

lab1_assignment3

Shwetha

11/7/2020

Assignment 3

```
library(glmnet)

## Warning: package 'glmnet' was built under R version 4.0.3

## Loading required package: Matrix

## Loaded glmnet 4.0-2

data = read.csv("C:/Users/vcshw/Machine Learning and Stats/Sem1/Machine learning/lab/tecator.csv", head=TRUE)
n = nrow(data)
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=data[id,]
test=data[-id,]
```

1.

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

Underlying probabilistic model is :

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

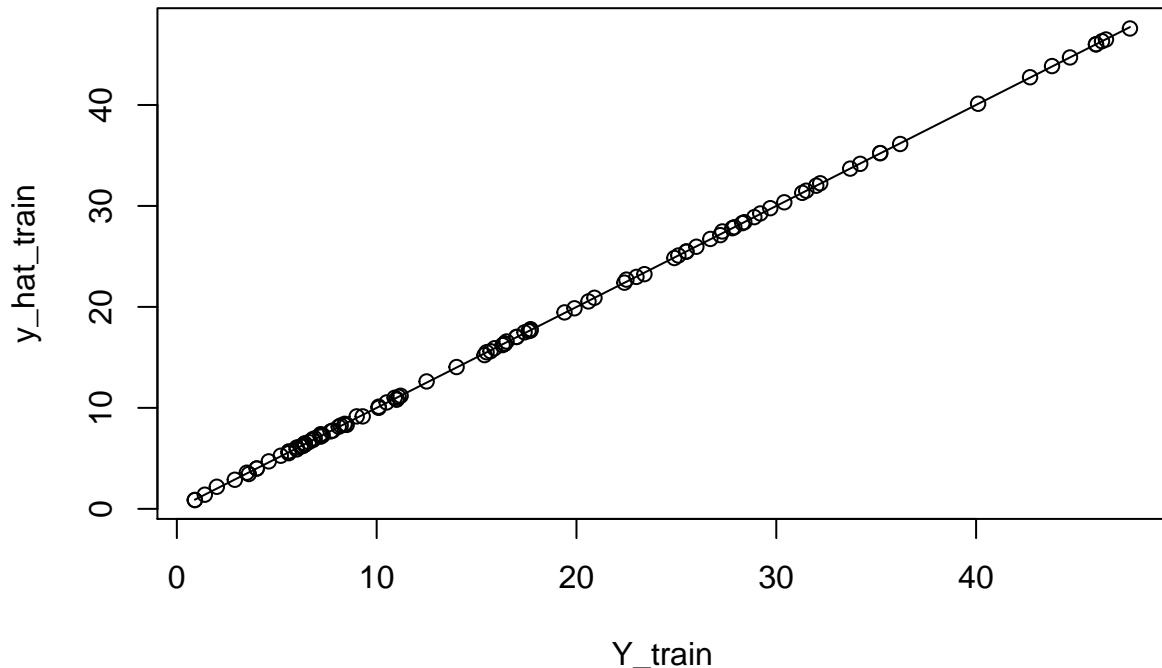
```
X_train = as.matrix(train[2:101])
Y_train = as.matrix(train$Fat)
fit_train = lm(Fat ~ . , data = train[2:102]) #fitting linear regression to training data
y_hat_train = predict(fit_train)
train_mse = mean((y_hat_train-Y_train)^2)
cat("MSE for training data is ",train_mse)
```

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

```
cat(" MAE for training data is ",mean(abs(y_hat_train - Y_train)))
```

