# Machine Learning - Lab 2

Hussnain Khalid (huskh803), Jaskirat S Marar (jasma356), Daniel Persson (danpe586)

12/03/2021

## Statement of Contribution

For solving this lab, the group decided to split the responsibility equally by assigning 1 question to each member. The split by mutual consensus was as follows:

- Assignment 1: Solution and report by Daniel Persson
- Assignment 2: Solution and report by Hussnain Khalid
- Assignment 3: Solution and report by Jaskirat Marar

We were able to communicate with each other effectively and responsibly. All the group members were forthcoming in discussing issues being faced while solving the problems. We were able to each present our solution to the others well before the deadline and were able to conclude on the structure and content of the final report.

*"We acknowledge that each member has contributed fairly and equally in solving this lab."*

*By undersigned:*

*Hussnain Khalid*

*Jaskirat Marar*

*Daniel Persson*

# 1. Assignment 1 - Explicit regularization

### 1.1 Linear Regression

The probabilistic model is:

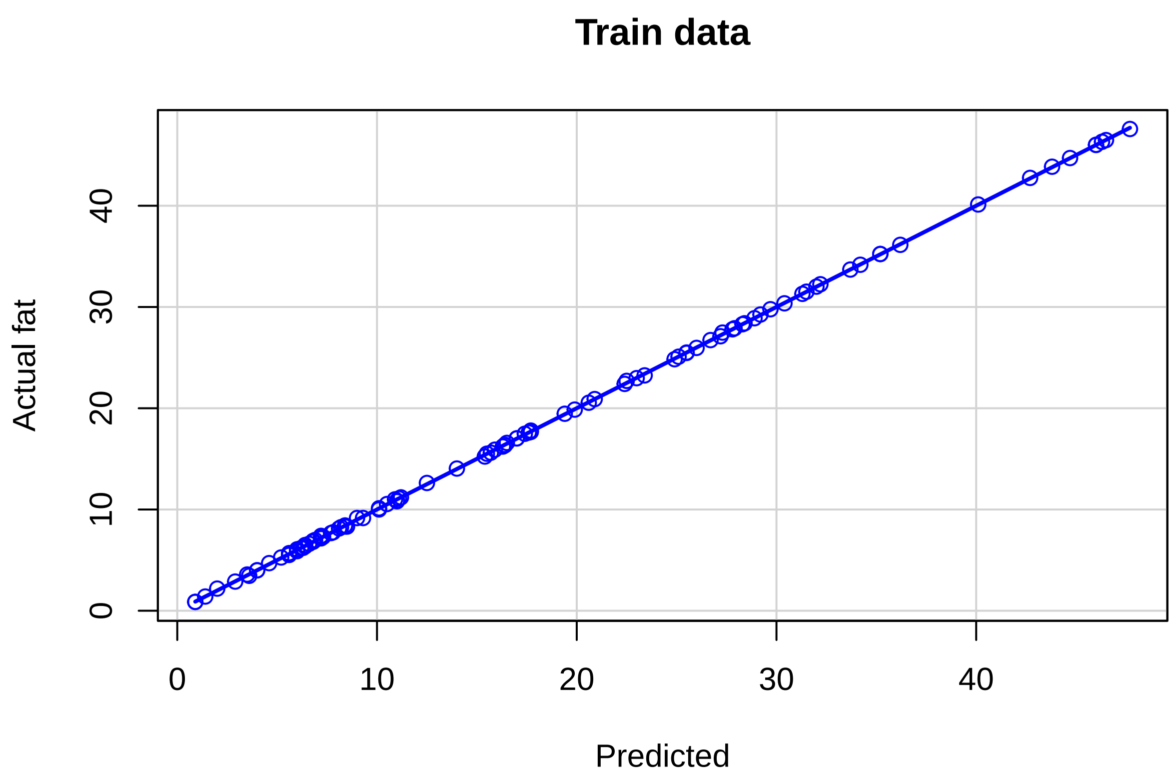
$$\boldsymbol{Y} \sim N(\boldsymbol{\beta}^T \boldsymbol{X}, \sigma^2)$$

