Report for Final Project Task 1

Ranveer Kaur and Gabriel Afriyie

14/03/2020

Introduction

In task 1 of our project, we were asked to predict 3 unknown responses $Y1$, $Y2$ and $Y3$ of 100 observations. We were provided with a training dataset of 5000 with 50 predictor variables $(X1, \ldots, X50)$. $Y1$ and $Y2$ are continuous variables and $Y3$ is a categorical variable with 2 levels (0 and 1). The main objective of task 1 was to apply various regression and classification methods on the training dataset to predict ($Y1$, $Y2$) and $Y3$, respectively. The best method is selected based on their cross validation error (CV error).

Model Fitting and Predictions For $Y1$

In this section, we seek to predict $Y1$ in the training data using test data. The following models will be considered: Linear regression model, Ridge regression model, Lasso rgression model and Regression trees. We begin by loading the following packages that will be useful in our analysis.

```
library(dplyr)
library(leaps)
library(glmnet)
library(MASS)
library(DAAG)
library(rpart)
library(e1071)
library(rpart.plot)
```

We also load the training and test datasets.

```
train <- read.csv("C:/Users/gafri/Desktop/Winter 2020/Statistical learning/Project/train.csv",
                  header=TRUE)
test <- read.csv("C:/Users/gafri/Desktop/Winter 2020/Statistical learning/Project/test.csv",
                 header=TRUE)
```

Firstly, we fit a linear model of $Y1$ on the predictor variables $X1, \ldots, X50$ and calculate the cross validation error with 10 folds.

```
Linearmodel = lm(Y1~., data = train[,-c(2,3)])
summary(Linearmodel)
```

```
##
## Call:
## lm(formula = Y1 ~ ., data = train[, -c(2, 3)])
##
## Residuals:
##      Min       1Q   Median       3Q      Max
```

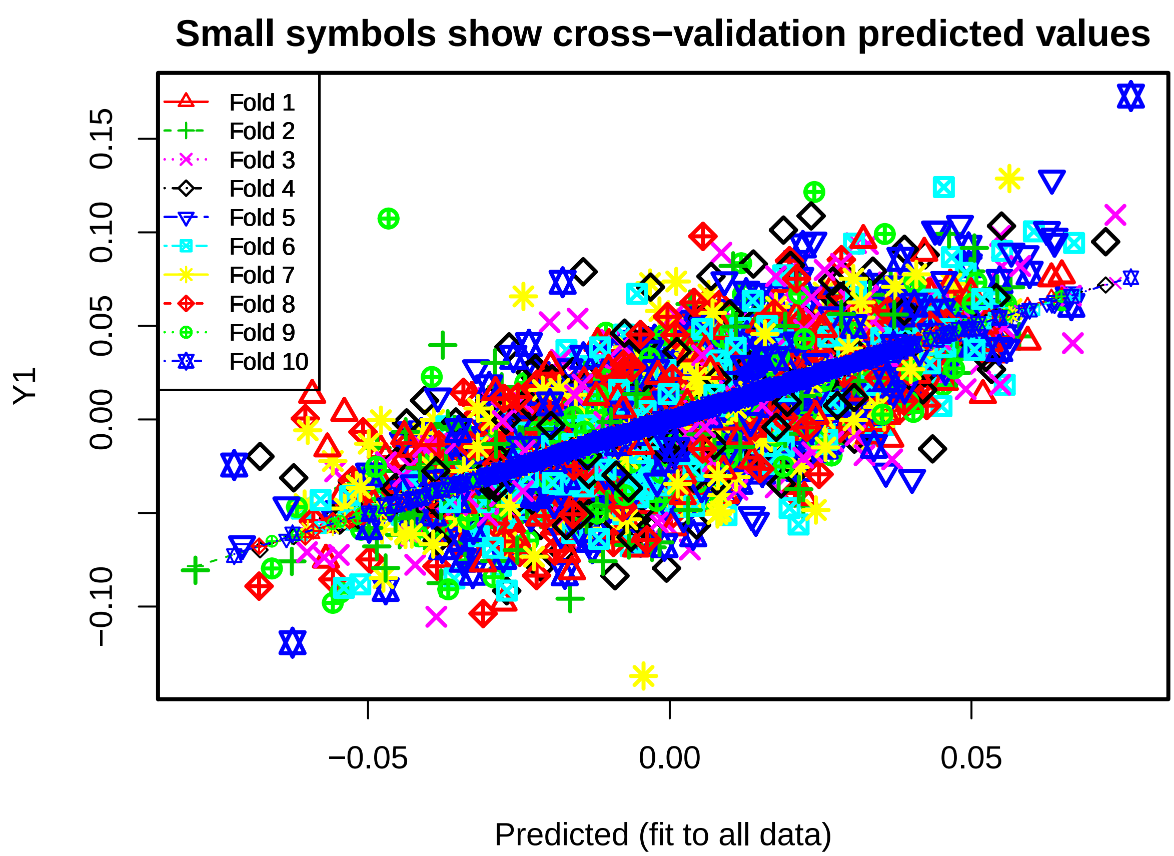
```
## -0.13278 -0.01236 -0.00011  0.01209  0.15400
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.0019248  0.0003054   6.303 3.18e-10 ***
## X1           0.0054017  0.0061483   0.879  0.37968
## X2          -0.0002369  0.0052270  -0.045  0.96385
## X3          -0.0063320  0.0167820  -0.377  0.70596
## X4           0.0046155  0.0239672   0.193  0.84730
## X5          -0.0213607  0.0139872  -1.527  0.12679
## X6           0.0017642  0.0096034   0.184  0.85425
## X7          -0.0144405  0.0070892  -2.037  0.04171 *
## X8           0.0331511  0.0163882   2.023  0.04314 *
## X9           0.0128698  0.0191205   0.673  0.50092
## X10          0.0098954  0.0050276   1.968  0.04910 *
## X11         -0.0050611  0.0175119  -0.289  0.77259
## X12          0.0149720  0.0086622   1.728  0.08397 .
## X13         -0.0067700  0.0041335  -1.638  0.10151
## X14          0.0185204  0.0108901   1.701  0.08907 .
## X15          0.0062983  0.0048835   1.290  0.19721
## X16          0.0112643  0.0093171   1.209  0.22672
## X17         -0.0054312  0.0056791  -0.956  0.33894
## X18         -0.0057369  0.0081960  -0.700  0.48399
## X19         -0.0026744  0.0091120  -0.294  0.76915
## X20         -0.0093561  0.0053802  -1.739  0.08210 .
## X21          0.0113238  0.0112033   1.011  0.31218
## X22          0.0029022  0.0055259   0.525  0.59947
## X23          0.0205498  0.0049134   4.182 2.93e-05 ***
## X24          0.0085589  0.0073442   1.165  0.24391
## X25          0.0100453  0.0096734   1.038  0.29911
## X26         -0.0084899  0.0035674  -2.380  0.01736 *
## X27          0.0059927  0.0169838   0.353  0.72422
## X28          0.0112555  0.0101205   1.112  0.26613
## X29         -0.0333354  0.0117116  -2.846  0.00444 **
## X30         -0.0022070  0.0104068  -0.212  0.83206
## X31         -0.0041420  0.0086851  -0.477  0.63345
## X32         -0.0059918  0.0128686  -0.466  0.64151
## X33          0.0083516  0.0230454   0.362  0.71707
## X34          0.0094415  0.0160502   0.588  0.55639
## X35         -0.0284453  0.0156362  -1.819  0.06894 .
## X36         -0.0099982  0.0110403  -0.906  0.36518
## X37          0.0109565  0.0069118   1.585  0.11299
## X38          0.0317835  0.0066316   4.793 1.69e-06 ***
## X39         -0.0018097  0.0032064  -0.564  0.57249
## X40         -0.0026119  0.0089681  -0.291  0.77088
## X41         -0.0046801  0.0049949  -0.937  0.34882
## X42          0.0020794  0.0082497   0.252  0.80100
## X43         -0.0131700  0.0225169  -0.585  0.55865
## X44          0.0022743  0.0102544   0.222  0.82449
## X45         -0.0103983  0.0193917  -0.536  0.59183
## X46         -0.0138834  0.0159366  -0.871  0.38371
## X47         -0.0046873  0.0040831  -1.148  0.25103
## X48         -0.0001672  0.0215560  -0.008  0.99381
## X49          0.0041729  0.0237150   0.176  0.86033
```

```
## X50          0.0142643  0.0190632   0.748  0.45434
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.02148 on 4949 degrees of freedom
## Multiple R-squared:  0.5145, Adjusted R-squared:  0.5096
## F-statistic: 104.9 on 50 and 4949 DF,  p-value: < 2.2e-16
```

We test the hypothesis: $H_0 : \beta_1 = \beta_2 = \cdots = \beta_{50} = 0$
H_1 :at least one of $\beta_i \neq 0$ $i = 1, 2, \ldots, 50$

We reject H_0 at $\alpha = 0.05$ because the $p-$value$< \alpha$. From the summary table we see that intercept and the variables $X7, X8, X10, X23, X26, X29, X38$ are all significant, since their $p-$values are less than 0.05. So, our final linear model will contain these predictors. $R^2 = 0.5145$ means that approximately, 51% of the total variability in $Y1$ is explained by the model.

```
cv_error = cv.lm(train, Linearmodel, m = 10, plotit = TRUE, printit = FALSE)
```



The cv error for our linear model is 0.000468 which is sufficiently small. The plot gives us the cross validation predicted values for the 10 different folds.