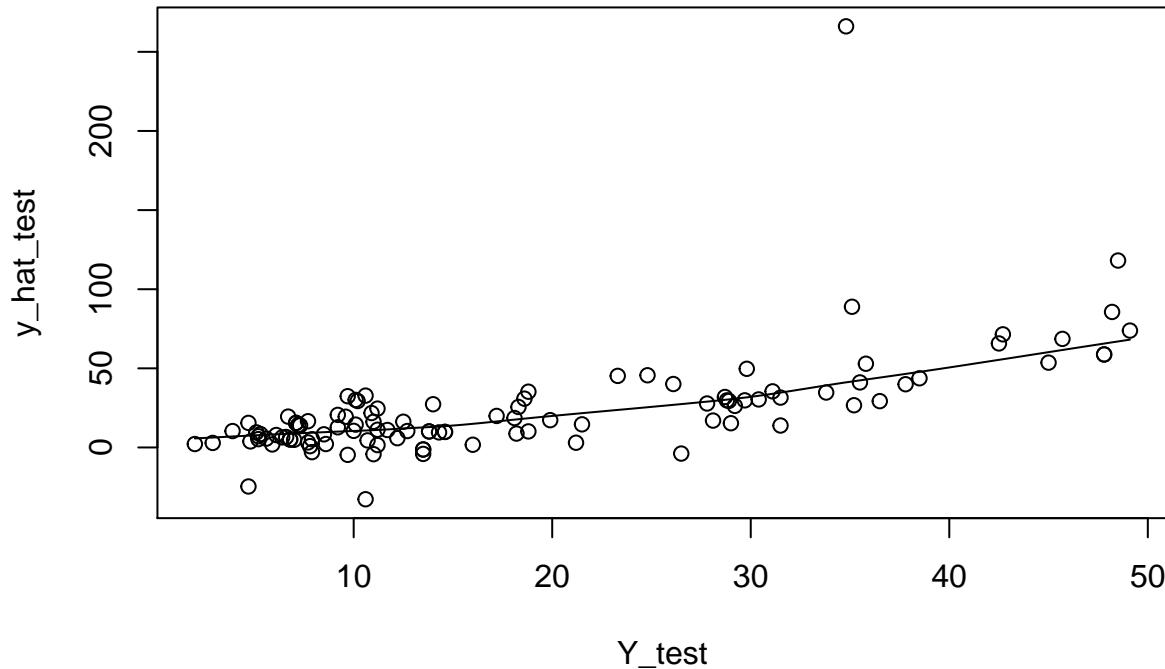
```
##  MAE for training data is  0.05503807
```

```
scatter.smooth(Y_train,y_hat_train)
```



Let us fit the same model for test data to find the test error.

```
new_x=as.matrix((test[-1, 2:101]))
Y_test = as.matrix((test[-1,102]))
y_hat_test = predict(fit_train, newdata = as.data.frame(new_x))
scatter.smooth(Y_test,y_hat_test)
```



```
test_mse = mean((y_hat_test-Y_test)^2)
cat("MSE for testing data ",test_mse)
```

```
## MSE for testing data  728.4527
```

```
cat("MAE for training data is ",mean(abs(y_hat_test - Y_test)))
```

```
## MAE for training data is  12.23764
```

We can see that error(MSE and MAE) for the train data was very less , 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 the plot of y_test vrs y_pred_test and increase in MSE and MAE when compared to train 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 input features are 100 channels, high degree of polynomials lead to overfitting. So we can keep all predictors but shrink the coefficients to make the model less complex.

$$\hat{w}_{ridge} = \underset{\beta}{\operatorname{argmin}} \left[\sum_{i=1}^n \left(Y_i - \beta_0 - \sum_{j=1}^p \beta_j X_{ji} \right)^2 + \lambda \sum_{j=1}^p |\beta_j| \right]$$

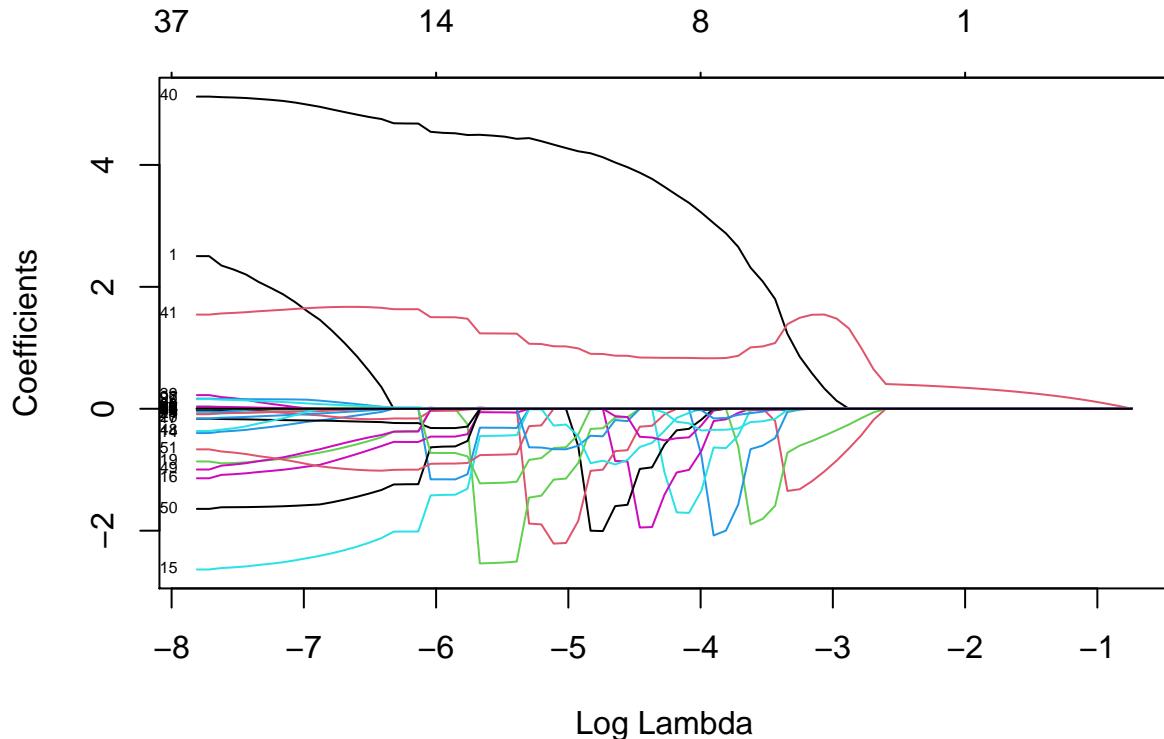
3.