The summary of the fit model is:

```
##
## Call:
## lm(formula = train$Fat ~ ., data = channels)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.201500 -0.041315 -0.001041  0.037636  0.187860
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.815e+01  5.488e+00  -3.306  0.01628 *
## Channel1     2.653e+04  1.126e+04   2.357  0.05649 .
## Channel2    -5.871e+04  3.493e+04  -1.681  0.14385
## Channel3     1.154e+05  7.373e+04   1.565  0.16852
## Channel4    -2.432e+05  1.175e+05  -2.070  0.08387 .
## Channel5     3.026e+05  1.193e+05   2.536  0.04430 *
## Channel6    -2.365e+05  8.160e+04  -2.898  0.02741 *
## Channel7     1.090e+05  3.169e+04   3.440  0.01380 *
## Channel8    -6.054e+04  1.508e+04  -4.015  0.00700 **
## Channel9     7.871e+04  2.160e+04   3.643  0.01079 *
## Channel10   -1.730e+04  1.640e+04  -1.055  0.33215
## Channel11    9.562e+04  3.529e+04   2.710  0.03512 *
## Channel12   -2.114e+05  6.198e+04  -3.410  0.01431 *
## Channel13    9.725e+04  4.424e+04   2.198  0.07026 .
## Channel14    5.296e+04  4.666e+04   1.135  0.29968
## Channel15   -7.855e+04  5.245e+04  -1.498  0.18491
## Channel16   -8.209e+03  1.893e+04  -0.434  0.67969
## Channel17    3.769e+04  1.987e+04   1.897  0.10666
## Channel18    3.306e+04  7.934e+03   4.167  0.00590 **
## Channel19   -8.405e+04  1.929e+04  -4.358  0.00478 **
## Channel20    1.510e+05  3.361e+04   4.492  0.00414 **
## Channel21   -2.069e+05  4.256e+04  -4.862  0.00282 **
## Channel22    1.348e+05  3.824e+04   3.526  0.01243 *
## Channel23   -4.094e+04  3.546e+04  -1.154  0.29222
## Channel24    2.023e+04  2.761e+04   0.733  0.49134
## Channel25    3.269e+03  1.071e+04   0.305  0.77045
## Channel26   -1.297e+04  7.636e+03  -1.699  0.14028
## Channel27    4.131e+03  1.422e+04   0.291  0.78120
## Channel28   -4.548e+03  2.988e+04  -0.152  0.88402
## Channel29    1.089e+04  1.768e+04   0.616  0.56072
## Channel30   -7.985e+04  2.653e+04  -3.010  0.02371 *
## Channel31    1.756e+05  5.279e+04   3.326  0.01589 *
## Channel32   -1.107e+05  2.904e+04  -3.813  0.00883 **
## Channel33   -6.525e+04  5.407e+04  -1.207  0.27294
## Channel34    1.007e+05  6.589e+04   1.528  0.17738
## Channel35   -2.841e+03  1.214e+04  -0.234  0.82266
```

```
## Channel36   -2.268e+04  2.295e+04  -0.988  0.36127
## Channel37   -4.479e+04  1.292e+04  -3.468  0.01334 *
## Channel38    3.209e+04  1.843e+04   1.742  0.13221
## Channel39    1.992e+04  2.067e+04   0.964  0.37246
## Channel40   -9.833e+03  2.431e+04  -0.404  0.69988
## Channel41    1.659e+04  3.648e+04   0.455  0.66531
## Channel42   -1.829e+04  3.528e+04  -0.519  0.62260
## Channel43   -2.423e+04  2.427e+04  -0.998  0.35669
## Channel44    3.246e+04  2.013e+04   1.613  0.15793
## Channel45   -8.089e+03  4.023e+04  -0.201  0.84728
## Channel46    7.065e+03  2.810e+04   0.251  0.80990
## Channel47   -4.062e+04  1.007e+04  -4.034  0.00685 **
## Channel48    9.080e+04  2.618e+04   3.469  0.01332 *
## Channel49   -6.647e+04  2.372e+04  -2.803  0.03105 *
## Channel50   -4.196e+04  2.856e+04  -1.469  0.19213
## Channel51    1.097e+05  5.572e+04   1.968  0.09661 .
## Channel52   -1.148e+05  6.376e+04  -1.800  0.12196
## Channel53    9.525e+04  7.450e+04   1.278  0.24830
## Channel54   -4.534e+04  7.363e+04  -0.616  0.56067
## Channel55   -1.535e+03  4.933e+04  -0.031  0.97618
## Channel56   -2.377e+03  2.109e+04  -0.113  0.91394
## Channel57    3.174e+04  1.005e+04   3.158  0.01961 *
## Channel58    2.221e+03  1.048e+04   0.212  0.83915
## Channel59   -8.504e+04  2.574e+04  -3.304  0.01634 *
## Channel60    6.382e+04  1.607e+04   3.972  0.00735 **
## Channel61    2.151e+04  1.234e+04   1.742  0.13211
## Channel62   -2.859e+04  1.065e+04  -2.685  0.03631 *
## Channel63    1.796e+04  9.187e+03   1.955  0.09838 .
## Channel64    5.759e+04  3.526e+04   1.633  0.15354
## Channel65   -1.470e+05  6.911e+04  -2.127  0.07752 .
## Channel66    9.121e+04  4.461e+04   2.045  0.08688 .
## Channel67   -5.733e+03  2.197e+04  -0.261  0.80288
## Channel68   -6.290e+04  2.192e+04  -2.870  0.02843 *
## Channel69    6.421e+04  2.074e+04   3.096  0.02121 *
## Channel70   -1.749e+04  1.581e+04  -1.106  0.31111
## Channel71   -7.248e+03  1.934e+04  -0.375  0.72075
## Channel72    3.406e+04  1.185e+04   2.873  0.02830 *
## Channel73   -2.100e+04  1.132e+04  -1.855  0.11308
## Channel74   -3.314e+04  1.220e+04  -2.717  0.03480 *
## Channel75    7.039e+04  2.054e+04   3.427  0.01402 *
## Channel76   -3.187e+04  1.736e+04  -1.836  0.11597
## Channel77    2.061e+04  1.810e+04   1.138  0.29832
## Channel78   -1.180e+04  2.273e+04  -0.519  0.62225
## Channel79    2.669e+04  2.997e+04   0.890  0.40750
## Channel80   -6.051e+04  1.483e+04  -4.080  0.00650 **
## Channel81    1.386e+03  2.628e+04   0.053  0.95966
## Channel82    1.020e+05  4.694e+04   2.173  0.07275 .
## Channel83   -1.706e+05  4.688e+04  -3.640  0.01083 *
## Channel84    1.097e+05  2.892e+04   3.792  0.00905 **
## Channel85   -1.294e+05  3.600e+04  -3.594  0.01145 *
## Channel86    2.130e+05  4.345e+04   4.903  0.00270 **
## Channel87   -1.198e+05  3.818e+04  -3.139  0.02011 *
## Channel88   -2.199e+04  6.085e+04  -0.361  0.73021
## Channel89    7.974e+04  5.077e+04   1.571  0.16733
```

```
## Channel90   -1.711e+05  5.499e+04  -3.112  0.02079 *
## Channel91    2.107e+05  6.406e+04   3.289  0.01663 *
## Channel92   -1.959e+05  7.171e+04  -2.733  0.03407 *
## Channel93    2.874e+05  9.937e+04   2.892  0.02762 *
## Channel94   -3.064e+05  9.601e+04  -3.191  0.01881 *
## Channel95    2.048e+05  6.220e+04   3.292  0.01656 *
## Channel96   -5.600e+04  2.929e+04  -1.912  0.10441
## Channel97   -1.318e+04  3.050e+04  -0.432  0.68065
## Channel98   -2.724e+04  2.107e+04  -1.292  0.24375
## Channel99    3.556e+04  1.382e+04   2.573  0.04218 *
## Channel100  -1.206e+04  4.264e+03  -2.828  0.03006 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3191 on 6 degrees of freedom
## Multiple R-squared:      1,  Adjusted R-squared:  0.9994
## F-statistic:  1651 on 100 and 6 DF,  p-value: 1.058e-09
```




The summary of the model is used later to explain the quality of the model.

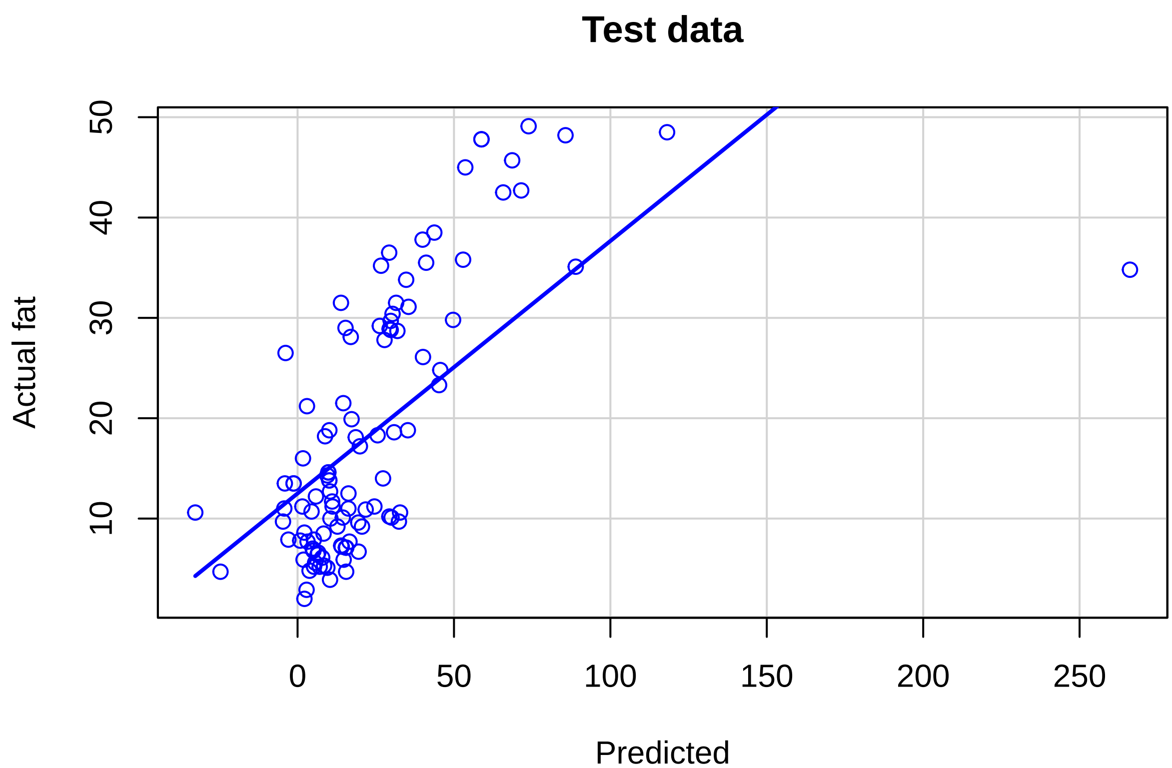
The mean square error (MSE) for the train data is

```
## [1] 0.005709117
```

The mean square error (MSE) for the test data is

```
## [1] 722.4294
```

Since the MSE for the test data is much higher than the train data, the predicted data versus the actual data is plotted in a scatter plot. The MSE for the test data is much higher since there are a point that really does not fit the model. While for the train data there is almost a perfect fit. This leads to the conclusion that the model is overfitted for the train data and the trained model is not suitable for the test data. That the adjusted $R^2 = 0.9994$ support this overfitted train data theory.