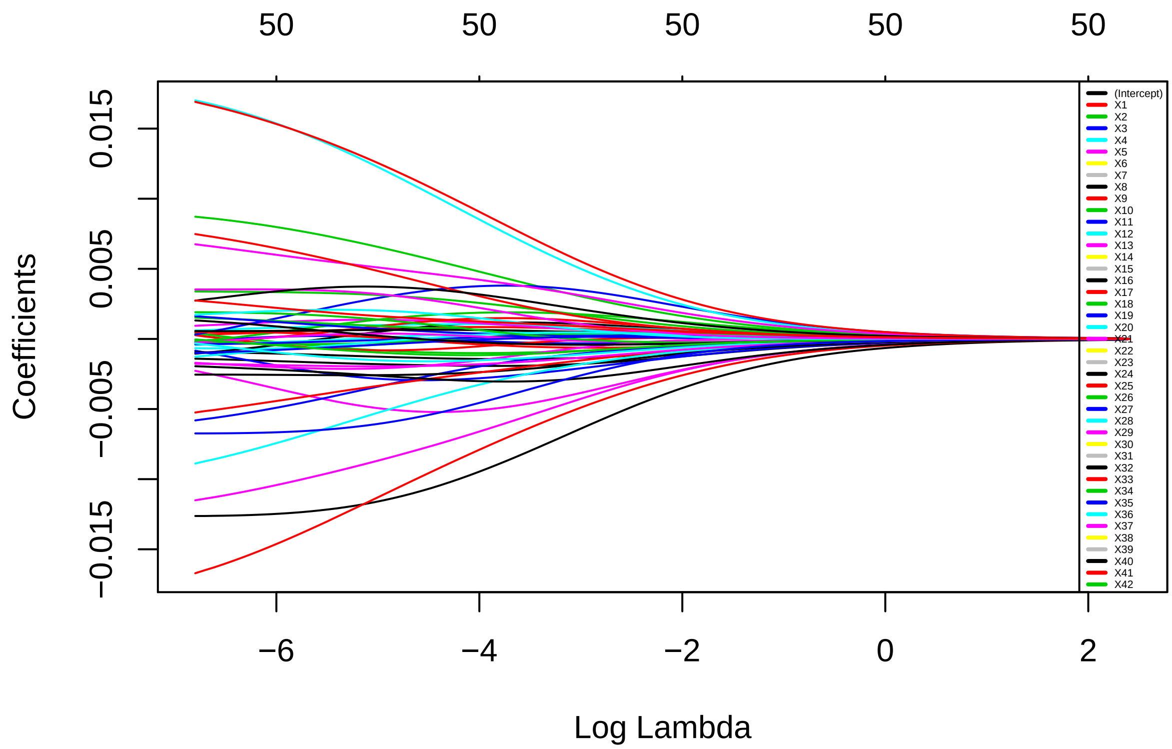
We now fit the ridge regression model.

```
X = model.matrix(Y1~., train[,-c(2:3)])
Y = train$Y1
ridgemod = glmnet(X,Y, alpha = 0)
ridgemod
```

```
##
## Call:  glmnet(x = X, y = Y, alpha = 0)
##
##      Df    %Dev  Lambda
## 1    50 0.00000 11.1500
## 2    50 0.00555 10.1500
## 3    50 0.00609  9.2530
## 4    50 0.00668  8.4310
## 5    50 0.00732  7.6820
## 6    50 0.00802  6.9990
## 7    50 0.00879  6.3780
## 8    50 0.00963  5.8110
## 9    50 0.01055  5.2950
## 10   50 0.01155  4.8240
## 11   50 0.01265  4.3960
## 12   50 0.01385  4.0050
## 13   50 0.01516  3.6500
## 14   50 0.01659  3.3250
## 15   50 0.01816  3.0300
## 16   50 0.01986  2.7610
## 17   50 0.02171  2.5150
## 18   50 0.02373  2.2920
## 19   50 0.02593  2.0880
## 20   50 0.02832  1.9030
## 21   50 0.03092  1.7340
## 22   50 0.03373  1.5800
## 23   50 0.03679  1.4390
## 24   50 0.04010  1.3120
## 25   50 0.04368  1.1950
## 26   50 0.04754  1.0890
## 27   50 0.05171  0.9922
## 28   50 0.05621  0.9040
## 29   50 0.06104  0.8237
## 30   50 0.06623  0.7505
## 31   50 0.07180  0.6839
## 32   50 0.07775  0.6231
## 33   50 0.08410  0.5677
## 34   50 0.09086  0.5173
## 35   50 0.09805  0.4714
## 36   50 0.10570  0.4295
## 37   50 0.11370  0.3913
## 38   50 0.12220  0.3566
## 39   50 0.13110  0.3249
## 40   50 0.14040  0.2960
## 41   50 0.15020  0.2697
## 42   50 0.16030  0.2458
## 43   50 0.17080  0.2239
## 44   50 0.18170  0.2040
## 45   50 0.19280  0.1859
## 46   50 0.20430  0.1694
## 47   50 0.21600  0.1543
## 48   50 0.22790  0.1406
## 49   50 0.23990  0.1281
## 50   50 0.25200  0.1168
```

```
## 51  50 0.26420  0.1064
## 52  50 0.27640  0.0969
## 53  50 0.28850  0.0883
## 54  50 0.30060  0.0805
## 55  50 0.31250  0.0733
## 56  50 0.32410  0.0668
## 57  50 0.33560  0.0609
## 58  50 0.34670  0.0555
## 59  50 0.35750  0.0505
## 60  50 0.36800  0.0460
## 61  50 0.37810  0.0420
## 62  50 0.38780  0.0382
## 63  50 0.39700  0.0348
## 64  50 0.40580  0.0317
## 65  50 0.41420  0.0289
## 66  50 0.42210  0.0264
## 67  50 0.42960  0.0240
## 68  50 0.43660  0.0219
## 69  50 0.44320  0.0199
## 70  50 0.44930  0.0182
## 71  50 0.45510  0.0166
## 72  50 0.46040  0.0151
## 73  50 0.46530  0.0137
## 74  50 0.46980  0.0125
## 75  50 0.47400  0.0114
## 76  50 0.47780  0.0104
## 77  50 0.48130  0.0095
## 78  50 0.48460  0.0086
## 79  50 0.48750  0.0079
## 80  50 0.49020  0.0072
## 81  50 0.49260  0.0065
## 82  50 0.49480  0.0059
## 83  50 0.49690  0.0054
## 84  50 0.49870  0.0049
## 85  50 0.50030  0.0045
## 86  50 0.50180  0.0041
## 87  50 0.50320  0.0037
## 88  50 0.50440  0.0034
## 89  50 0.50550  0.0031
## 90  50 0.50650  0.0028
## 91  50 0.50740  0.0026
## 92  50 0.50820  0.0023
## 93  50 0.50890  0.0021
## 94  50 0.50950  0.0019
## 95  50 0.51010  0.0018
## 96  50 0.51060  0.0016
## 97  50 0.51100  0.0015
## 98  50 0.51140  0.0013
## 99  50 0.51170  0.0012
## 100 50 0.51200  0.0011
```

```r
plot(ridgemod, xvar = "lambda")
legend("topright", lwd = 2, col = 1:50, legend = colnames(X), cex = .3046)
```

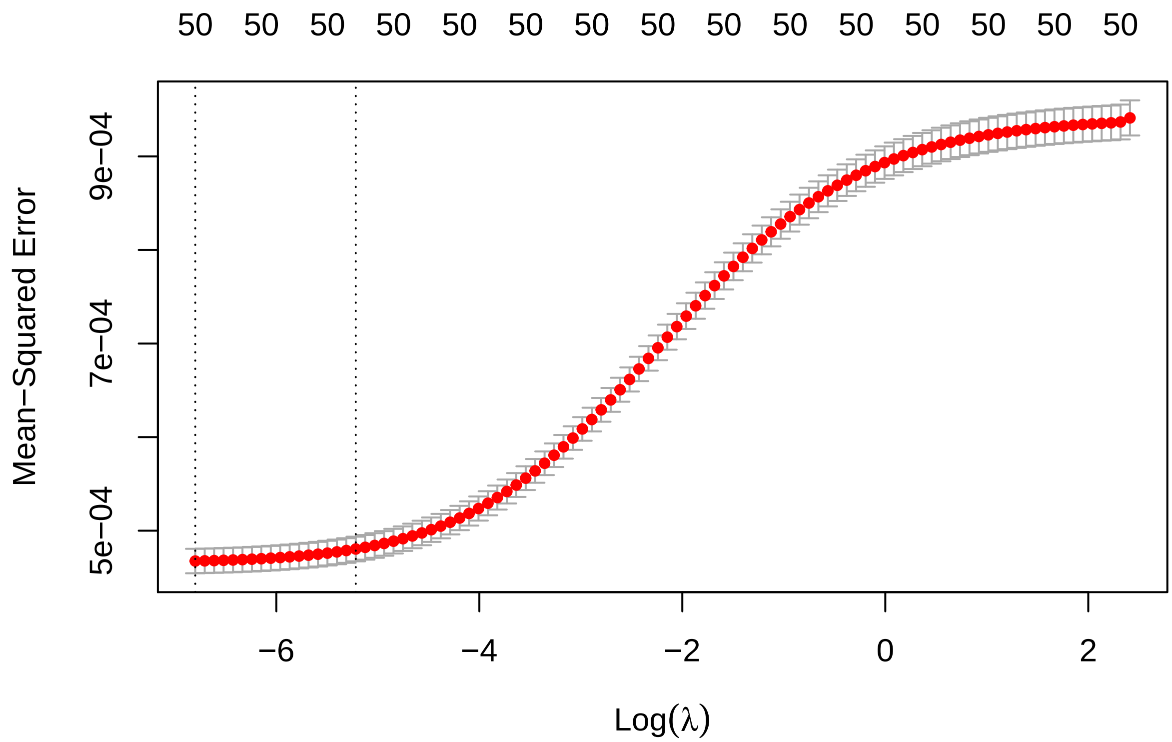


The ridge regression penalizes the coefficients, such that those who are the least efficient in our estimation will "shrink" the fastest. As the λ increases, we are penalizing more. From the plot, each line represents a coefficient whose value is going to zero as λ increases. The faster a coefficient is shrinking the less important it is in prediction. To choose the best λ we consult the MSE vs λ plot. This is obtained by performing a cross validation.

```
cv.out = cv.glmnet(X,Y,alpha=0)
cv.out
```

```
##
## Call:  cv.glmnet(x = X, y = Y, alpha = 0)
##
## Measure: Mean-Squared Error
##
##       Lambda   Measure        SE Nonzero
## min 0.001115 0.0004676 1.305e-05      50
## 1se 0.005419 0.0004804 1.317e-05      50
```

```
plot(cv.out)
```




```r
coef(cv.out)
```

```
## 52 x 1 sparse Matrix of class "dgCMatrix"
##                          1
## (Intercept)  2.020259e-03
## (Intercept)  .
## X1          -1.758024e-03
## X2           6.759063e-04
## X3          -7.122339e-04
## X4           1.872014e-04
## X5          -5.003499e-04
## X6          -9.088808e-03
## X7           3.122083e-04
## X8          -7.851472e-04
## X9           6.909997e-03
## X10         -2.705886e-03
## X11          9.528740e-04
## X12          1.366406e-03
## X13         -1.216039e-03
## X14          7.276915e-04
## X15          1.202656e-03
## X16          2.628976e-03
## X17          1.127282e-05
## X18         -4.711752e-03
## X19         -1.188349e-02
## X20          1.925776e-04
```

```
## X21          -3.144969e-04
## X22          -3.743135e-03
## X23           1.301852e-02
## X24           5.285317e-03
## X25          -2.589349e-03
## X26          -3.575799e-03
## X27           9.978716e-04
## X28           8.248864e-04
## X29          -5.741580e-03
## X30          -2.131275e-03
## X31           6.277583e-04
## X32          -1.204991e-02
## X33           3.205617e-03
## X34          -1.252101e-04
## X35          -3.358292e-04
## X36           3.710258e-04
## X37          -2.516631e-03
## X38           1.322375e-02
## X39           1.512749e-03
## X40          -4.962840e-04
## X41          -1.441793e-03
## X42          -1.950831e-03
## X43           3.645739e-04
## X44           5.227618e-03
## X45          -8.690230e-04
## X46          -6.262085e-03
## X47           2.068850e-03
## X48           3.331767e-03
## X49           3.730645e-03
## X50           1.739829e-03
```

```
bestlam=cv.out$lambda.min
test1 = model.matrix(~.,test)
rY1hat = predict(ridgemod, s= bestlam, newx = test1)
```

The lowest point on the curve indictates the optimal λ: the log value of λ that best minimized the error in cross validation. Ridge regression includes all the predictors in the final model and gives cv error = 0.000468. We can see that the cv error for ridge regression is approximately same as for linear regression but in linear model we have less number of predictors. So, in our case linear regresson performs much better than ridge regression. We now fit the Lasso model.