Fitting LASSO regression model to training data

```

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

```



Choosing lambda so that there are only 3 features.

```

model=cv.glmnet(as.matrix(covariates),
response, alpha=1,family="gaussian")
which(model$nonzero == 3) #here we can see that for lambda from s21 to s22 , non zero coefficients are 3.

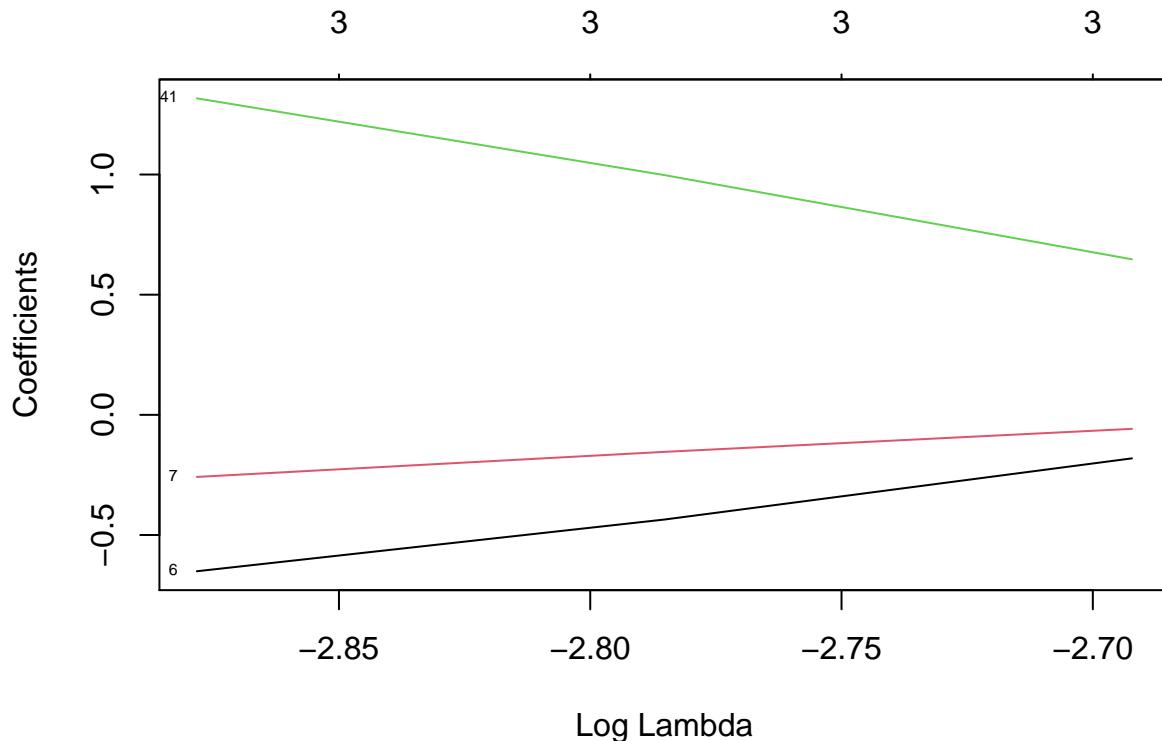
## s21 s22 s23
## 22 23 24

model$lambda[22:24] # these are the lambda values for which only 3 variables will be selected

## [1] 0.06773096 0.06171393 0.05623144

model_lasso_3 = glmnet(as.matrix(covariates), response, alpha = 1, family = "gaussian", lambda = c(0.06773096, 0.06171393, 0.05623144))
plot(model_lasso_3, xvar = "lambda", label = T)