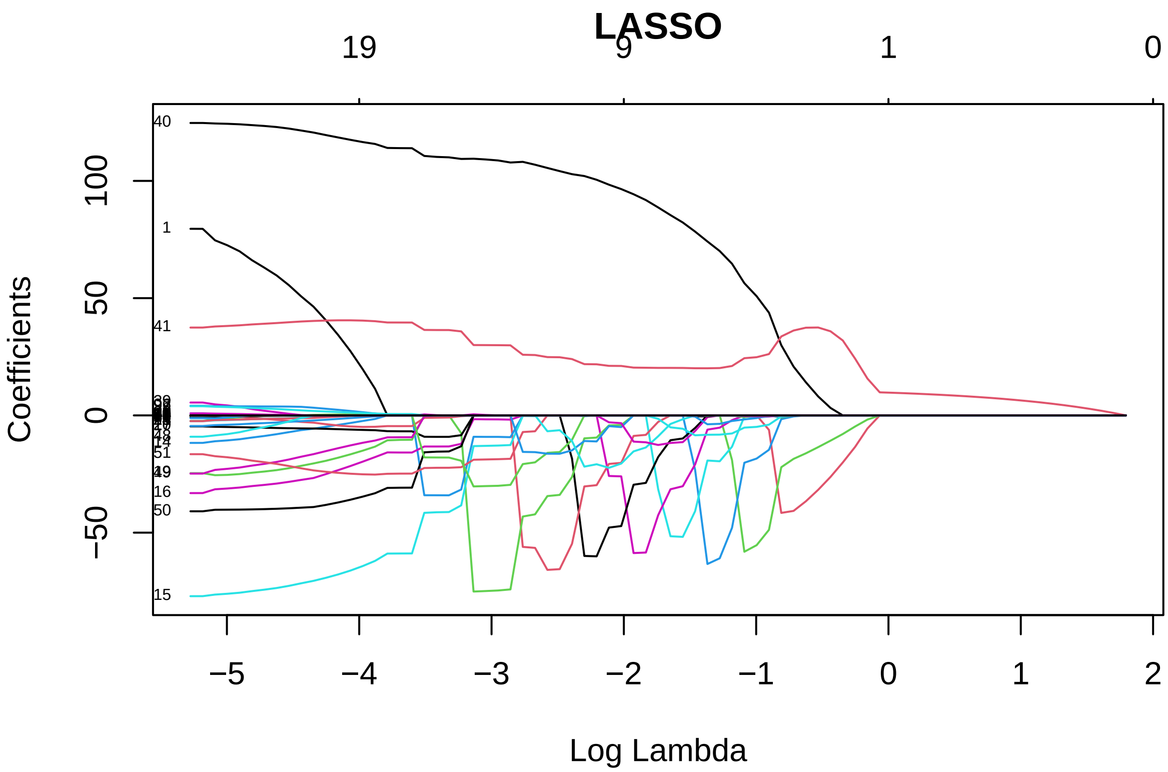



### 1.2 The LASSO cost function

The LASSO cost function is

$$\frac{1}{n}||\boldsymbol{X\beta} - \boldsymbol{y}||_2^2 + \lambda||\boldsymbol{\beta}||_1$$

### 1.3 LASSO regression




By looking at the graph above it can be interpreted that as the penalty $\lambda$ increases the important channels will differ. Channel 41 has a high contribution as it does not go to 0 as fast as the other parameters.

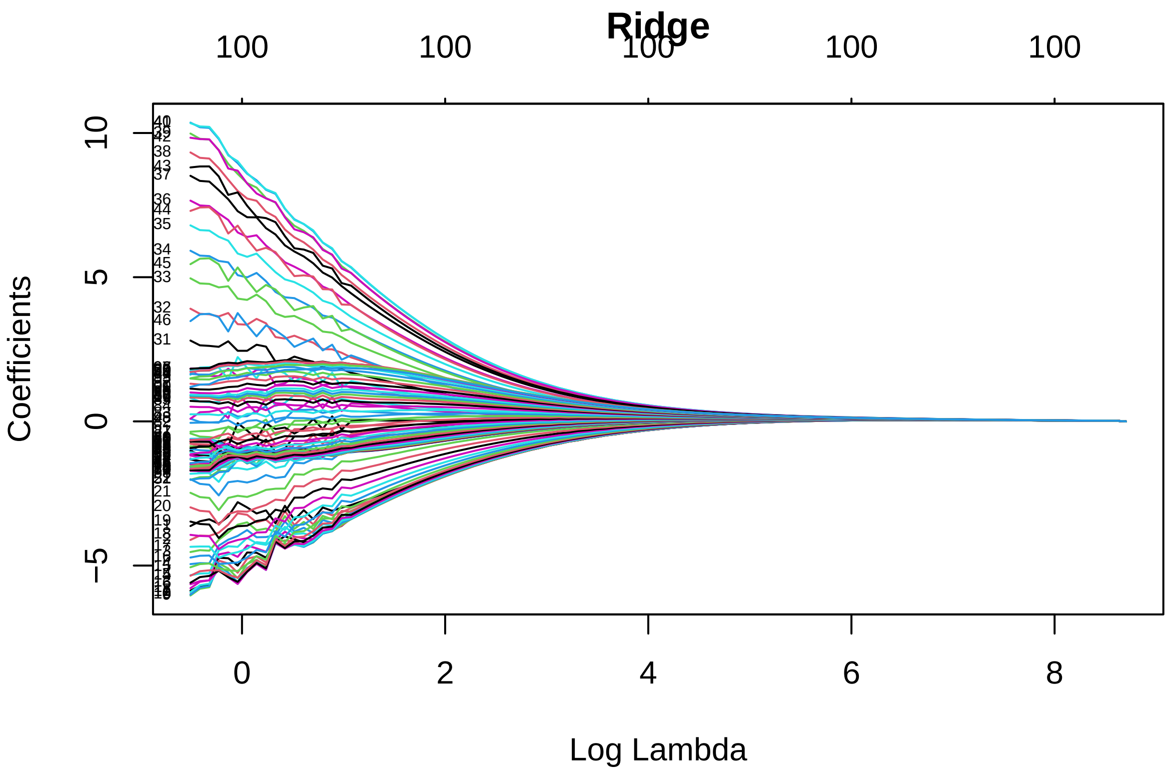
By looking at the degrees of freedom in the print of the fit, it can be seen that the $\lambda$ in the interval from 0.7082 to 0.8530 gives a penalty factor with only three features.

```
##
## Call:  glmnet(x = x, y = y, family = "gaussian", alpha = 1)
##
##    Df  %Dev Lambda
## 1   0  0.00 6.0180
## 2   1  3.91 5.4830
## 3   1  7.16 4.9960
## 4   1  9.86 4.5520
## 5   1 12.10 4.1480
## 6   1 13.96 3.7800
## 7   1 15.50 3.4440
## 8   1 16.78 3.1380
## 9   1 17.85 2.8590
## 10  1 18.73 2.6050
## 11  1 19.46 2.3740
## 12  1 20.07 2.1630
## 13  1 20.58 1.9710
## 14  1 21.00 1.7960
## 15  1 21.34 1.6360
```

```
## 16  1 21.63 1.4910
## 17  1 21.87 1.3580
## 18  1 22.07 1.2380
## 19  1 22.24 1.1280
## 20  1 22.38 1.0270
## 21  1 22.49 0.9362
## 22  3 29.24 0.8530
## 23  3 38.34 0.7773
## 24  3 45.91 0.7082
## 25  4 52.22 0.6453
## 26  4 57.47 0.5880
## 27  5 61.86 0.5357
## 28  5 65.53 0.4881
## 29  9 69.56 0.4448
## 30  8 73.16 0.4053
## 31  8 76.06 0.3693
## 32 10 78.45 0.3365
## 33  8 80.59 0.3066
## 34  7 82.37 0.2793
## 35 11 83.76 0.2545
## 36 10 85.16 0.2319
## 37  8 86.36 0.2113
## 38  9 87.19 0.1925
## 39 10 88.01 0.1754
## 40  7 88.76 0.1598
## 41  8 89.27 0.1456
## 42  9 89.84 0.1327
## 43 10 90.17 0.1209
## 44  8 90.63 0.1102
## 45  9 90.86 0.1004
## 46  9 91.30 0.0915
## 47  9 91.57 0.0833
## 48 10 91.72 0.0759
## 49  9 91.96 0.0692
## 50 10 92.08 0.0630
## 51  8 92.48 0.0574
## 52  9 92.55 0.0523
## 53 11 92.59 0.0477
## 54 13 92.63 0.0435
## 55 11 93.04 0.0396
## 56 10 93.12 0.0361
## 57 12 93.13 0.0329
## 58 14 93.17 0.0300
## 59 12 93.52 0.0273
## 60 16 93.53 0.0249
## 61 19 93.54 0.0227
## 62 16 93.89 0.0206
## 63 18 94.11 0.0188
## 64 19 94.30 0.0171
## 65 21 94.45 0.0156
## 66 22 94.59 0.0142
## 67 24 94.70 0.0130
## 68 23 94.78 0.0118
## 69 23 94.86 0.0108
```