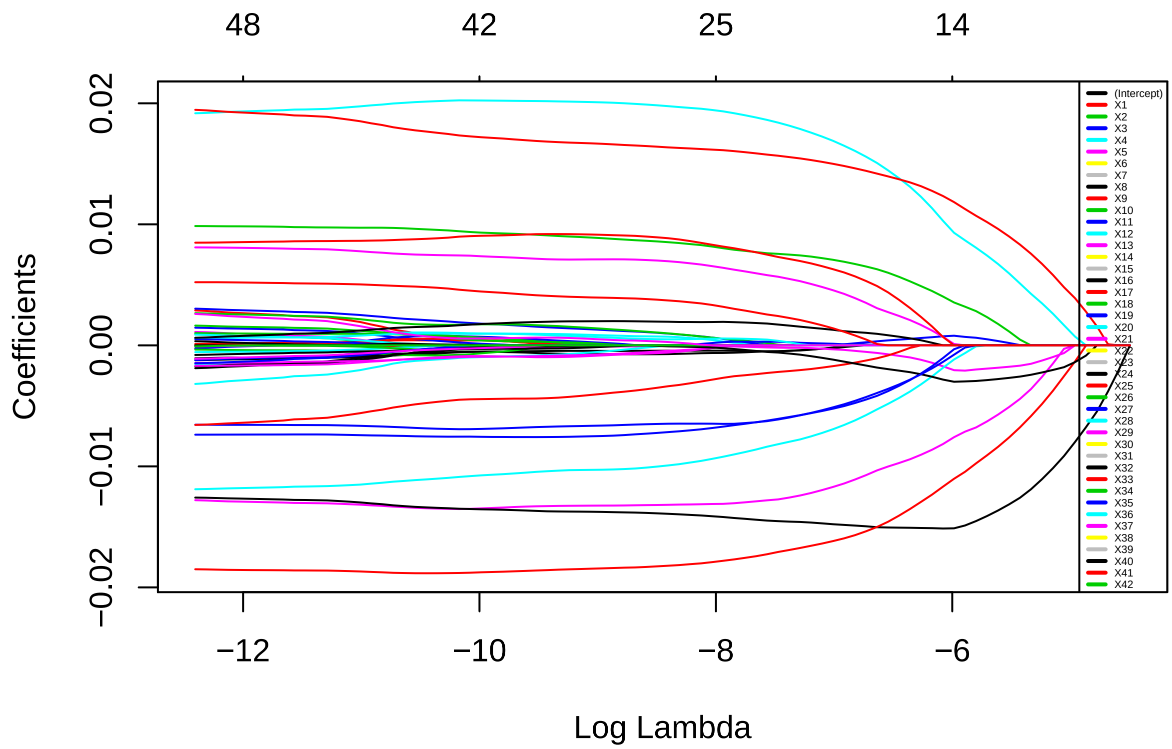
```
lassomod = glmnet(X,Y, alpha = 1)
lassomod
```

```
##
## Call:  glmnet(x = X, y = Y, alpha = 1)
##
##    Df    %Dev    Lambda
## 1   0 0.00000 0.0111500
## 2   1 0.02241 0.0101500
## 3   2 0.04203 0.0092530
## 4   2 0.07271 0.0084310
## 5   4 0.10420 0.0076820
## 6   5 0.14450 0.0069990
## 7   7 0.18570 0.0063780
## 8   7 0.22530 0.0058110
```

```
## 9   7 0.25820 0.0052950
## 10  7 0.28560 0.0048240
## 11  9 0.31010 0.0043960
## 12  9 0.33160 0.0040050
## 13  9 0.34940 0.0036500
## 14  9 0.36420 0.0033250
## 15 10 0.37680 0.0030300
## 16 11 0.38940 0.0027610
## 17 13 0.40250 0.0025150
## 18 14 0.41850 0.0022920
## 19 15 0.43320 0.0020880
## 20 15 0.44520 0.0019030
## 21 16 0.45560 0.0017340
## 22 16 0.46440 0.0015800
## 23 16 0.47170 0.0014390
## 24 18 0.47790 0.0013120
## 25 18 0.48340 0.0011950
## 26 19 0.48790 0.0010890
## 27 20 0.49180 0.0009922
## 28 19 0.49490 0.0009040
## 29 19 0.49760 0.0008237
## 30 19 0.49980 0.0007505
## 31 20 0.50170 0.0006839
## 32 20 0.50320 0.0006231
## 33 22 0.50460 0.0005677
## 34 22 0.50570 0.0005173
## 35 24 0.50680 0.0004714
## 36 24 0.50770 0.0004295
## 37 24 0.50840 0.0003913
## 38 26 0.50920 0.0003566
## 39 25 0.50980 0.0003249
## 40 26 0.51030 0.0002960
## 41 26 0.51070 0.0002697
## 42 25 0.51110 0.0002458
## 43 25 0.51140 0.0002239
## 44 26 0.51160 0.0002040
## 45 26 0.51190 0.0001859
## 46 28 0.51200 0.0001694
## 47 29 0.51220 0.0001543
## 48 29 0.51240 0.0001406
## 49 31 0.51250 0.0001281
## 50 32 0.51260 0.0001168
## 51 32 0.51280 0.0001064
## 52 33 0.51280 0.0000969
## 53 34 0.51290 0.0000883
## 54 35 0.51300 0.0000805
## 55 36 0.51310 0.0000733
## 56 36 0.51320 0.0000668
## 57 37 0.51320 0.0000609
## 58 37 0.51330 0.0000555
## 59 39 0.51330 0.0000505
## 60 39 0.51340 0.0000460
## 61 42 0.51340 0.0000420
## 62 42 0.51350 0.0000382
```

```
## 63 44 0.51350 0.0000348
## 64 44 0.51360 0.0000317
## 65 44 0.51360 0.0000289
## 66 45 0.51370 0.0000264
## 67 47 0.51370 0.0000240
## 68 47 0.51380 0.0000219
## 69 47 0.51380 0.0000199
## 70 47 0.51390 0.0000182
## 71 46 0.51390 0.0000166
## 72 46 0.51390 0.0000151
## 73 47 0.51400 0.0000137
## 74 47 0.51400 0.0000125
## 75 46 0.51400 0.0000114
## 76 46 0.51400 0.0000104
## 77 46 0.51400 0.0000095
## 78 47 0.51400 0.0000086
## 79 48 0.51400 0.0000079
## 80 47 0.51400 0.0000072
## 81 47 0.51400 0.0000065
## 82 48 0.51410 0.0000059
## 83 49 0.51410 0.0000054
## 84 49 0.51410 0.0000049
## 85 48 0.51410 0.0000045
## 86 49 0.51410 0.0000041
```

```
plot(lassomod, xvar = "lambda")
legend("topright", lwd = 2, col = 1:50, legend = colnames(X), cex = .3046)
```

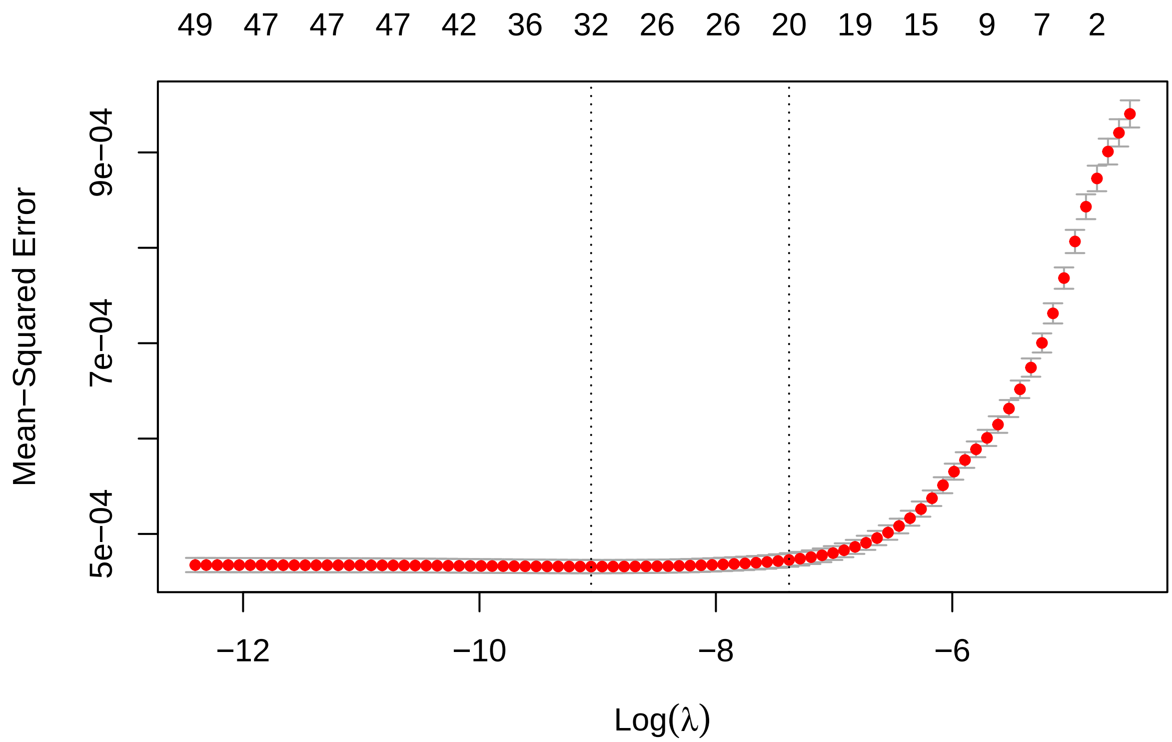


Similar to ridge regression, we plot the coefficients against different values of λ. Lasso regression forces some of the coefficients to exactly 0. This may be achieved by increasing the value of λ. From the plot above, the number of coefficients in our model decrease from 48 to 14, as $\log \lambda$ increases from -12 to -6. We select the optimum λ in the plot below.

```r
Lassocv.out=cv.glmnet(X,Y,alpha=1)
Lassocv.out
```

```
##
## Call:  cv.glmnet(x = X, y = Y, alpha = 1)
##
## Measure: Mean-Squared Error
##
##         Lambda   Measure        SE Nonzero
## min 0.0001168 0.0004658 6.993e-06      32
## 1se 0.0006231 0.0004727 7.106e-06      20
```

```r
plot(Lassocv.out)
```




```r
coef(Lassocv.out)
```

```
## 52 x 1 sparse Matrix of class "dgCMatrix"
##                         1
## (Intercept)  0.0019660365
## (Intercept)  .
## X1          -0.0004555138
## X2           .
## X3           .
## X4           .
## X5           .
## X6          -0.0125810437
## X7           .
## X8           .
## X9           0.0074970899
## X10          .
## X11          .
## X12          .
## X13         -0.0004475483
## X14          .
## X15          .
## X16          .
## X17          .
## X18         -0.0002024503
## X19         -0.0145653716
## X20          .
```

```
## X21          .
## X22         -0.0059678670
## X23          0.0181554967
## X24          0.0054910096
## X25          .
## X26         -0.0021256755
## X27          .
## X28          .
## X29         -0.0079718688
## X30          .
## X31          .
## X32         -0.0169130421
## X33          .
## X34          .
## X35          .
## X36          .
## X37         -0.0006775323
## X38          0.0155581656
## X39          .
## X40          0.0002593120
## X41          .
## X42          .
## X43          .
## X44          0.0071355315
## X45          .
## X46         -0.0059224577
## X47          0.0002485670
## X48          .
## X49          0.0016341317
## X50          0.0022649010
```

```
bestlam=Lassocv.out$lambda.min
test1 = model.matrix(~.,test)
Y1hat = predict(lassomod, s= bestlam, newx = test1)
```

In Lasso, our best model contains 29 predictors and cv error for this is 0.000466. But the model for 1SE from minimum λ contains 19 predictors and the cv error is 0.000473 which is approximate to that for best model. The main point of the 1SE rule, with which we agree, is to choose the simplest model whose accuracy is comparable with the best model. We introduce another regression method: Regression Trees.

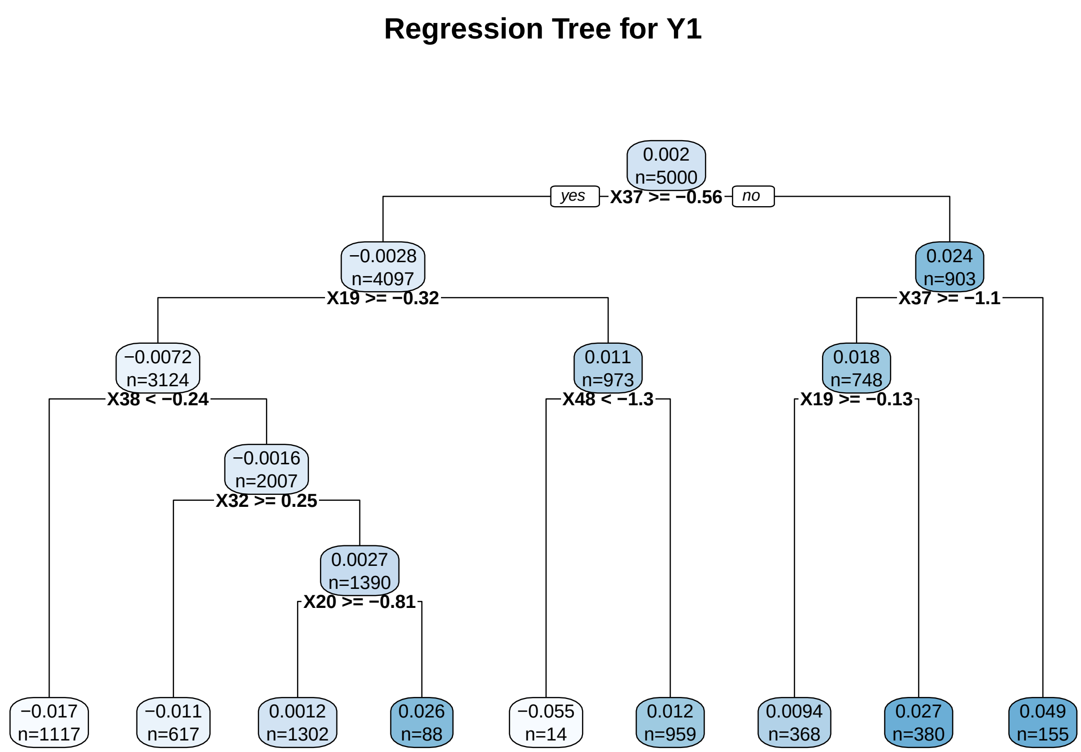
Another regression method to consider is Regression Trees. Regression trees are built through a process known as binary recurssive partitioning which is an iterative process that splits the data into partitions or branches and then continues splitting each partition into smaller groups, as the method moves up each branch. We apply this method to our data to predict $Y1$.

```
library(rpart)
fit1 <- rpart(Y1~., method="anova", data=train[,-c(2,3)], model = TRUE )
printcp(fit1)
```

```
##
## Regression tree:
## rpart(formula = Y1 ~ ., data = train[, -c(2, 3)], method = "anova",
##     model = TRUE)
##
## Variables actually used in tree construction:
## [1] X19 X20 X32 X37 X38 X48
```

```
##
## Root node error: 4.7046/5000 = 0.00094091
##
## n= 5000
##
##          CP nsplit rel error  xerror     xstd
## 1 0.110849      0   1.00000 1.00035 0.023448
## 2 0.054794      1   0.88915 0.89318 0.020544
## 3 0.037384      2   0.83436 0.83866 0.020118
## 4 0.025877      3   0.79697 0.80185 0.019506
## 5 0.018187      4   0.77110 0.78350 0.018568
## 6 0.013208      5   0.75291 0.77004 0.018348
## 7 0.012740      6   0.73970 0.76353 0.018312
## 8 0.010958      7   0.72696 0.75664 0.018197
## 9 0.010000      8   0.71600 0.74926 0.017893
```