```



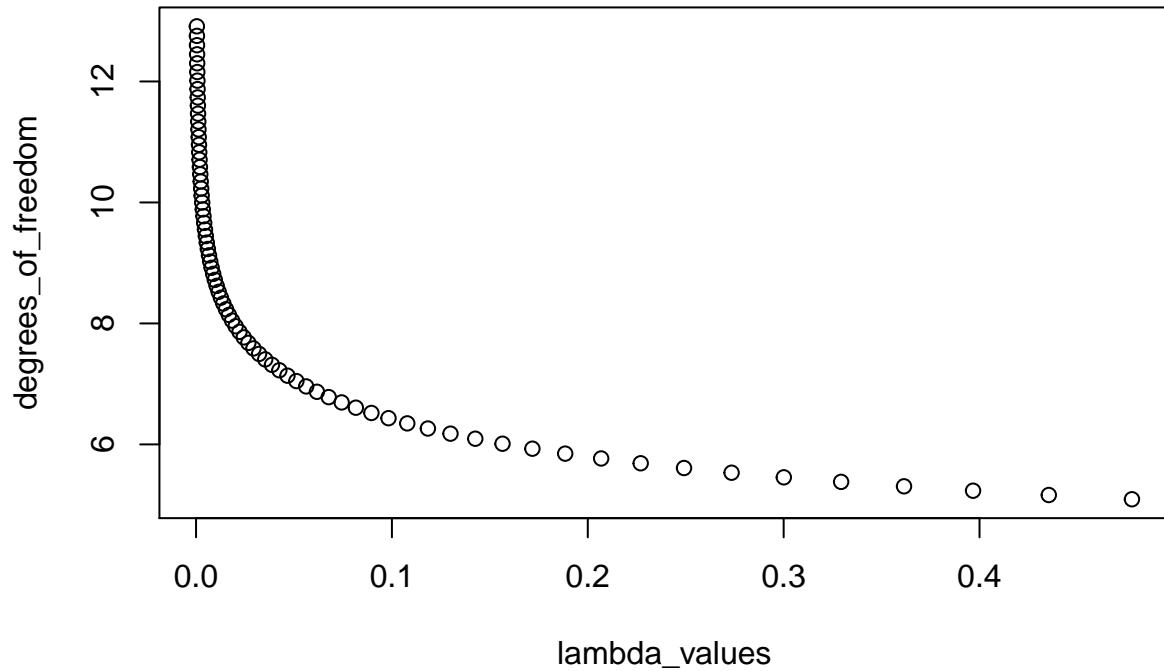
4.

Dependence of degrees of freedom on penalty factor

```

df = function(lambda){
  ld = lambda * diag(ncol(covariates))
  H = covariates %*% solve(t(covariates) %*% covariates + ld) %*% t(covariates)
  DOF = sum(diag(H))
  return(DOF)
}
lambda_values = model_lasso$lambda
degrees_of_freedom = c()
for(i in 1:length(lambda_values)){
  degrees_of_freedom[i] = df(lambda_values[i])
}
plot(lambda_values, degrees_of_freedom)

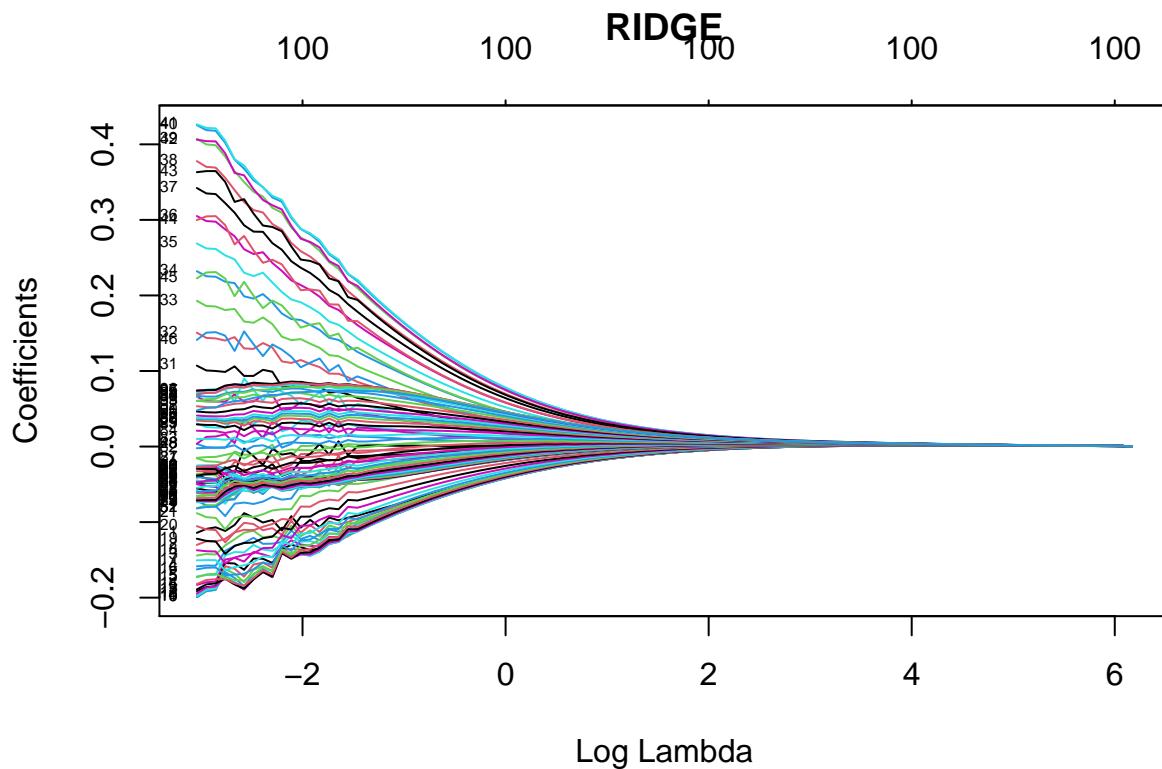
```