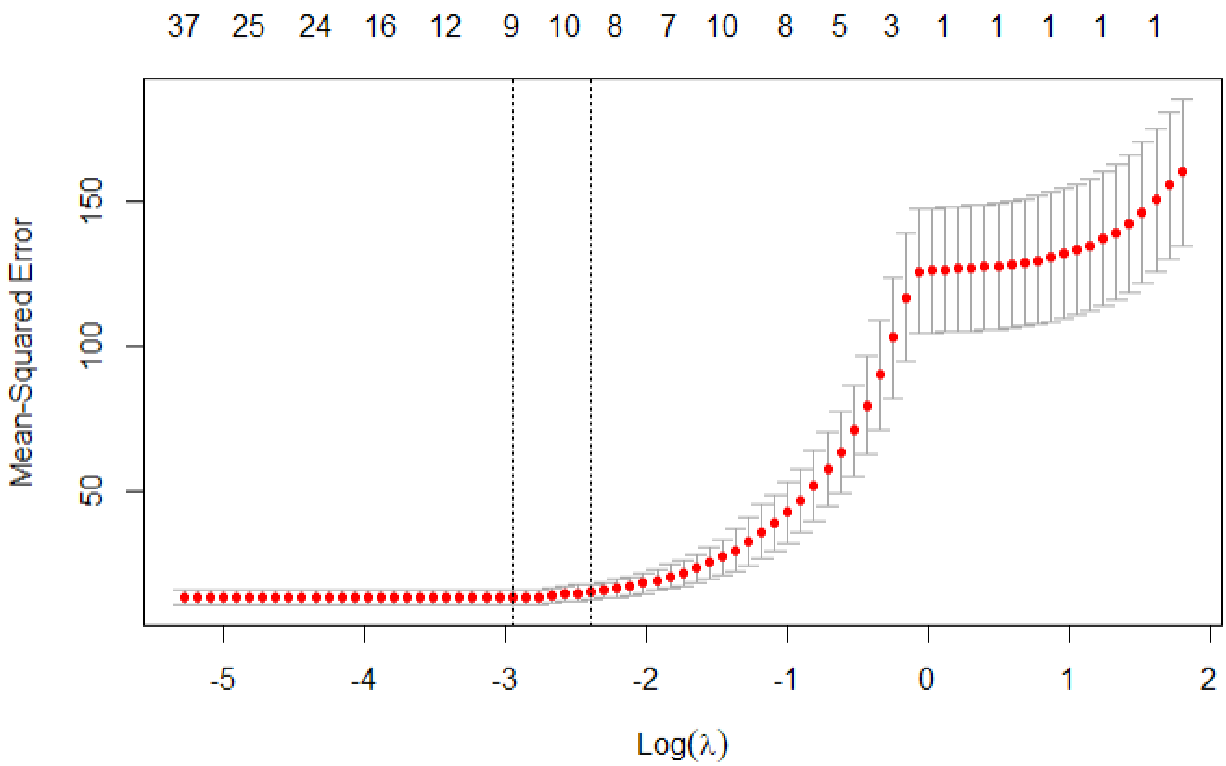
```
## 70 24 94.93 0.0098
## 71 26 94.98 0.0089
## 72 25 95.03 0.0081
## 73 25 95.08 0.0074
## 74 29 95.11 0.0068
## 75 33 95.14 0.0062
## 76 35 95.19 0.0056
## 77 37 95.19 0.0051
```

### 1.4 Ridge regression




One conclusion is that ridge is not suitable for many variables since all the coefficients go towards 0 at the same time.

### 1.5 Cross-validation and scatterplot

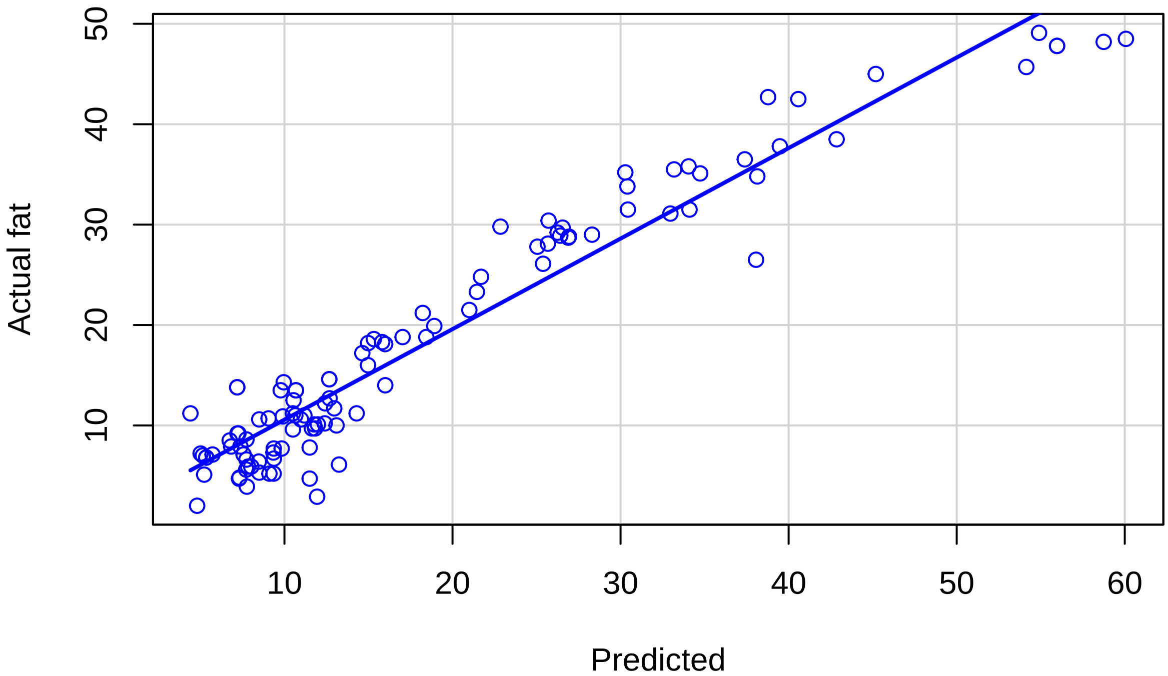



The $\lambda$ that gives the smallest cross validation error is:

```
## [1] 0.05234206
```

and the number of variables chosen were 9, according to the plot. The $\log \lambda = -4$ is not statistically significantly better since is has the same MSE as the min lambda but more dependent variables.