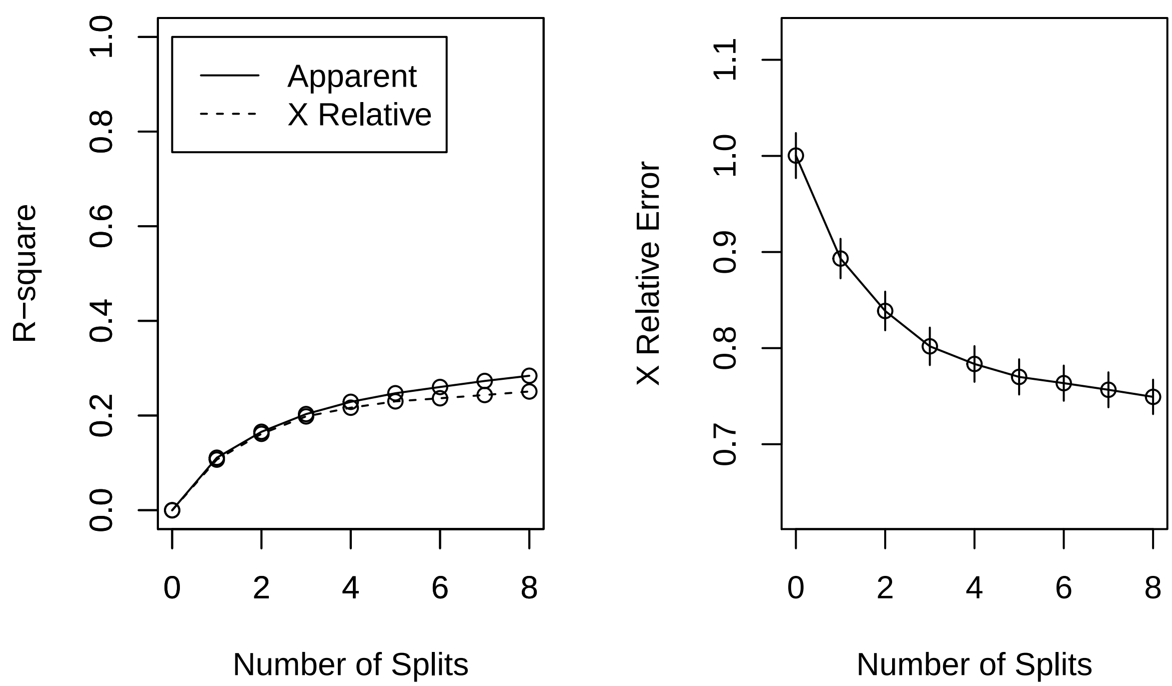
```r
rpart.plot(fit1, type = 2, extra = 1, cex = 0.6, main="Regression Tree for Y1")
```



The following variables are used in the contruction of the tree: $X19, X20, X32, X37, X38, X48$. The root node error is 0.00094091. The rel error of each iteration of the tree is the fraction of mislabeled elements in the iteration relative to the fraction of mislabeled elements in the root and nsplit is the number of splits in the tree. When rpart grows a tree it performs 10-fold cross validation on the data. We note that the xerror (cross validation error) gets better with each split. The tree diagram also shows the marked splits (for example: $X37 >= -0.561$). Also, at the terminating point of each branch, is the number of elements from the data file that fit at the end of that branch. There are 9 terminal nodes. To get a better picture of the change in xerror as the splits increase, we look at a new visualization.

```r
par(mfrow=c(1,2))
rsq.rpart(fit1)
```

```
##
## Regression tree:
## rpart(formula = Y1 ~ ., data = train[, -c(2, 3)], method = "anova",
##     model = TRUE)
##
## Variables actually used in tree construction:
## [1] X19 X20 X32 X37 X38 X48
##
## Root node error: 4.7046/5000 = 0.00094091
##
## n= 5000
##
##          CP nsplit rel error  xerror     xstd
## 1 0.110849      0   1.00000 1.00035 0.023448
## 2 0.054794      1   0.88915 0.89318 0.020544
## 3 0.037384      2   0.83436 0.83866 0.020118
## 4 0.025877      3   0.79697 0.80185 0.019506
## 5 0.018187      4   0.77110 0.78350 0.018568
## 6 0.013208      5   0.75291 0.77004 0.018348
## 7 0.012740      6   0.73970 0.76353 0.018312
## 8 0.010958      7   0.72696 0.75664 0.018197
## 9 0.010000      8   0.71600 0.74926 0.017893
```



The first chart shows how R-Squared improves as splits increase. The second chart shows how xerror decreases with each split. This shows that the model does not need pruning. We can also make predictions on the test data.

```
pred1 = predict(fit1, newdata = test, method = "anova")
```

Based on the results obtained above, there is not much difference between cv errors for the first 3 models but cv error for the tree model is 0.74926 which is bigger as compared to other models. So, the linear model is selected as the best model because it contains the least number of significant predictors with minimum cv error. We now test the accuracy of the selected model by making predictions using test data.

```
lmY1hat = predict(Linearmodel, newdata = test)
```

Model Fitting and Predictions For $Y2$

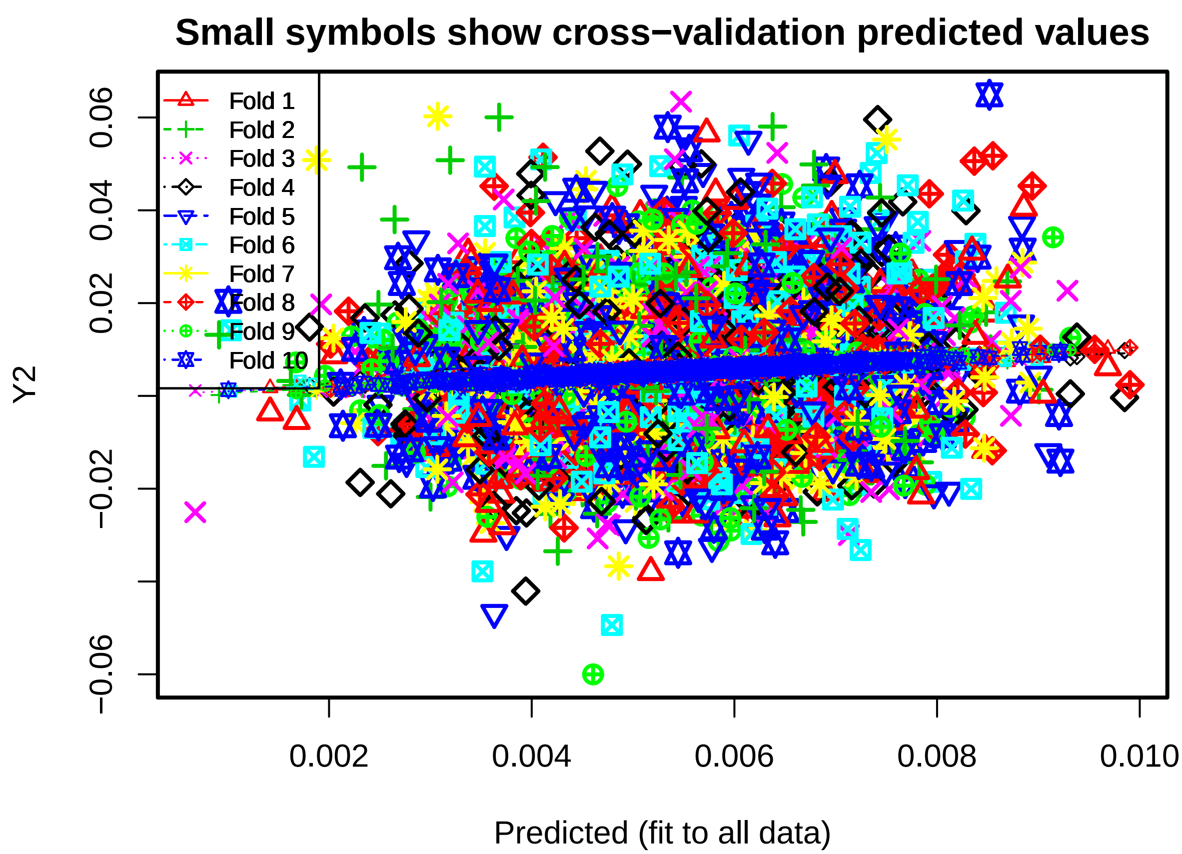
The same procedures for $Y1$ are undertaken for $Y2$. We fit the Linear, Ridge, Lasso and Regression tree models on the training data. We select the model with the least cv error and make some predictions based on the test data.

```
Linearmodel2 = lm(Y2~., data = train[,-c(1,3)])
summary(Linearmodel2)
```

```
##
## Call:
## lm(formula = Y2 ~ ., data = train[, -c(1, 3)])
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.064618 -0.009442 -0.001122  0.008292  0.057937
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  5.452e-03  2.012e-04  27.102   <2e-16 ***
## X1          -2.255e-03  4.050e-03  -0.557    0.578
## X2           3.621e-03  3.443e-03   1.052    0.293
## X3          -8.419e-03  1.105e-02  -0.762    0.446
## X4           1.268e-02  1.579e-02   0.803    0.422
## X5          -1.949e-03  9.214e-03  -0.212    0.832
## X6           1.589e-03  6.326e-03   0.251    0.802
## X7           5.570e-03  4.670e-03   1.193    0.233
## X8          -1.142e-02  1.080e-02  -1.058    0.290
## X9           1.273e-02  1.259e-02   1.011    0.312
## X10         -2.419e-04  3.312e-03  -0.073    0.942
## X11          1.529e-02  1.154e-02   1.325    0.185
## X12         -1.121e-03  5.706e-03  -0.196    0.844
## X13         -4.064e-05  2.723e-03  -0.015    0.988
## X14         -7.716e-03  7.173e-03  -1.076    0.282
## X15          2.461e-03  3.217e-03   0.765    0.444
## X16         -3.825e-03  6.137e-03  -0.623    0.533
## X17          4.307e-03  3.741e-03   1.151    0.250
## X18          6.961e-03  5.399e-03   1.289    0.197
## X19         -7.778e-03  6.002e-03  -1.296    0.195
## X20         -9.559e-04  3.544e-03  -0.270    0.787
## X21         -7.769e-03  7.380e-03  -1.053    0.293
```

```
## X22          1.014e-03  3.640e-03   0.278    0.781
## X23          1.075e-03  3.237e-03   0.332    0.740
## X24         -4.342e-03  4.838e-03  -0.897    0.370
## X25         -4.657e-05  6.372e-03  -0.007    0.994
## X26         -1.125e-03  2.350e-03  -0.479    0.632
## X27         -1.288e-02  1.119e-02  -1.151    0.250
## X28          2.911e-03  6.666e-03   0.437    0.662
## X29          5.058e-03  7.715e-03   0.656    0.512
## X30         -9.748e-03  6.855e-03  -1.422    0.155
## X31          4.851e-03  5.721e-03   0.848    0.397
## X32          3.249e-03  8.477e-03   0.383    0.702
## X33         -1.630e-02  1.518e-02  -1.074    0.283
## X34          5.277e-03  1.057e-02   0.499    0.618
## X35          2.808e-03  1.030e-02   0.273    0.785
## X36         -1.815e-03  7.272e-03  -0.250    0.803
## X37         -3.186e-03  4.553e-03  -0.700    0.484
## X38         -3.842e-03  4.368e-03  -0.880    0.379
## X39          2.079e-03  2.112e-03   0.984    0.325
## X40         -7.567e-03  5.907e-03  -1.281    0.200
## X41         -1.496e-03  3.290e-03  -0.455    0.649
## X42          5.251e-03  5.434e-03   0.966    0.334
## X43         -8.610e-03  1.483e-02  -0.580    0.562
## X44         -2.034e-04  6.755e-03  -0.030    0.976
## X45         -1.090e-02  1.277e-02  -0.854    0.393
## X46         -7.463e-03  1.050e-02  -0.711    0.477
## X47         -2.250e-03  2.690e-03  -0.836    0.403
## X48         -1.141e-02  1.420e-02  -0.804    0.421
## X49         -1.751e-02  1.562e-02  -1.121    0.262
## X50          9.083e-03  1.256e-02   0.723    0.470
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.01415 on 4949 degrees of freedom
## Multiple R-squared:  0.008093,   Adjusted R-squared:  -0.001928
## F-statistic: 0.8076 on 50 and 4949 DF,  p-value: 0.8315
```

```r
cv_error2 = cv.lm(train, Linearmodel2, m = 10, plotit = TRUE, printit = FALSE)
```




```
lmY2hat = predict(Linearmodel2, newdata = test)
```