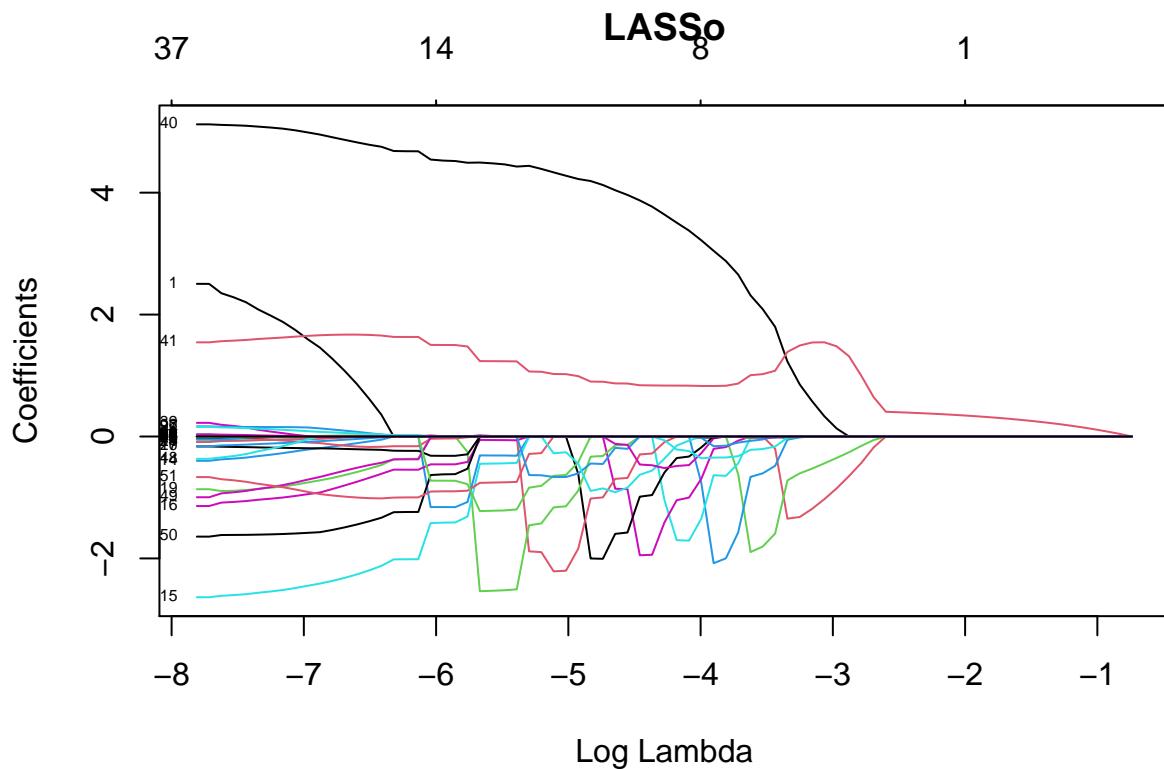
Here as expected we can see that degrees of freedom decreases as penalty factor increases.