According to the plot below, the fit for the optimal $\lambda$ is much better than the original model.




The MSE of the LASSO model for the optimal $\lambda$ is

```
## [1] 13.93654
```

And thus the LASSO model for the optimal $\lambda$ is a much better model, remember the linear regression model that had an MSE of 722.

# 2. Assignment 2 - Decision trees and logistic regression for bank marketing

## 2.1

```
# ASSIGNMENT 2

# Data import
df <- read.csv("data/bank-full.csv", stringsAsFactors = TRUE, sep=";", header=TRUE)
# remove variable "duration"
df=df[,-12]
# Split data
n=dim(df)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.4))
train=df[id,]
id1=setdiff(1:n, id)
set.seed(12345)
id2=sample(id1, floor(n*0.3))
valid=df[id2,]
id3=setdiff(id1,id2)
test=df[id3,]
```

## 2.2

```
# Training models using training data and different parameters
model_1 <- tree(y~., train)
model_2 <- tree(y~., train, minsize=7000)
model_3 <- tree(y~., train, mindev=0.0005)
```

Misclassification Error (training data):

```
## mmce_train_1 mmce_train_2 mmce_train_3
##   0.10484406   0.10484406   0.09400575
```

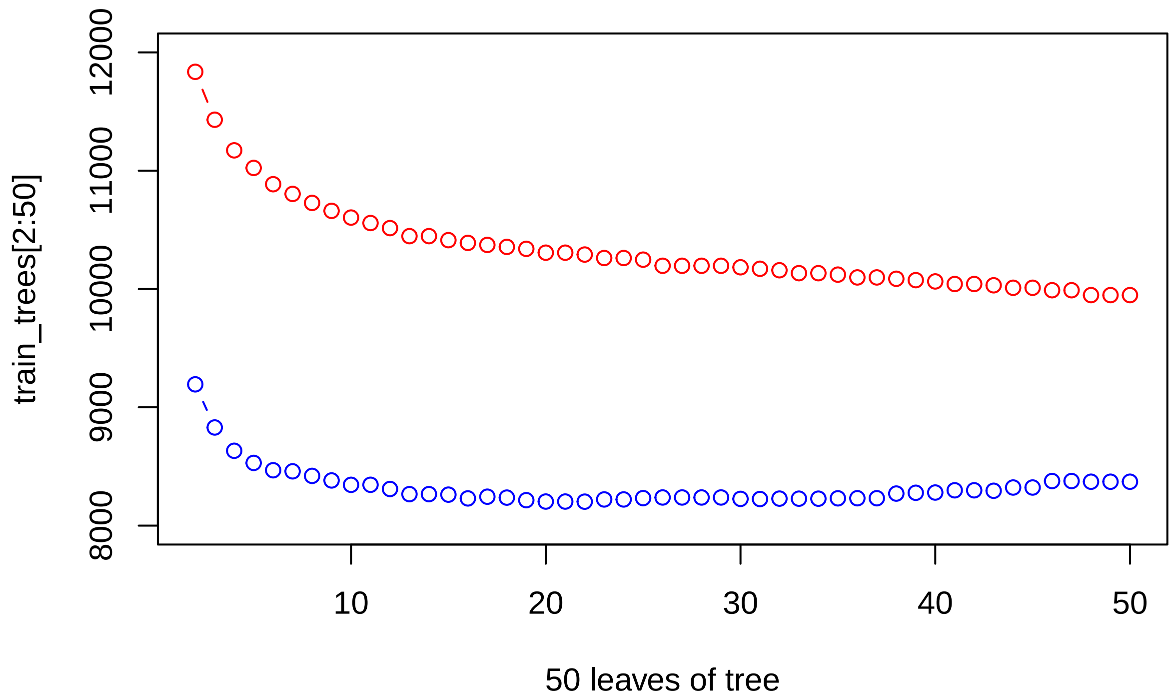
Misclassification Error (validation data):

```
## mmce_val_1 mmce_val_2 mmce_val_3
##  0.1092679  0.1092679  0.1119221
```

The decision trees does not seem to be overfitted, since their is no big deviance between the misclassification rates of the two datasets. We noted that change in deviance to 0.0005 is resulting in more accurate classifications. And we also noted that setting the minimum node size to 7000 doesn't change the classification rate at all compared to the default settings.

## 2.3

Graph of 50 leaves of tree:




Optimal number of node:

```
## [1] 22
```

The optimal number of nodes are those with the lowest deviance when validation data is used. In this case the optimal node is 22.

And in order to make the decision model optimal following features are used:

```
## [1] "poutcome" "month"    "contact"  "pdays"    "age"      "day"      "balance"
## [8] "housing"  "job"
```

### 2.4

Confusion Matrix of optimal tree:

```
##          actual
## predicted    no   yes
##       no  11872  1371
##       yes   107   214
```

Misclassification Error of optimal tree:

```
## [1] 0.1089649
```

F1 Score of optimal tree:

```
## [1] 0.224554
```

Accuracy of optimal tree:

```
## [1] 0.8910351
```

The accuracy of the model is 0.8910351 This means almost 90% of predictions are correct, and hence the model has a good predictive power. But as we have imbalanced data we should prefer F1 score here.

### 2.5

```
loss_tree <- rpart(y~., data = train, method="class",
                   parm = list(loss = matrix(c(0,5,1,0), byrow=TRUE, nrow=2)))
```

Confusion matrix of Loss matrix model:

```
##          actual
## predicted    no   yes
##       no  11979  1585
##       yes     0     0
```

Misclassification Error of Loss matrix model:

```
## [1] 0.1168534
```

Accuracy of Loss matrix model:

```
## [1] 0.8831466
```

The confusion matrix give us the mislassification rate 0.1168534, which means 11.6% misclassification. That is 1% more then in task 4 which is 0.1089649 means 10.8% misclassification. So we can say our optimal model is more accurate then loss matrix model.