Here, only the intercept is significant. In our final model there is no predictor variable. CV error is 0.000203. We do not consider this model because its adjusted R^2 is negative. The ridge regression model is fitted as follows:

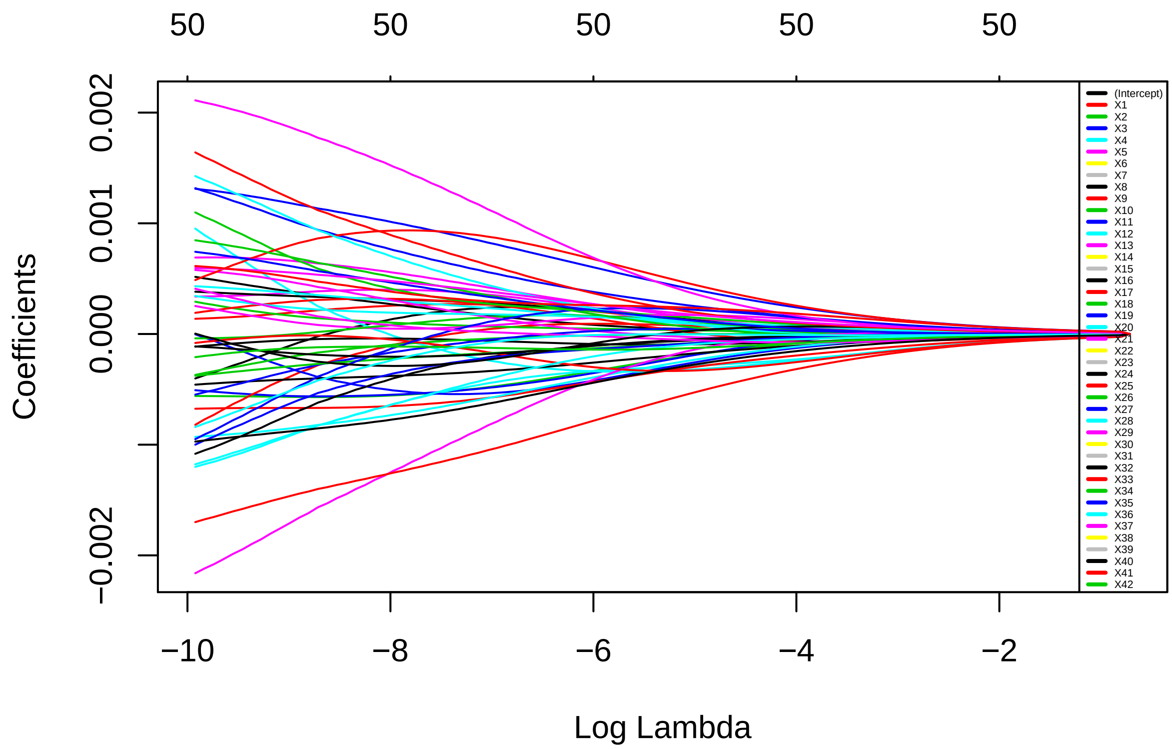
```
X_2 = model.matrix(Y2~., train[,-c(1,3)])
Y_2 = train$Y2
ridgemod2 = glmnet(X_2,Y_2, alpha = 0)
ridgemod2
```

```
##
## Call:  glmnet(x = X_2, y = Y_2, alpha = 0)
##
##      Df      %Dev  Lambda
## 1    50 0.0000000 0.49020
## 2    50 0.0005364 0.44670
## 3    50 0.0005828 0.40700
## 4    50 0.0006326 0.37080
## 5    50 0.0006860 0.33790
## 6    50 0.0007432 0.30790
## 7    50 0.0008044 0.28050
## 8    50 0.0008695 0.25560
## 9    50 0.0009389 0.23290
## 10   50 0.0010120 0.21220
## 11   50 0.0010900 0.19330
## 12   50 0.0011730 0.17620
```

```
## 13  50 0.0012590 0.16050
## 14  50 0.0013500 0.14630
## 15  50 0.0014450 0.13330
## 16  50 0.0015440 0.12140
## 17  50 0.0016470 0.11060
## 18  50 0.0017540 0.10080
## 19  50 0.0018640 0.09186
## 20  50 0.0019780 0.08370
## 21  50 0.0020950 0.07626
## 22  50 0.0022140 0.06949
## 23  50 0.0023350 0.06331
## 24  50 0.0024590 0.05769
## 25  50 0.0025840 0.05256
## 26  50 0.0027100 0.04789
## 27  50 0.0028370 0.04364
## 28  50 0.0029650 0.03976
## 29  50 0.0030930 0.03623
## 30  50 0.0032210 0.03301
## 31  50 0.0033480 0.03008
## 32  50 0.0034750 0.02741
## 33  50 0.0036020 0.02497
## 34  50 0.0037280 0.02275
## 35  50 0.0038520 0.02073
## 36  50 0.0039760 0.01889
## 37  50 0.0040990 0.01721
## 38  50 0.0042200 0.01568
## 39  50 0.0043400 0.01429
## 40  50 0.0044580 0.01302
## 41  50 0.0045750 0.01186
## 42  50 0.0046900 0.01081
## 43  50 0.0048040 0.00985
## 44  50 0.0049160 0.00898
## 45  50 0.0050260 0.00818
## 46  50 0.0051340 0.00745
## 47  50 0.0052400 0.00679
## 48  50 0.0053440 0.00619
## 49  50 0.0054480 0.00564
## 50  50 0.0055480 0.00514
## 51  50 0.0056450 0.00468
## 52  50 0.0057400 0.00426
## 53  50 0.0058330 0.00388
## 54  50 0.0059220 0.00354
## 55  50 0.0060090 0.00322
## 56  50 0.0060930 0.00294
## 57  50 0.0061730 0.00268
## 58  50 0.0062510 0.00244
## 59  50 0.0063260 0.00222
## 60  50 0.0063970 0.00203
## 61  50 0.0064660 0.00185
## 62  50 0.0065310 0.00168
## 63  50 0.0065930 0.00153
## 64  50 0.0066520 0.00140
## 65  50 0.0067080 0.00127
## 66  50 0.0067650 0.00116
```

```
## 67   50 0.0068110 0.00106
## 68   50 0.0068620 0.00096
## 69   50 0.0069030 0.00088
## 70   50 0.0069480 0.00080
## 71   50 0.0069840 0.00073
## 72   50 0.0070240 0.00066
## 73   50 0.0070550 0.00060
## 74   50 0.0070910 0.00055
## 75   50 0.0071190 0.00050
## 76   50 0.0071510 0.00046
## 77   50 0.0071750 0.00042
## 78   50 0.0072040 0.00038
## 79   50 0.0072260 0.00035
## 80   50 0.0072520 0.00032
## 81   50 0.0072720 0.00029
## 82   50 0.0072960 0.00026
## 83   50 0.0073140 0.00024
## 84   50 0.0073360 0.00022
## 85   50 0.0073530 0.00020
## 86   50 0.0073730 0.00018
## 87   50 0.0073880 0.00016
## 88   50 0.0074100 0.00015
## 89   50 0.0074270 0.00014
## 90   50 0.0074460 0.00012
## 91   50 0.0074640 0.00011
## 92   50 0.0074810 0.00010
## 93   50 0.0074990 0.00009
## 94   50 0.0075140 0.00009
## 95   50 0.0075310 0.00008
## 96   50 0.0075440 0.00007
## 97   50 0.0075590 0.00006
## 98   50 0.0075730 0.00006
## 99   50 0.0075840 0.00005
## 100  50 0.0075970 0.00005
```

```r
plot(ridgemod2, xvar = "lambda")
legend("topright", lwd = 2, col = 1:50, legend = colnames(X), cex = .3046)
```

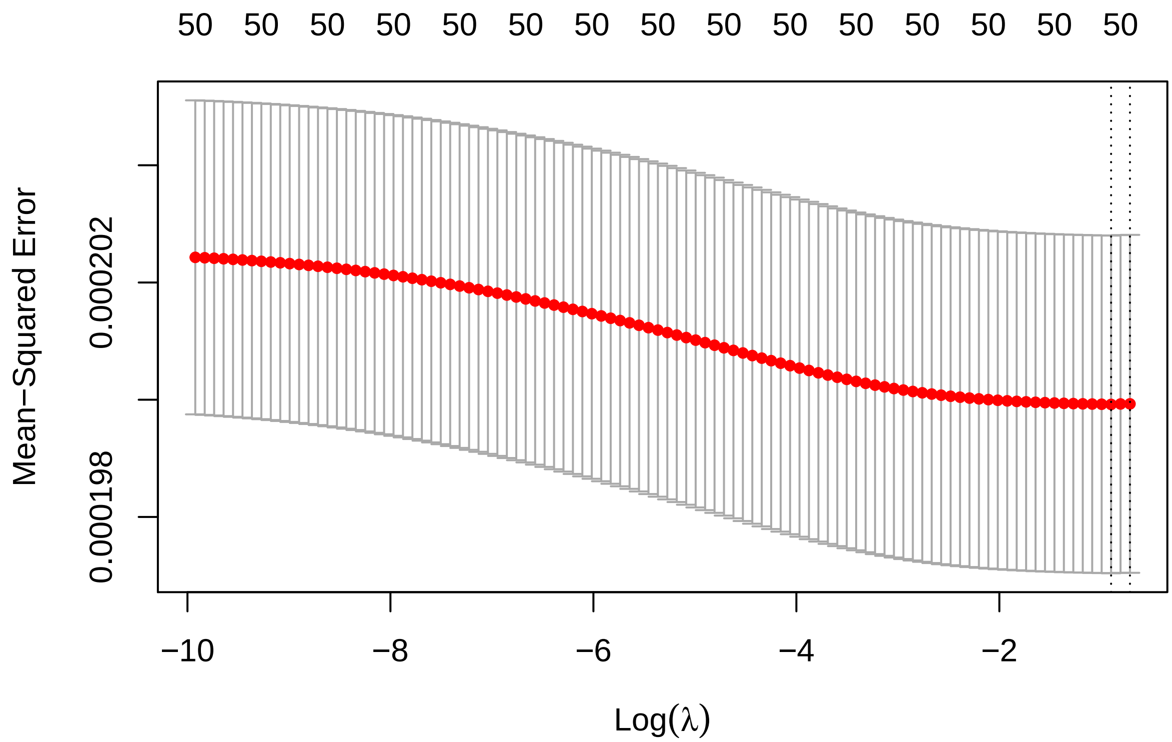


From the plot above, all the coefficients are shrunk closer to 0 as λ increases.

```
cv.out2 = cv.glmnet(X_2,Y_2,alpha=0)
cv.out2
```

```
##
## Call:  cv.glmnet(x = X_2, y = Y_2, alpha = 0)
##
## Measure: Mean-Squared Error
##
##     Lambda   Measure        SE Nonzero
## min 0.4070 0.0001999 2.880e-06      50
## 1se 0.4902 0.0001999 2.883e-06      50
```

```
plot(cv.out2)
```




```r
coef(cv.out2)
```

```
## 52 x 1 sparse Matrix of class "dgCMatrix"
##                         1
## (Intercept)  5.453196e-03
## (Intercept)  .
## X1          -2.750465e-41
## X2           6.303140e-41
## X3          -9.235356e-41
## X4           1.656460e-40
## X5          -9.388142e-40
## X6           1.497275e-40
## X7           2.259766e-40
## X8           2.260020e-40
## X9           3.820614e-41
## X10         -2.381968e-40
## X11         -8.157349e-40
## X12         -9.010140e-41
## X13         -5.985686e-41
## X14         -5.336996e-40
## X15          1.953637e-40
## X16          7.214191e-40
## X17          1.328406e-40
## X18          4.364926e-40
## X19         -3.405273e-40
## X20         -6.723673e-40
```

```
## X21          -8.722061e-41
## X22           4.501030e-40
## X23          -1.478619e-40
## X24          -1.280539e-40
## X25          -2.917218e-40
## X26           6.520556e-40
## X27          -5.261454e-41
## X28           3.502097e-41
## X29           2.169541e-40
## X30           1.122376e-40
## X31          -6.146801e-41
## X32          -8.769239e-40
## X33          -4.797648e-41
## X34           4.202255e-40
## X35          -2.801824e-40
## X36           2.277565e-40
## X37          -8.312747e-41
## X38          -8.927375e-40
## X39          -1.931765e-40
## X40           1.095731e-40
## X41          -1.796986e-40
## X42          -4.630243e-40
## X43           4.193562e-40
## X44           2.708140e-40
## X45          -2.086687e-40
## X46          -9.009831e-42
## X47          -1.297086e-40
## X48           3.799685e-40
## X49          -3.671072e-40
## X50           7.159940e-40
```

```
bestlam2=cv.out2$lambda.min
```

Ridge regression includes all the predictors in the final model and gives cv error = 0.0002. We can see that the cv error for ridge regression is almost same as linear regression. So, in this case ridge regression is much better than linear regression as linear regression doesn't contain any predictor. Now we will try to fit the Lasso regression.