5. Fitting ridge model

```
covariates = scale(train[, 2:101])
response = scale(train[, 102])
model_ridge = glmnet(as.matrix(covariates), response, alpha = 0, family = "gaussian")
plot(model_ridge, main = "RIDGE", xvar = "lambda", label = T)
```



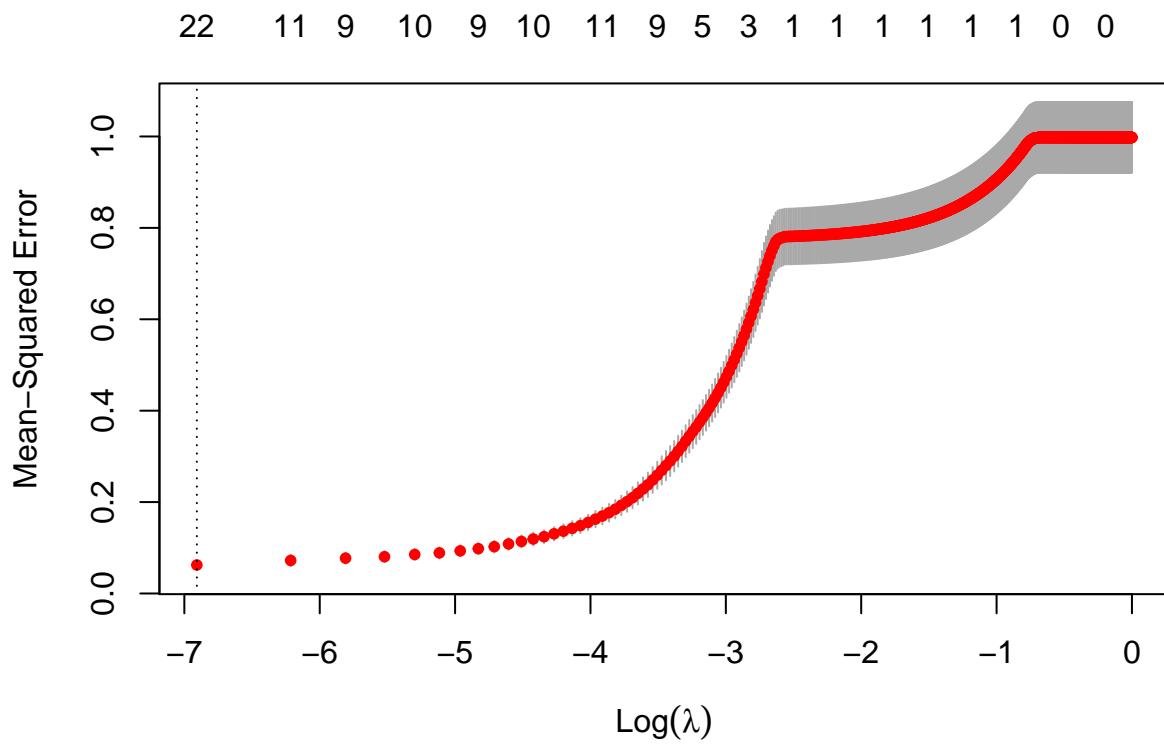
```
plot(model_lasso, xvar = "lambda", label = T, main = "LASSo")
```