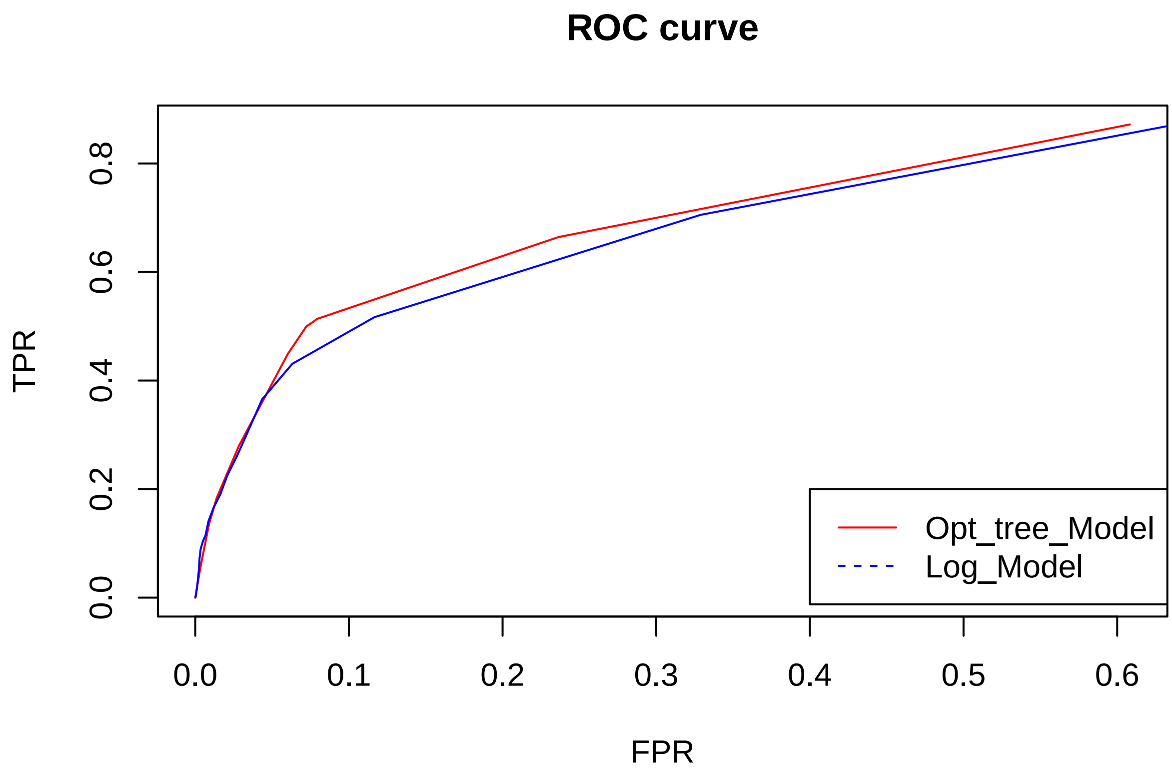
### 2.6

Logistic regression model:

```
## glm(formula = y ~ ., family = binomial(link = "logit"), data = train)
```

Optimal tree model:

```
##
## Classification tree:
## snip.tree(tree = model_3, nodes = c(581L, 17L, 577L, 79L, 37L,
## 77L, 14L, 576L, 153L, 580L, 6L, 1157L, 15L, 16L, 5L, 1156L, 156L,
## 152L, 579L))
## Variables actually used in tree construction:
## [1] "poutcome" "month"    "contact"  "pdays"    "age"      "day"      "balance"
## [8] "housing"  "job"
## Number of terminal nodes:  22
## Residual mean deviance:  0.5698 = 10290 / 18060
## Misclassification error rate: 0.1039 = 1879 / 18084
```




As both models ROC curve looks similar here so its hard to tell which one is better here. So precision recall curve could be a better choice to differentiate between them.

# 3. Assignment 3 - PCA

We first scaled all the variables except ViolentCrimesPerPop and implemented PCA using eigen by first calculating the covariance matrix of the scaled data sd then using the eigen() on the covariance matrix to get the eigen vectors. We obtain our PCs using the matrix multiplication between the scaled data and the eigen vectors.

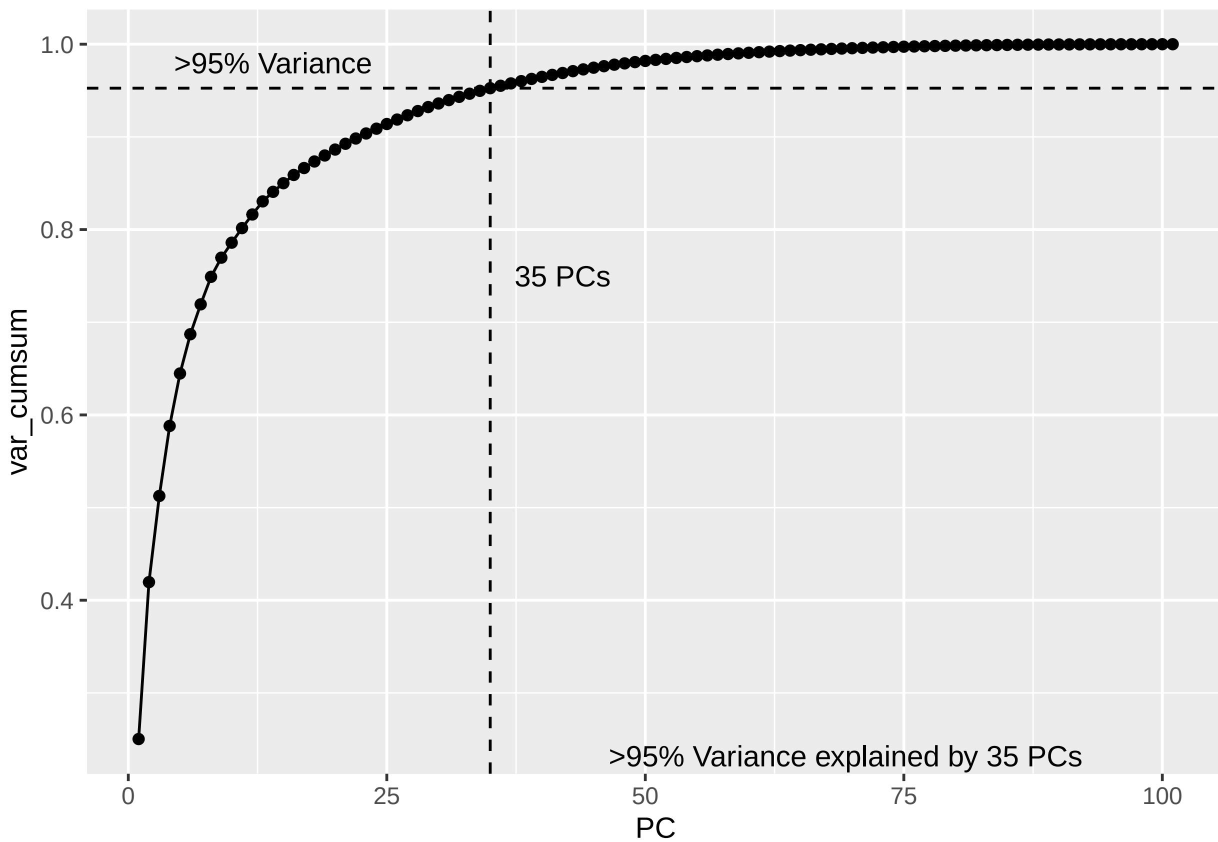
In order to check how many features are needed to obtain atleast 95% variance in the data, we calculate the variance explained by each PC using the following expression:

$$Variance_{PCi} = \frac{eigenvalue_{PCi}}{\sum eigenvalues}$$

The proportion of the variance explained by the first 2 PCs will be the first 2 values in the cumulative variance vector

```
## [1] 0.2502494 0.1693082
```

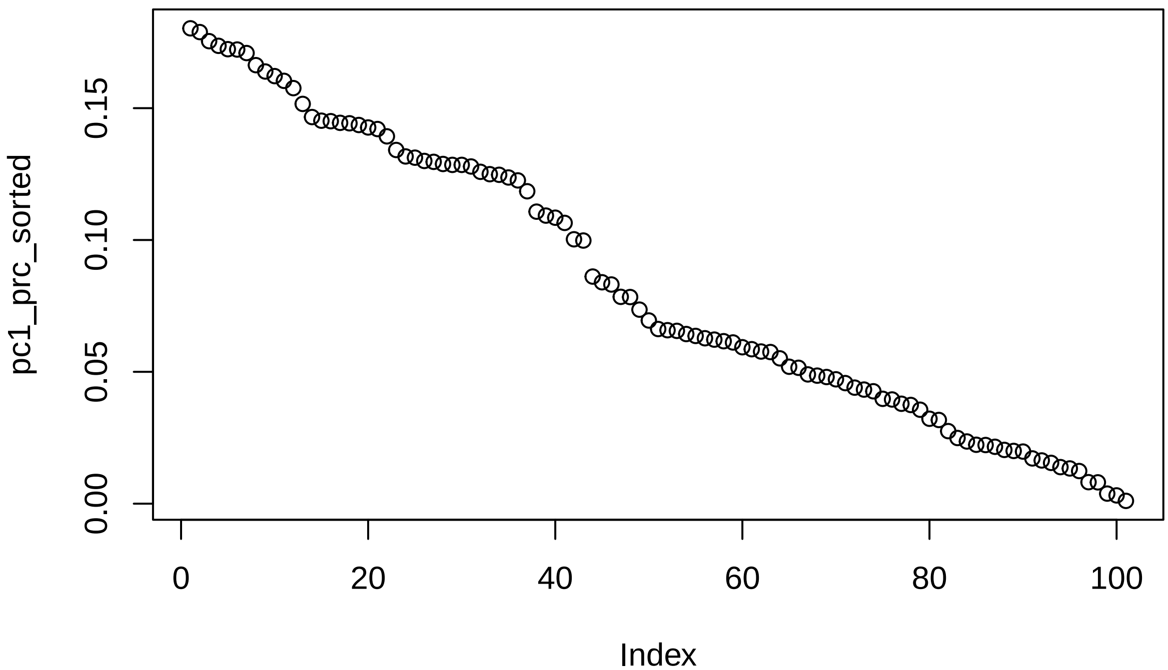
The result shows that the the first 2 PCs explain ~25% and ~17% variance respectively. As for the rest of the PCs, we plot a curve to find out exactly where does the threshold lie for 95% variance