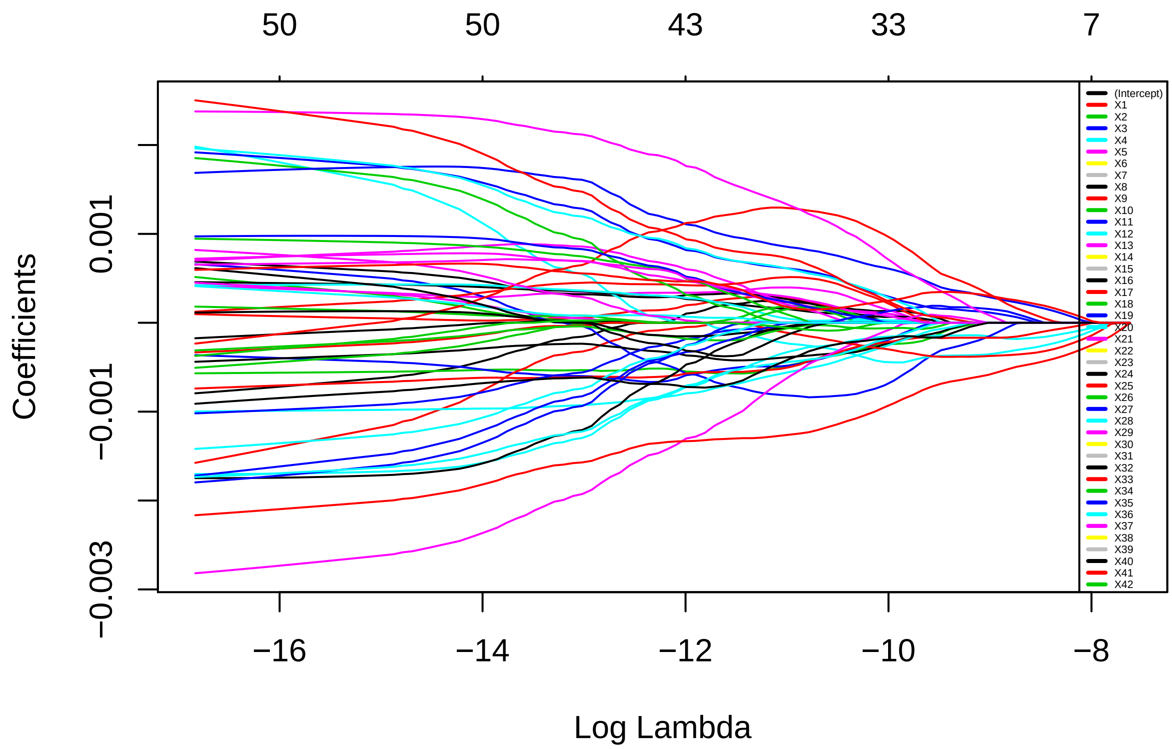
```
lassomod2 = glmnet(X_2,Y_2, alpha = 1)
lassomod2
```

```
##
## Call:  glmnet(x = X_2, y = Y_2, alpha = 1)
##
##      Df      %Dev    Lambda
## 1     0 0.0000000 4.902e-04
## 2     1 0.0002042 4.467e-04
## 3     3 0.0004007 4.070e-04
## 4     4 0.0007735 3.708e-04
## 5     6 0.0011320 3.379e-04
## 6     7 0.0015290 3.079e-04
## 7     7 0.0018760 2.805e-04
## 8     7 0.0021630 2.556e-04
## 9     8 0.0024130 2.329e-04
## 10    8 0.0026350 2.122e-04
## 11   10 0.0028590 1.933e-04
```

```
## 12   10 0.0030750 1.762e-04
## 13   11 0.0032590 1.605e-04
## 14   11 0.0034490 1.463e-04
## 15   12 0.0036450 1.333e-04
## 16   12 0.0038120 1.214e-04
## 17   14 0.0039680 1.106e-04
## 18   18 0.0041220 1.008e-04
## 19   18 0.0042950 9.186e-05
## 20   20 0.0044410 8.370e-05
## 21   23 0.0046160 7.626e-05
## 22   25 0.0048370 6.949e-05
## 23   29 0.0050580 6.331e-05
## 24   29 0.0052770 5.769e-05
## 25   31 0.0054560 5.256e-05
## 26   31 0.0056460 4.789e-05
## 27   33 0.0058030 4.364e-05
## 28   36 0.0059660 3.976e-05
## 29   36 0.0061130 3.623e-05
## 30   36 0.0062420 3.301e-05
## 31   37 0.0063540 3.008e-05
## 32   36 0.0064750 2.741e-05
## 33   38 0.0065680 2.497e-05
## 34   38 0.0066530 2.275e-05
## 35   40 0.0067220 2.073e-05
## 36   40 0.0068050 1.889e-05
## 37   41 0.0068630 1.721e-05
## 38   42 0.0069200 1.568e-05
## 39   43 0.0069700 1.429e-05
## 40   44 0.0070190 1.302e-05
## 41   44 0.0070640 1.186e-05
## 42   44 0.0071090 1.081e-05
## 43   43 0.0071400 9.850e-06
## 44   42 0.0071720 8.980e-06
## 45   43 0.0072030 8.180e-06
## 46   43 0.0072450 7.450e-06
## 47   42 0.0072750 6.790e-06
## 48   42 0.0072870 6.190e-06
## 49   43 0.0073240 5.640e-06
## 50   43 0.0073470 5.140e-06
## 51   43 0.0073680 4.680e-06
## 52   44 0.0073780 4.260e-06
## 53   45 0.0074020 3.880e-06
## 54   47 0.0074250 3.540e-06
## 55   47 0.0074580 3.220e-06
## 56   48 0.0074770 2.940e-06
## 57   50 0.0075030 2.680e-06
## 58   49 0.0075310 2.440e-06
## 59   48 0.0075440 2.220e-06
## 60   50 0.0075500 2.030e-06
## 61   49 0.0075600 1.850e-06
## 62   50 0.0075650 1.680e-06
## 63   50 0.0075780 1.530e-06
## 64   50 0.0075900 1.400e-06
## 65   49 0.0076050 1.270e-06
```

```
## 66  50 0.0076140 1.160e-06
## 67  49 0.0076250 1.060e-06
## 68  49 0.0076390 9.600e-07
## 69  50 0.0076480 8.800e-07
## 70  50 0.0076570 8.000e-07
## 71  50 0.0076660 7.300e-07
## 72  50 0.0076730 6.600e-07
## 73  50 0.0076780 6.000e-07
## 74  50 0.0076830 5.500e-07
## 75  50 0.0076880 5.000e-07
## 76  50 0.0076930 4.600e-07
## 77  50 0.0076970 4.200e-07
## 78  50 0.0076990 3.800e-07
## 79  50 0.0077040 3.500e-07
## 80  50 0.0077060 3.200e-07
## 81  50 0.0077080 2.900e-07
## 82  50 0.0077100 2.600e-07
## 83  50 0.0077120 2.400e-07
## 84  50 0.0077140 2.200e-07
## 85  50 0.0077160 2.000e-07
## 86  50 0.0077180 1.800e-07
## 87  50 0.0077200 1.600e-07
## 88  50 0.0077220 1.500e-07
## 89  50 0.0077240 1.400e-07
## 90  50 0.0077250 1.200e-07
## 91  50 0.0077270 1.100e-07
## 92  50 0.0077290 1.000e-07
## 93  50 0.0077310 9.000e-08
## 94  50 0.0077320 9.000e-08
## 95  50 0.0077340 8.000e-08
## 96  50 0.0077360 7.000e-08
## 97  50 0.0077370 6.000e-08
## 98  50 0.0077390 6.000e-08
## 99  50 0.0077400 5.000e-08
## 100 50 0.0077420 5.000e-08
```

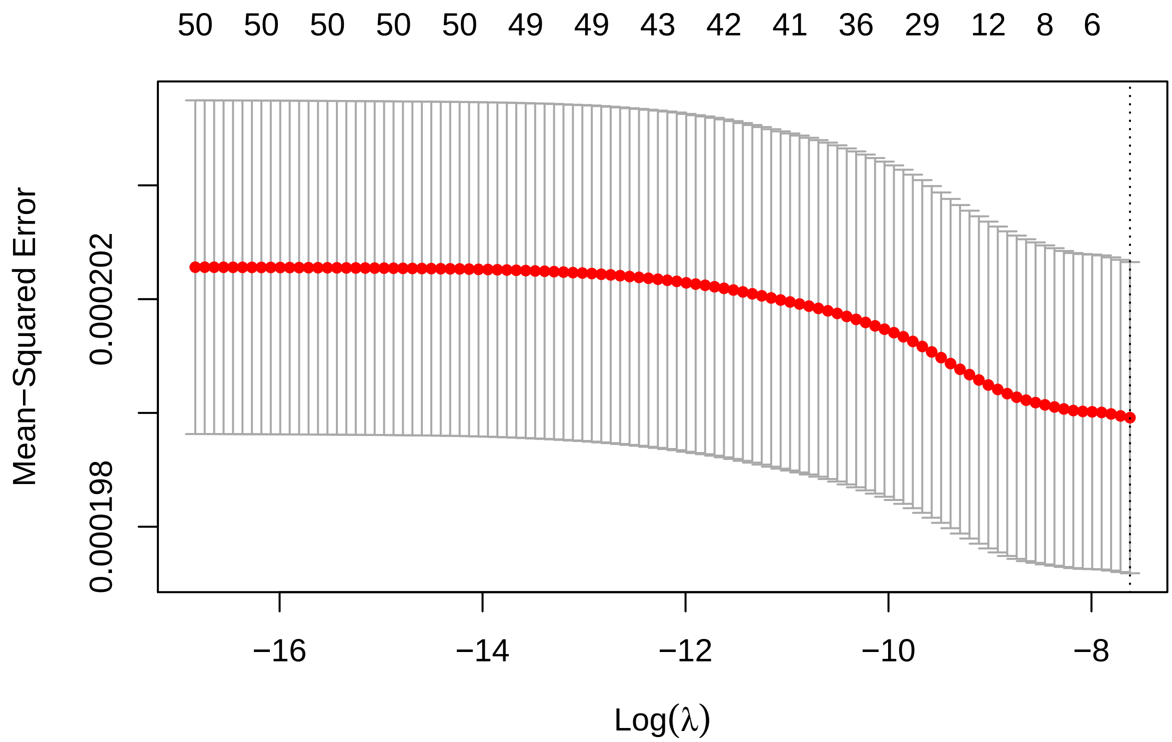
```r
plot(lassomod2, xvar = "lambda")
legend("topright", lwd = 2, col = 1:50, legend = colnames(X), cex = .3046)
```




```r
Lassocv.out2=cv.glmnet(X_2,Y_2,alpha=1)
Lassocv.out2
```

```
##
## Call:  cv.glmnet(x = X_2, y = Y_2, alpha = 1)
##
## Measure: Mean-Squared Error
##
##        Lambda   Measure        SE Nonzero
## min 0.0004902 0.0001999 2.734e-06       0
## 1se 0.0004902 0.0001999 2.734e-06       0
```

```r
plot(Lassocv.out2)
```




```r
coef(Lassocv.out2)
```

```
## 52 x 1 sparse Matrix of class "dgCMatrix"
##                       1
## (Intercept) 0.005453196
## (Intercept) .
## X1          .
## X2          .
## X3          .
## X4          .
## X5          .
## X6          .
## X7          .
## X8          .
## X9          .
## X10         .
## X11         .
## X12         .
## X13         .
## X14         .
## X15         .
## X16         .
## X17         .
## X18         .
## X19         .
## X20         .
```

```
## X21          .
## X22          .
## X23          .
## X24          .
## X25          .
## X26          .
## X27          .
## X28          .
## X29          .
## X30          .
## X31          .
## X32          .
## X33          .
## X34          .
## X35          .
## X36          .
## X37          .
## X38          .
## X39          .
## X40          .
## X41          .
## X42          .
## X43          .
## X44          .
## X45          .
## X46          .
## X47          .
## X48          .
## X49          .
## X50          .
```

```
bestlam2=Lassocv.out2$lambda.min
lY2hat = predict(lassomod2, s= bestlam2, newx = test1)
```