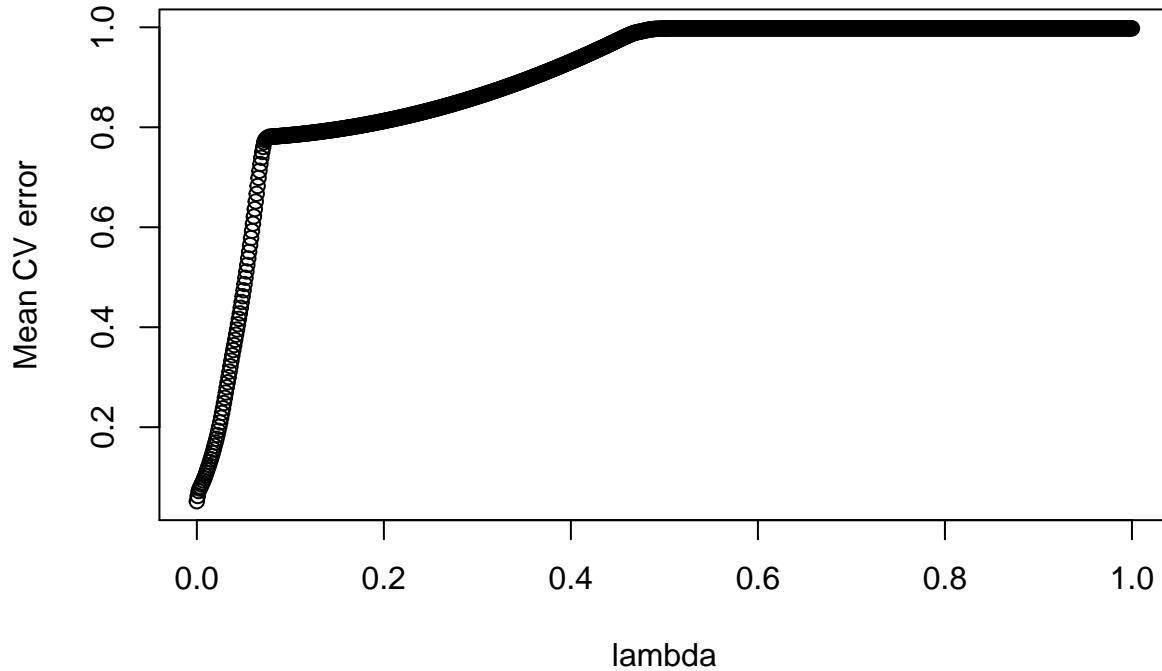
In both lasso and ridge we can see that the value of coefficients decreases with increase in lambda values, But only in case of lasso , coefficients converge to 0 with increase in lambda.

6.

```
model_1=cv.glmnet(as.matrix(covariates), response, alpha=1,family="gaussian",lambda=seq(0,1,0.001))
best_lambda = model_1$lambda.min
plot(model_1)
```



```
plot(model_1$lambda,model_1$cvm,xlab="lambda",ylab="Mean CV error")
```



```
cat("Optimal lambda is :",best_lambda)
```

```
## Optimal lambda is : 0
```

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 , this makes sense as we can see from cvm(mean cross-validated error) vrs lambda plot , the error is minimum when lambda is zero.

```
m = cv.glmnet(as.matrix(covariates), response, alpha=1,family="gaussian",lambda=c(best_lambda,0.1353353))
cat("CV Error when lambda is ",m$lambda[1] , " =",m$cvm[1])
```

```
## CV Error when lambda is  0.1353353  = 0.8133424
```

```
cat(" and CV Error when lambda is ",m$lambda[2] , " =",m$cvm[2])
```

```
##  and CV Error when lambda is  0  = 0.06861109
```

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 negative 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

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

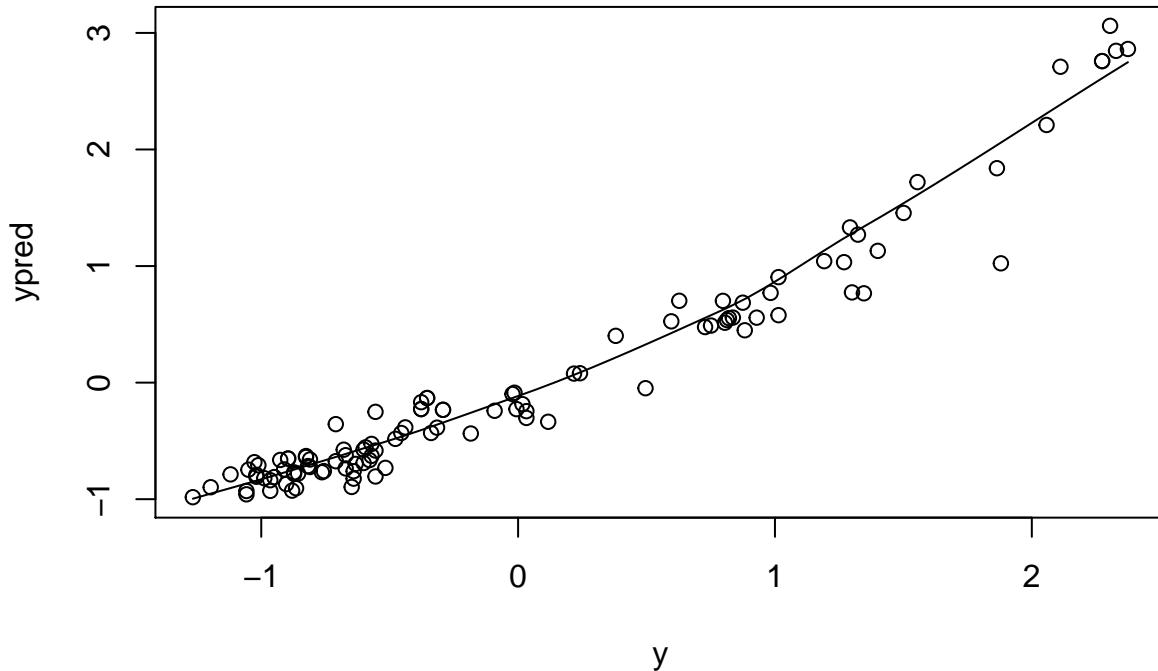
```

model_optimallasso = glmnet(as.matrix(covariates), response, alpha = 1, family = "gaussian", lambda = 1)
y = as.matrix(scale(test[,102]))
ypred=predict(model_optimallasso, newx=as.matrix(scale(test[, 2:101])), type="response")
cat("coefficient of determination is ",
sum((ypred-mean(y))^2)/sum((y-mean(y))^2), " MSE is ", mean((y-ypred)^2))

## coefficient of determination is  0.9889883  MSE is  0.06737877

scatter.smooth(y,ypred)