What we find when searching within the variance vector is that we need 35 PCs to explain atleast 95% of the variance. The same has been plotted graphically for ease of interpretation.

Now we move to the next part where we repeat the analysis using the princomp()

We are now interested in finding the number of significant features that contribute to PC1



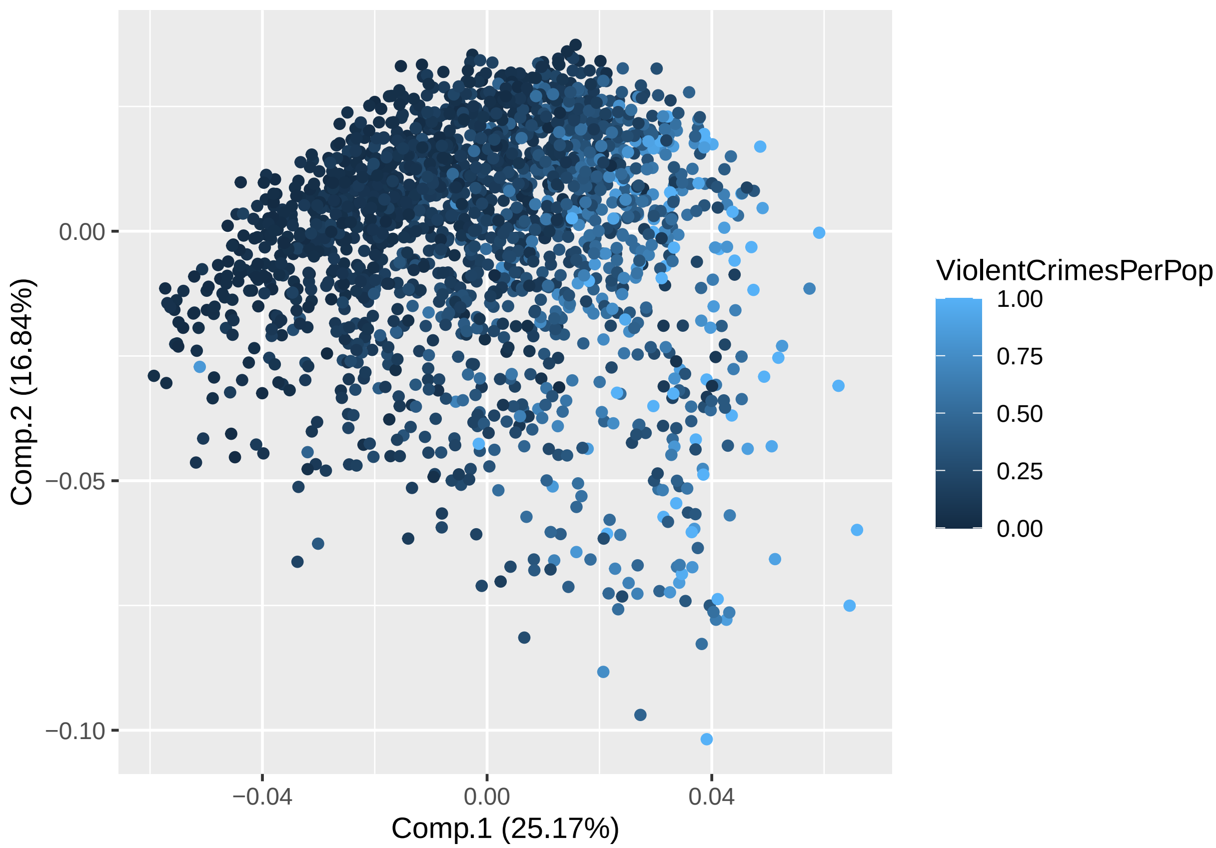

```
##   medFamInc   medIncome PctKids2Par  pctWInvInc  PctFam2Par
##   0.1802784   0.1788524   0.1754108   0.1736464   0.1723687
```

As we can see from the plot, a lot of features contribute in a similar magnitude to PC1. We also sort them by magnitude to find which are the top 5 most significant contributors. The top contributors are:

1. Median family income
2. Median houshold income
3. percent of kids in family housing with 2 parents
4. percentage of households with investment / rent income
5. percentage of families (with kids) that are headed by two parents

The top two factors contributing to violent crimes seem to concern with income. This ties into the next set of important features as well which are also related to poverty levels, income levels etc. The second underlying theme in significant features seems to be kids in the family/household. It might seem to suggest that having multiple dependents in the household can also contribute to pressures resulting in the responsible adult leading to commit violent crimes.

Below we show the plot of PC scores in [PC1, PC2] coordinates where the color of the points is given by ViolentCrimesPerPop




Now we will analyze the data by running a linear regression. We first scale the entire data before splitting into train and test before running linear regression on train data to find significant features and MSE.

```
##     TotalPctDiv  MalePctDivorce    FemalePctDiv PctPersOwnOccup     whitePerCap
##       1.9278475       1.2584421       0.7760016       0.7005451       0.5949093
```

