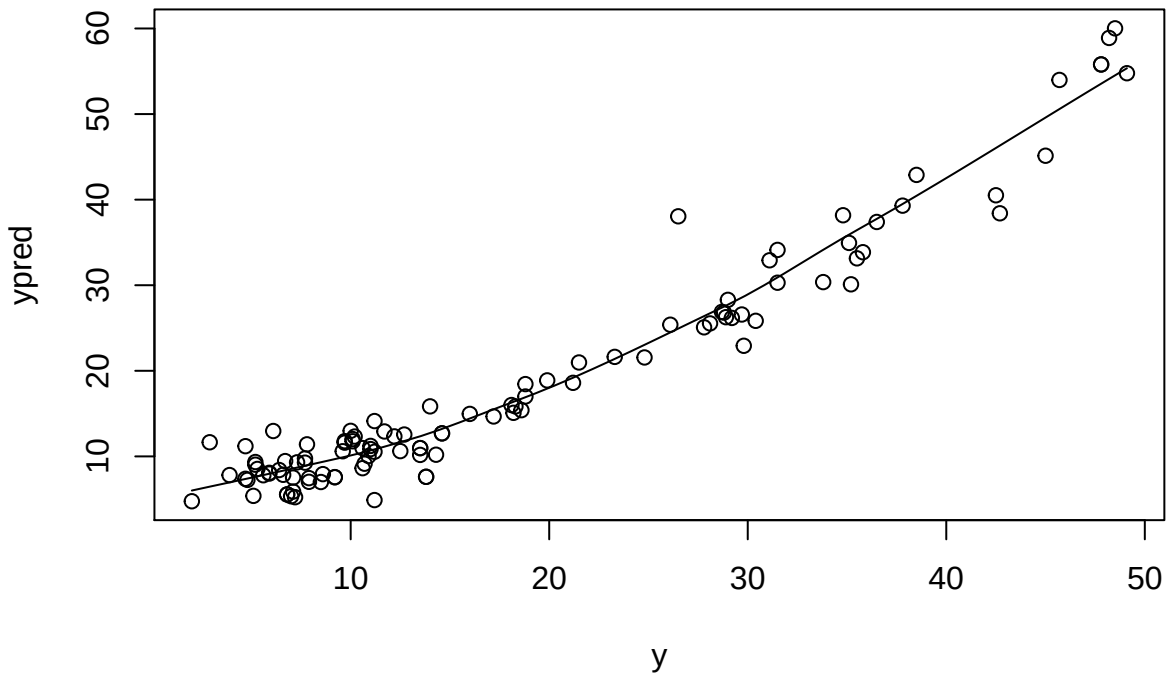
```
m = cv.glmnet((covariates), response, alpha=1,family="gaussian",lambda=c(best_lambda,0.1353353))
```

```
## CV Error when lambda is  0.1353353  = 16.72538
```

```
##  and CV Error when lambda is  0.05744535  = 12.61884
```

Log(bestlambda) ie log(0) is -Inf, from the plot of MSE vrs log(lambda) we can see that MSE is lowest for log(lambda) = -7(in the plot this is the lowest negetive number) for when compared to log(lambda) = -2. Also in lambda vrs Mean Cv error plot , we can see that for lambda = 0.1353353 (log(lambda)=-2 , so lambda = exp(-2) = 0.1353353) error is higher than error for lambda = 0.

Plot for Y vrs predicted Y for test data using optimal lambda

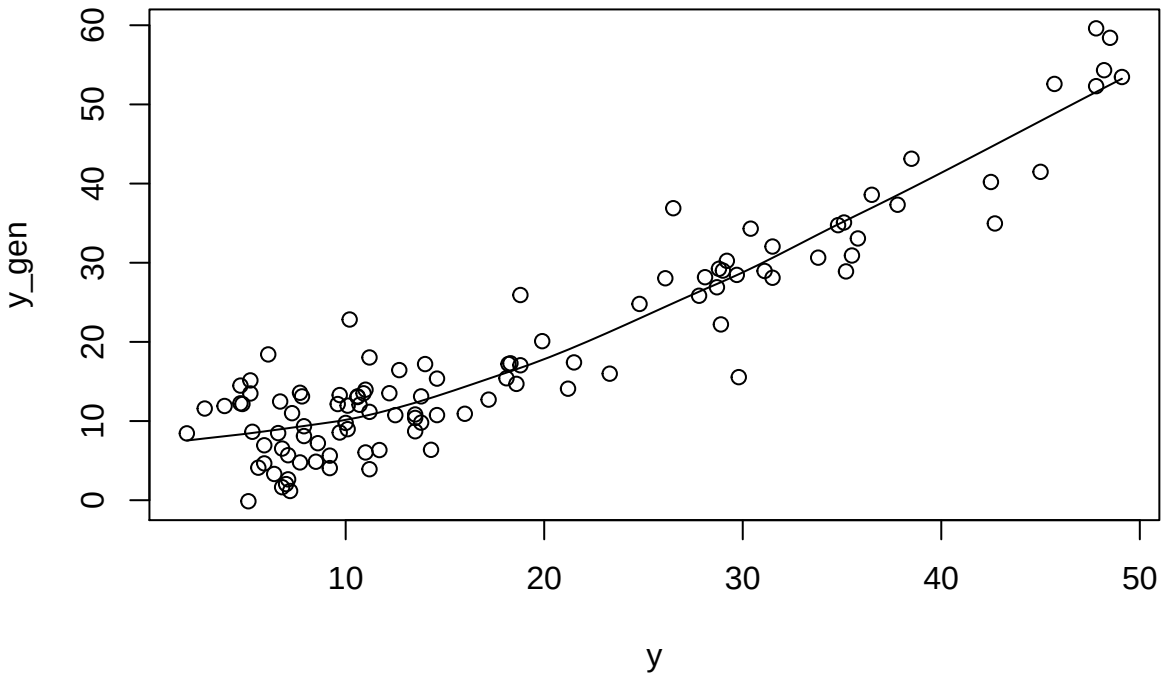



```
## coefficient of determination is  1.131442 and MSE is  13.2998
```

From the scatterplot we can see that y is similar to predited y and also the value of coefficient of determination is very close to 1 , this shows that lasso model is pretty good at predicting the Y for test data.

7.

Generating response using test data as features and optimal Lasso model




```
## coefficient of determination is  1.111923  MSE is  25.63655
```

The generated data seem to fit well as we can see value of coefficient of determination is close to 1 and MSE is also low.