```



From the scatterplot we can see that y is similar to predicted \hat{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.

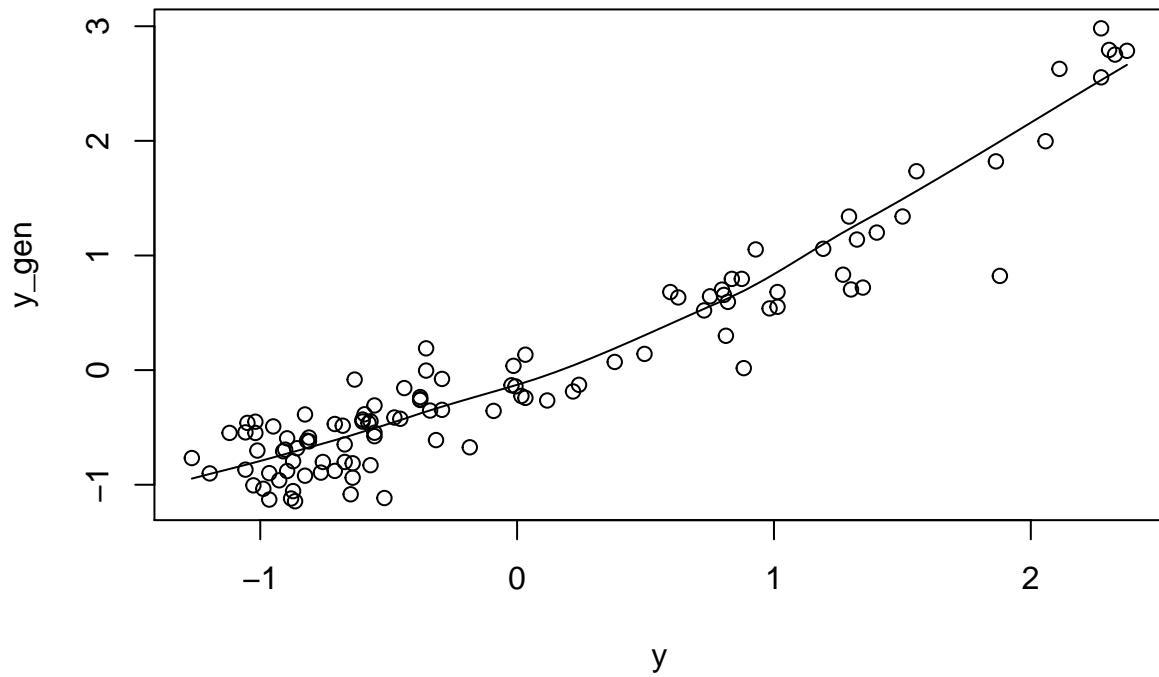
7.

```

betas = as.vector((as.matrix(coef(model_optimallasso))[-1, ])) # removing the first row for intercept
resid = response - (covariates %*% betas)
sigma = sd(resid)

ypred=predict(model_optimallasso, newx=as.matrix(scale(test[, 2:101])), type="response")
set.seed(90000)
y_gen = rnorm(108,ypred,sigma)
scatter.smooth(y,y_gen)

```



```
cat("coefficient of determination is ",sum((y_gen-mean(y))^2)/sum((y-mean(y))^2), " MSE is ",mean((y_gen
```

```
## coefficient of determination is  0.9572077  MSE is  0.1063286
```

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.