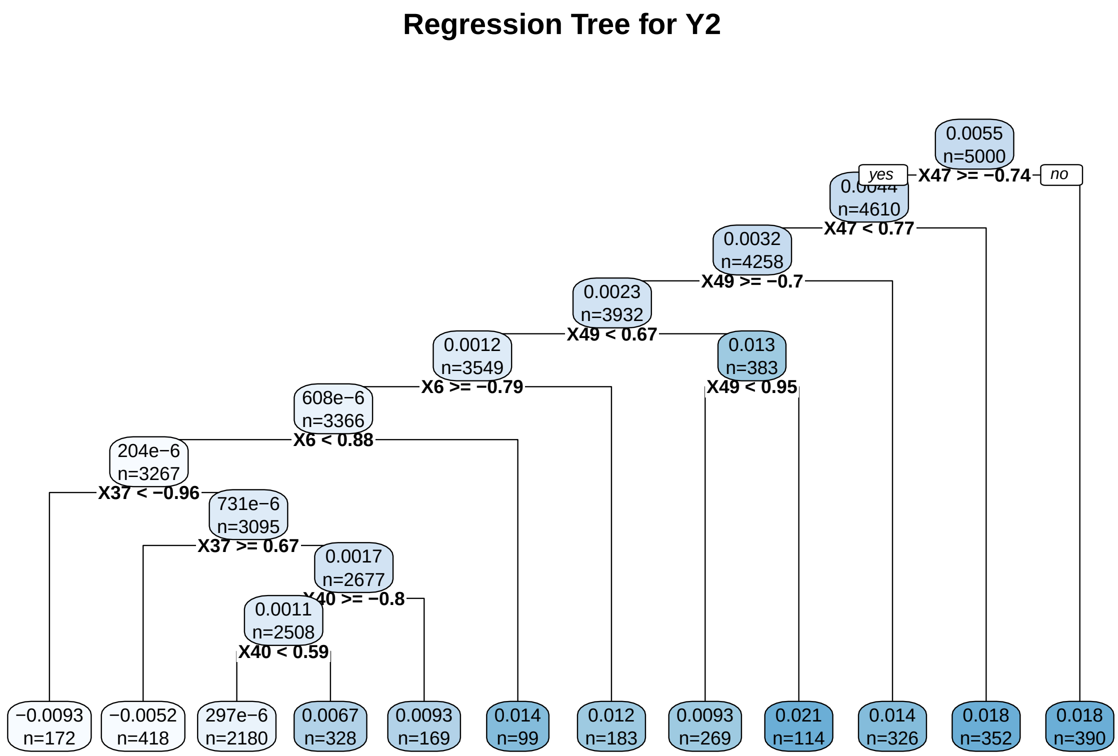
Lasso forces all coefficients except the intercept to 0. This means all the 3 methods perform poorly. Now we will try Regression Tree.

```
fit <- rpart(Y2~., method="anova", data=train[,-c(1,3)], model = TRUE)
printcp(fit)
```

```
##
## Regression tree:
## rpart(formula = Y2 ~ ., data = train[, -c(1, 3)], method = "anova",
##     model = TRUE)
##
## Variables actually used in tree construction:
## [1] X37 X40 X47 X49 X6
##
## Root node error: 0.99911/5000 = 0.00019982
##
## n= 5000
##
##         CP nsplit rel error  xerror     xstd
## 1 0.071687      0   1.00000 1.00049 0.023294
## 2 0.042787      2   0.85663 0.86605 0.020530
```

```
## 3 0.024444      4   0.77105 0.78872 0.018538
## 4 0.018105      5   0.74661 0.77049 0.018109
## 5 0.016645      6   0.72850 0.76466 0.018115
## 6 0.011203      8   0.69522 0.74038 0.017334
## 7 0.010096     10   0.67281 0.72231 0.017090
## 8 0.010000     11   0.66271 0.71356 0.016948
```

```
rpart.plot(fit, type = 2, extra = 1, cex = 0.6, main="Regression Tree for Y2")
```

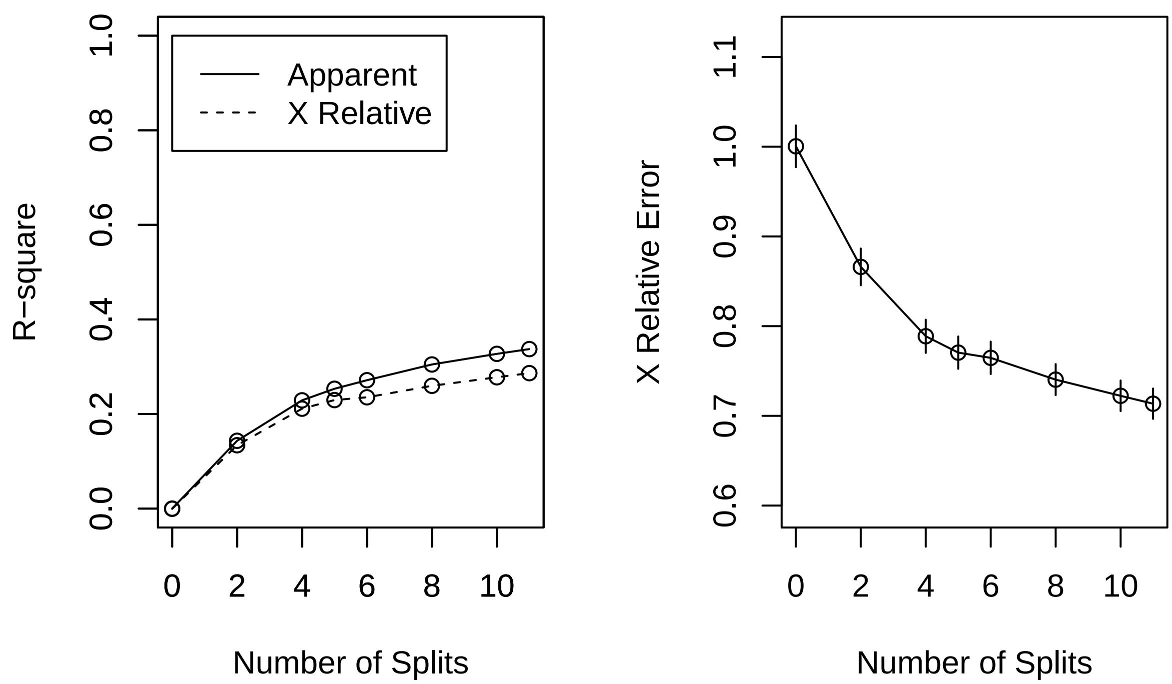


The following variables are used in the contruction of the tree: $X6, X37, X40, X47, X49$. The root node error is 0.00019982. We note that the xerror (cross validation error) gets better with each split. The tree diagram also shows the marked splits (for example: $X47 >= -0.74$). Also, at the terminating point of each branch, is the number of elements from the data file that fit at the end of that branch. There are 12 terminal nodes. To get a better picture of the change in xerror as the splits increase, we look at a new visualization just like in case of Y1.

```
par(mfrow=c(1,2))
rsq.rpart(fit)
```

```
##
## Regression tree:
## rpart(formula = Y2 ~ ., data = train[, -c(1, 3)], method = "anova",
##     model = TRUE)
##
## Variables actually used in tree construction:
## [1] X37 X40 X47 X49 X6
##
## Root node error: 0.99911/5000 = 0.00019982
```

```
##
## n= 5000
##
##          CP nsplit rel error  xerror     xstd
## 1 0.071687      0   1.00000 1.00049 0.023294
## 2 0.042787      2   0.85663 0.86605 0.020530
## 3 0.024444      4   0.77105 0.78872 0.018538
## 4 0.018105      5   0.74661 0.77049 0.018109
## 5 0.016645      6   0.72850 0.76466 0.018115
## 6 0.011203      8   0.69522 0.74038 0.017334
## 7 0.010096     10   0.67281 0.72231 0.017090
## 8 0.010000     11   0.66271 0.71356 0.016948
```



From the first chart it is clear that R-square is increasing with the increasing number of splits. The second chart shows how xerror decreases with each split. This shows that the model does not need pruning. We can also make predictions on the test data.

```
pred2 = predict(fit, newdata = test, method = "anova")
```

Since, linear and lasso models contain no predictor so we are rejecting these two. Moreover, from ridge model and tree model, cv error is minimum for ridge. So, in case of Y2 ridge model is the best one. We now test the accuracy of the selected model by making predictions using test data.

```
rY2hat = predict(ridgemod2, s= bestlam2, newx = test1)
```

Model Fitting and Predictions For $Y3$

In this section, we will study 4 different algorithms and determine the best in terms of predicting $Y3$. We will consider:

- Logistic Regression
- Linear Discriminant Analysis (LDA)
- Quadratic Discriminant Analysis (QDA)
- Support Vector Machines (SVM)

We take a look at the structure of $Y3$.

```
str(train$Y3)
```

```
##  int [1:5000] 1 0 1 1 0 0 0 0 1 0 ...
```

We can see Y3 is an integer. But for classification Y3 should be a factor. So firstly change Y3 into factor and generate new data with Y3 as factor.

```
Y3 = factor(train$Y3)
train1 = cbind(Y3, train[,-3])
```

Firstly, we fit a Logistic regression for $Y3$.

```
glm.fit = glm(Y3~., data = train1[,-c(2,3)], family = binomial)
summary(glm.fit)
```

```
##
## Call:
## glm(formula = Y3 ~ ., family = binomial, data = train1[, -c(2,
##     3)])
##
## Deviance Residuals:
##        Min          1Q      Median          3Q         Max
## -1.173e-03  -2.000e-08   2.000e-08   2.000e-08   1.233e-03
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)     2.009    688.929   0.003    0.998
## X1            176.459  18052.639   0.010    0.992
## X2            489.510  14743.482   0.033    0.974
## X3            441.968  57568.147   0.008    0.994
## X4            397.917  85157.802   0.005    0.996
## X5             78.331  46757.020   0.002    0.999
## X6           -518.739  21333.794  -0.024    0.981
## X7           -152.806  11143.711  -0.014    0.989
## X8           -163.176  31092.432  -0.005    0.996
## X9            270.237  59009.003   0.005    0.996
## X10           557.756  13169.575   0.042    0.966
## X11          -597.218  52200.367  -0.011    0.991
## X12          -576.876  26417.574  -0.022    0.983
## X13           336.080  15190.618   0.022    0.982
## X14          -776.826  23376.408  -0.033    0.973
## X15          -159.099  11480.695  -0.014    0.989
## X16          -208.134  19182.747  -0.011    0.991
## X17          -122.535  17979.879  -0.007    0.995
```

```
## X18           -87.562  21089.042  -0.004    0.997
## X19          -489.401  22031.939  -0.022    0.982
## X20           638.571  12065.344   0.053    0.958
## X21          -151.496  36091.269  -0.004    0.997
## X22          -275.968  11934.701  -0.023    0.982
## X23           588.911  16330.186   0.036    0.971
## X24           262.549  19449.964   0.013    0.989
## X25            82.025  34383.990   0.002    0.998
## X26           -72.470  11328.085  -0.006    0.995
## X27           213.038  47434.209   0.004    0.996
## X28            51.995  33588.594   0.002    0.999
## X29          -213.284  27811.075  -0.008    0.994
## X30           -31.528  32512.857  -0.001    0.999
## X31           436.725  19995.610   0.022    0.983
## X32          -701.587  46041.914  -0.015    0.988
## X33          -453.017  73590.640  -0.006    0.995
## X34           206.741  49882.233   0.004    0.997
## X35            38.191  40472.531   0.001    0.999
## X36          -101.150  29928.093  -0.003    0.997
## X37           409.444  15135.158   0.027    0.978
## X38           485.276  17011.843   0.029    0.977
## X39           200.901   6422.385   0.031    0.975
## X40          -427.170  27836.774  -0.015    0.988
## X41           453.577  16970.563   0.027    0.979
## X42          -141.419  25696.046  -0.006    0.996
## X43           610.287  78527.999   0.008    0.994
## X44           389.474  40923.997   0.010    0.992
## X45          -278.299  58395.272  -0.005    0.996
## X46          -193.820  51360.887  -0.004    0.997
## X47          -553.099  12530.338  -0.044    0.965
## X48          -547.996  75106.250  -0.007    0.994
## X49          -599.465  72752.421  -0.008    0.993
## X50           111.347  71894.377   0.002    0.999
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 6.9292e+03  on 4999  degrees of freedom
## Residual deviance: 2.8469e-05  on 4949  degrees of freedom
## AIC: 102
##
## Number of Fisher Scoring iterations: 25
```

From the output, it is clear that in our case the logistic regression algorithm does not converge. We reject logistic classifier and fit a Linear discriminant analysis (LDA) model. LDA is mainly used to classify multiclass classification problems. To make a prediction the model estimates the input data matching probability to each class by using Bayes theorem.

```r
ldafit = lda(Y3~., data = train1[,-c(2,3)])
ldafit
```

```
## Call:
## lda(Y3 ~ ., data = train1[, -c(2, 3)])
##
## Prior probabilities of groups:
##      0      1
```

```
## 0.4894 0.5106
##
## Group means:
##            X1         X2         X3          X4         X5          X6
## 0 -0.04981217 -0.1688893 -0.1700950 -0.07001943  0.1236802  0.02112887
## 1  0.03460252  0.1666549  0.1588247  0.06807096 -0.1099812 -0.02627110
##            X7          X8          X9         X10        X11         X12
## 0 -0.03118018 -0.03953663  0.05336750 -0.07632446  0.2398369  0.09922647
## 1  0.02524338  0.05577184 -0.03782301  0.07163817 -0.2380200 -0.08559733
##           X13        X14          X15         X16          X17        X18
## 0 -0.04534577  0.1074277 -0.001056085 -0.07245301  0.006218594  0.1204319
## 1  0.05278325 -0.1129902  0.017952232  0.05617460 -0.009754120 -0.1161611
##           X19        X20         X21          X22         X23         X24
## 0  0.06232570 -0.1255947 -0.02905336 -0.008856194 -0.08018230 -0.03833871
## 1 -0.05621573  0.1371824  0.04326432  0.004929992  0.06016817  0.03525554
##           X25         X26         X27         X28          X29        X30
## 0 0.021189332 -0.05422772 -0.02175910  0.06651374 -0.004884464  0.1208644
## 1 0.000629122  0.06613591  0.01210241 -0.06459496 -0.005583696 -0.1256267
##          X31          X32          X33         X34         X35         X36
## 0 -0.1422584 -0.005722139 -0.004676796 0.003824521 -0.03088912 -0.02627846
## 1  0.1370168  0.010229058  0.028739345 0.002779524  0.03734637  0.03617459
##          X37          X38        X39         X40        X41         X42
## 0 -0.1442889  0.015527103 -0.1290985  0.09104531 -0.1188501  0.09633521
## 1  0.1195683 -0.003418597  0.1300152 -0.09459839  0.1211744 -0.07905206
##           X43         X44        X45         X46        X47        X48
## 0 -0.06746378 -0.03726907  0.1491458  0.07704186  0.1504928  0.1153167
## 1  0.08775206  0.04746618 -0.1296251 -0.07771314 -0.1419775 -0.1444527
##           X49         X50
## 0  0.04501482  0.06465456
## 1 -0.05283969 -0.06349392
##
## Coefficients of linear discriminants:
##             LD1
## X1   0.288283275
## X2   0.835230469
## X3  -0.413536224
## X4   1.750713761
## X5  -0.413130708
## X6  -0.545695429
## X7   0.107542519
## X8  -0.696674119
## X9   1.278096064
## X10  0.670865663
## X11  0.536905139
## X12 -0.611952066
## X13  0.234167291
## X14 -1.499170501
## X15  0.079888821
## X16 -0.309563334
## X17  0.276645229
## X18  0.495741650
## X19 -1.062247063
## X20  0.873926005
## X21 -0.885843096
```

```
## X22 -0.288307356
## X23  0.762716060
## X24 -0.149461437
## X25  0.185257427
## X26 -0.264548413
## X27 -0.748619619
## X28  0.395808290
## X29  0.148024803
## X30 -0.880615309
## X31  0.882969475
## X32 -0.209398624
## X33 -1.904131320
## X34  0.937986594
## X35  0.012166763
## X36 -0.427020220
## X37  0.100793816
## X38  0.504570263
## X39  0.429311034
## X40 -1.104790242
## X41  0.478308532
## X42  0.249554238
## X43 -0.376602731
## X44  0.002624277
## X45 -1.179238335
## X46 -1.071048004
## X47 -0.648067318
## X48 -1.609511219
## X49 -2.177049448
## X50  1.135775273
```

LDA returns the prior probability of each class. These probabilities are the ones that already exist in the training data which are $0.4894, 0.5106$ for class 0 *and* 1 respectively. The second thing that we can see are the Group means, which are the average of each predictor within each class. The last one are the coefficients of linear discriminants. We will now calculate the cv error for 10 folds.

```
K = 10
folds = cut(seq(1,nrow(train1)), breaks = K, labels = FALSE)
set.seed(1)
cv.lda = sapply(1:K, FUN = function(i){
  testid = which(folds == i, arr.ind = TRUE)
  Test1 = train1[testid,]
  Train1 = train1[-testid,]
  lda_fit = lda(Y3~., data = Train1[,-c(2,3)])
  lda.pred = predict(lda_fit, Test1[,-c(2,3)])
  cv.est.lda = mean(lda.pred$class!=Test1$Y3)
  return(cv.est.lda)
})
cv.lda
```

```
##  [1] 0.022 0.028 0.036 0.018 0.048 0.018 0.026 0.030 0.028 0.036
```

```
mean(cv.lda)
```

```
## [1] 0.029
```

CV error is 0.029. Further make the predictions based on test data.

```r
preds = predict(ldafit, test)
```

We will do the same for another method known as Quadratic discriminant analysis(QDA). QDA allows for each class in the dependent variable to have its own covariance rather than a shared covariance as in LDA.

```r
qdafit = qda(Y3~., data = train1[,-c(2,3)])
qdafit
```

```
## Call:
## qda(Y3 ~ ., data = train1[, -c(2, 3)])
##
## Prior probabilities of groups:
##      0      1
## 0.4894 0.5106
##
## Group means:
##            X1         X2         X3          X4         X5          X6
## 0 -0.04981217 -0.1688893 -0.1700950 -0.07001943  0.1236802  0.02112887
## 1  0.03460252  0.1666549  0.1588247  0.06807096 -0.1099812 -0.02627110
##            X7          X8          X9         X10        X11         X12
## 0 -0.03118018 -0.03953663  0.05336750 -0.07632446  0.2398369  0.09922647
## 1  0.02524338  0.05577184 -0.03782301  0.07163817 -0.2380200 -0.08559733
##           X13        X14          X15         X16          X17        X18
## 0 -0.04534577  0.1074277 -0.001056085 -0.07245301  0.006218594  0.1204319
## 1  0.05278325 -0.1129902  0.017952232  0.05617460 -0.009754120 -0.1161611
##           X19        X20         X21          X22         X23         X24
## 0  0.06232570 -0.1255947 -0.02905336 -0.008856194 -0.08018230 -0.03833871
## 1 -0.05621573  0.1371824  0.04326432  0.004929992  0.06016817  0.03525554
##           X25         X26         X27         X28          X29        X30
## 0 0.021189332 -0.05422772 -0.02175910  0.06651374 -0.004884464  0.1208644
## 1 0.000629122  0.06613591  0.01210241 -0.06459496 -0.005583696 -0.1256267
##          X31          X32          X33         X34         X35         X36
## 0 -0.1422584 -0.005722139 -0.004676796 0.003824521 -0.03088912 -0.02627846
## 1  0.1370168  0.010229058  0.028739345 0.002779524  0.03734637  0.03617459
##          X37          X38        X39         X40        X41         X42
## 0 -0.1442889  0.015527103 -0.1290985  0.09104531 -0.1188501  0.09633521
## 1  0.1195683 -0.003418597  0.1300152 -0.09459839  0.1211744 -0.07905206
##           X43         X44        X45         X46        X47        X48
## 0 -0.06746378 -0.03726907  0.1491458  0.07704186  0.1504928  0.1153167
## 1  0.08775206  0.04746618 -0.1296251 -0.07771314 -0.1419775 -0.1444527
##           X49         X50
## 0  0.04501482  0.06465456
## 1 -0.05283969 -0.06349392
```

```r
K = 10
folds = cut(seq(1,nrow(train)), breaks = K, labels = FALSE)
set.seed(1)
cv.qda = sapply(1:K, FUN = function(i){
  testid = which(folds == i, arr.ind = TRUE)
  Test1 = train1[testid,]
  Train1 = train1[-testid,]
  qda_fit = qda(Y3~., data = Train1[,-c(2,3)])
  qda.pred = predict(qda_fit, Test1[,-c(2,3)])
  cv.est.qda = mean(qda.pred$class!=Test1$Y3)
  return(cv.est.qda)
```

```
})
cv.qda
```

```
##  [1] 0.084 0.062 0.078 0.050 0.100 0.078 0.088 0.068 0.068 0.094
```

```
mean(cv.qda)
```

```
## [1] 0.077
```

For QDA, cv error is 0.077. We will calculate predicted classes for test data.

```
preds = predict(qdafit, test)
```

We will now build linear SVM classifier. SVM classifiers are well-known for good prediction capabilities. A k-fold cross validation with 10 folds is performed to assess the quality of the classifier.

```
set.seed(1)
tune_out = tune(svm,
Y3~.,
data = train1[,-c(2,3)],
kernel = "linear",
ranges = list(cost = c(0.001, 0.01, 0.1, 1,5,10,100)))
summary(tune_out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##    10
##
## - best performance: 0.0066
##
## - Detailed performance results:
##    cost  error  dispersion
## 1 1e-03 0.0600 0.008692270
## 2 1e-02 0.0302 0.008508819
## 3 1e-01 0.0146 0.006736303
## 4 1e+00 0.0108 0.004442222
## 5 5e+00 0.0070 0.003299832
## 6 1e+01 0.0066 0.003893014
## 7 1e+02 0.0068 0.003910101
```

For the SVM with a linear kernel, the cost paramter, $c = 10$ produces the SVM with the smallest cross validation error (0.0066). We can then select the best model and use it for predictions on the test set.

```
best_model = tune_out$best.model
best_model
```

```
##
## Call:
## best.tune(method = svm, train.x = Y3 ~ ., data = train1[, -c(2, 3)],
##     ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100)), kernel = "linear")
##
##
## Parameters:
```

```
##     SVM-Type:  C-classification
##   SVM-Kernel:  linear
##         cost:  10
##
## Number of Support Vectors:  136
```

From the above outputs, cv error is minimum for linear SVM classifier. Hence this is best classifier for Y3. We can then check the accuracy for our selected classifier by using predicted classes from test data.

```r
svm.preds = predict(best_model, newdata = test)
```

Conclusion

We were tasked to predict 2 continuous variables and a categorical variable. Various algorithms were implemented for each variable. The best method for predicting each variable was selected based on their 10-fold cross validation (cv) error. The linear regression model had the least cv error for predicting $Y1$, Ridge regression for $Y2$ and Linear Support Vector Machines (SVM) for $Y3$.

Predictions for these response variables were made on the test data and were stored in a csv file. Several other prediction and classification techniques could have been used for the task. It is recommended to the reader to investigate further with techniques such as KNN, Neural Networks and Random Forests.