In terms of magnitude of coefficients we have listed our top 5 results, but as we know this alone is not a conclusive summary. So we also look at the p-values to understand the significance of the coefficients and also compute the MSE for both training and test

```
##
## Call:
## lm(formula = ViolentCrimesPerPop ~ 0 + ., data = as.data.frame(train))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.48240 -0.20821 -0.02503  0.14685  2.09335
##
## Coefficients:
##                         Estimate Std. Error t value Pr(>|t|)
## state                  -0.091826   0.034947  -2.628 0.008747 **
## population              0.217809   0.224474   0.970 0.332153
## householdsize          -0.031701   0.173948  -0.182 0.855434
## racepctblack            0.292736   0.065053   4.500 7.69e-06 ***
## racePctWhite           -0.015199   0.187323  -0.081 0.935351
## racePctAsian           -0.005339   0.035204  -0.152 0.879499
## racePctHisp             0.024068   0.060546   0.398 0.691087
## agePct12t21             0.241373   0.204913   1.178 0.239138
```

```
## agePct12t29          -0.245227   0.314595  -0.780 0.435891
## agePct16t24          -0.201657   0.253178  -0.797 0.425951
## agePct65up           -0.007896   0.183387  -0.043 0.965666
## numbUrban            -0.263899   0.224656  -1.175 0.240435
## pctUrban              0.160292   0.054159   2.960 0.003161 **
## medIncome            -0.043808   0.302052  -0.145 0.884716
## pctWWage             -0.470692   0.205819  -2.287 0.022432 *
## pctWFarmSelf          0.081789   0.032434   2.522 0.011850 *
## pctWInvInc           -0.093742   0.145503  -0.644 0.519570
## pctWSocSec            0.170999   0.218197   0.784 0.433429
## pctWPubAsst          -0.132238   0.075005  -1.763 0.078232 .
## pctWRetire           -0.021890   0.078592  -0.279 0.780668
## medFamInc             0.145250   0.293227   0.495 0.620474
## perCapInc             0.303314   0.330378   0.918 0.358822
## whitePerCap          -0.594909   0.281900  -2.110 0.035104 *
## blackPerCap          -0.045087   0.033734  -1.337 0.181708
## indianPerCap         -0.028526   0.020297  -1.405 0.160246
## AsianPerCap           0.018248   0.030833   0.592 0.554113
## HispPerCap            0.004724   0.041514   0.114 0.909425
## NumUnderPov           0.089164   0.079359   1.124 0.261505
## PctPopUnderPov       -0.168559   0.095230  -1.770 0.077061 .
## PctLess9thGrade      -0.059364   0.108809  -0.546 0.585491
## PctNotHSGrad         -0.006407   0.168976  -0.038 0.969761
## PctBSorMore           0.185362   0.135699   1.366 0.172289
## PctUnemployed        -0.022671   0.070998  -0.319 0.749554
## PctEmploy             0.515241   0.177087   2.910 0.003709 **
## PctEmplManu          -0.109346   0.058670  -1.864 0.062685 .
## PctEmplProfServ      -0.111440   0.079402  -1.403 0.160814
## PctOccupManu          0.214418   0.098481   2.177 0.029722 *
## PctOccupMgmtProf      0.203120   0.172956   1.174 0.240545
## MalePctDivorce        1.258442   0.560483   2.245 0.024993 *
## MalePctNevMarr        0.314970   0.132019   2.386 0.017249 *
## FemalePctDiv          0.776002   0.751536   1.033 0.302091
## TotalPctDiv          -1.927848   1.290268  -1.494 0.135489
## PersPerFam            0.094086   0.354604   0.265 0.790819
## PctFam2Par           -0.384100   0.439965  -0.873 0.382883
## PctKids2Par          -0.312748   0.434819  -0.719 0.472169
## PctYoungKids2Par     -0.003168   0.140253  -0.023 0.981983
## PctTeen2Par           0.070052   0.107584   0.651 0.515125
## PctWorkMomYoungKids   0.157424   0.103445   1.522 0.128408
## PctWorkMom           -0.439083   0.126038  -3.484 0.000518 ***
## NumIlleg             -0.060454   0.051305  -1.178 0.238981
## PctIlleg              0.049832   0.069423   0.718 0.473066
## NumImmig             -0.057884   0.026715  -2.167 0.030523 *
## PctImmigRecent        0.048169   0.073230   0.658 0.510850
## PctImmigRec5         -0.159182   0.128827  -1.236 0.216920
## PctImmigRec8          0.287533   0.161318   1.782 0.075023 .
## PctImmigRec10        -0.175226   0.121444  -1.443 0.149410
## PctRecentImmig       -0.262324   0.149414  -1.756 0.079483 .
## PctRecImmig5          0.108578   0.276588   0.393 0.694735
## PctRecImmig8          0.265523   0.361549   0.734 0.462895
## PctRecImmig10        -0.270430   0.291975  -0.926 0.354587
## PctSpeakEnglOnly      0.438277   0.229148   1.913 0.056113 .
## PctNotSpeakEnglWell  -0.026078   0.082501  -0.316 0.752002
```

```
## PctLargHouseFam       -0.055621   0.322430  -0.173 0.863079
## PctLargHouseOccup     -0.245405   0.324681  -0.756 0.449948
## PersPerOccupHous       0.590898   0.518717   1.139 0.254943
## PersPerOwnOccHous     -0.067230   0.358930  -0.187 0.851463
## PersPerRentOccHous    -0.217692   0.144651  -1.505 0.132689
## PctPersOwnOccup       -0.700545   0.879741  -0.796 0.426064
## PctPersDenseHous       0.380857   0.093325   4.081 4.88e-05 ***
## PctHousLess3BR         0.156489   0.123834   1.264 0.206667
## MedNumBR               0.081895   0.032744   2.501 0.012559 *
## HousVacant             0.037059   0.051786   0.716 0.474412
## PctHousOccup          -0.123451   0.094715  -1.303 0.192775
## PctHousOwnOcc          0.475973   0.893791   0.533 0.594489
## PctVacantBoarded       0.037936   0.026430   1.435 0.151537
## PctVacMore6Mos        -0.041532   0.047421  -0.876 0.381369
## MedYrHousBuilt        -0.001574   0.063317  -0.025 0.980167
## PctHousNoPhone         0.085377   0.054406   1.569 0.116938
## PctWOFullPlumb         0.029965   0.027069   1.107 0.268603
## OwnOccLowQuart        -0.267668   0.285759  -0.937 0.349169
## OwnOccMedVal          -0.131936   0.421575  -0.313 0.754384
## OwnOccHiQuart          0.255293   0.224484   1.137 0.255740
## RentLowQ              -0.199445   0.104364  -1.911 0.056317 .
## RentMedian            -0.101737   0.288187  -0.353 0.724154
## RentHighQ             -0.091954   0.172109  -0.534 0.593281
## MedRent                0.477506   0.245863   1.942 0.052430 .
## MedRentPctHousInc      0.160958   0.072702   2.214 0.027083 *
## MedOwnCostPctInc      -0.045054   0.072059  -0.625 0.531972
## MedOwnCostPctIncNoMtg -0.058766   0.046513  -1.263 0.206767
## NumInShelters          0.056162   0.025697   2.186 0.029106 *
## NumStreet              0.039241   0.017616   2.228 0.026154 *
## PctForeignBorn         0.387341   0.117854   3.287 0.001053 **
## PctBornSameState       0.290112   0.110849   2.617 0.009015 **
## PctSameHouse85         0.035439   0.140905   0.252 0.801475
## PctSameCity85          0.106880   0.104841   1.019 0.308265
## PctSameState85        -0.194986   0.121276  -1.608 0.108234
## LandArea               0.010220   0.028034   0.365 0.715517
## PopDens                0.013788   0.038754   0.356 0.722088
## PctUsePubTrans        -0.023612   0.027348  -0.863 0.388152
## LemasPctOfficDrugUn    0.027085   0.016298   1.662 0.096884 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3756 on 897 degrees of freedom
## Multiple R-squared:  0.8666, Adjusted R-squared:  0.8517
## F-statistic: 58.26 on 100 and 897 DF,  p-value: < 2.2e-16
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.2102  0.4503  0.7007  0.9607  3.0023
```

```
##
##  MSE training: 0.1269181
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.1801  0.4503  0.7282  1.0208  3.0023
```

```
##
##  MSE test 0.1967919
```

As we can see from the results of the linear regression, the the significant coefficients are different from what we got earlier. The training error seems within reasonable range as it is lower than the 1st quartile of the data. Test error using the fitted model, on the other hand, is higher than training. Overall the model shows a very high adj. $R^2$ value but also shows overfitting.

We will now create a loss function for the mean square error that depends on $\theta$ and the data. Since, for the optimization part of this problem, we have to compute the evolution of the MSE as the $\theta$ optimizes, we will use a trick to calculate the MSE with the loss function and assign it to a global array that stores the each individual MSE corresponding to the result of the optimization iterations.

```
# optimize theta

# define the loss function to calculate MSE for both train & test

loss_function <- function(theta, X_train, Y_train, X_test, Y_test) {
  error_train <- as.matrix(Y_train) - as.matrix(X_train) %*% theta
  error_test <- as.matrix(Y_test) - as.matrix(X_test) %*% theta
  optim_values_train[i] <<- mean(error_train^2)
  optim_values_test[i] <<- mean(error_test^2)
  i <<- i + 1
  return(mean(error_train^2))
}
```

In continuation, we will now define an optimization function which uses the BFGS method for optimizing $\theta$

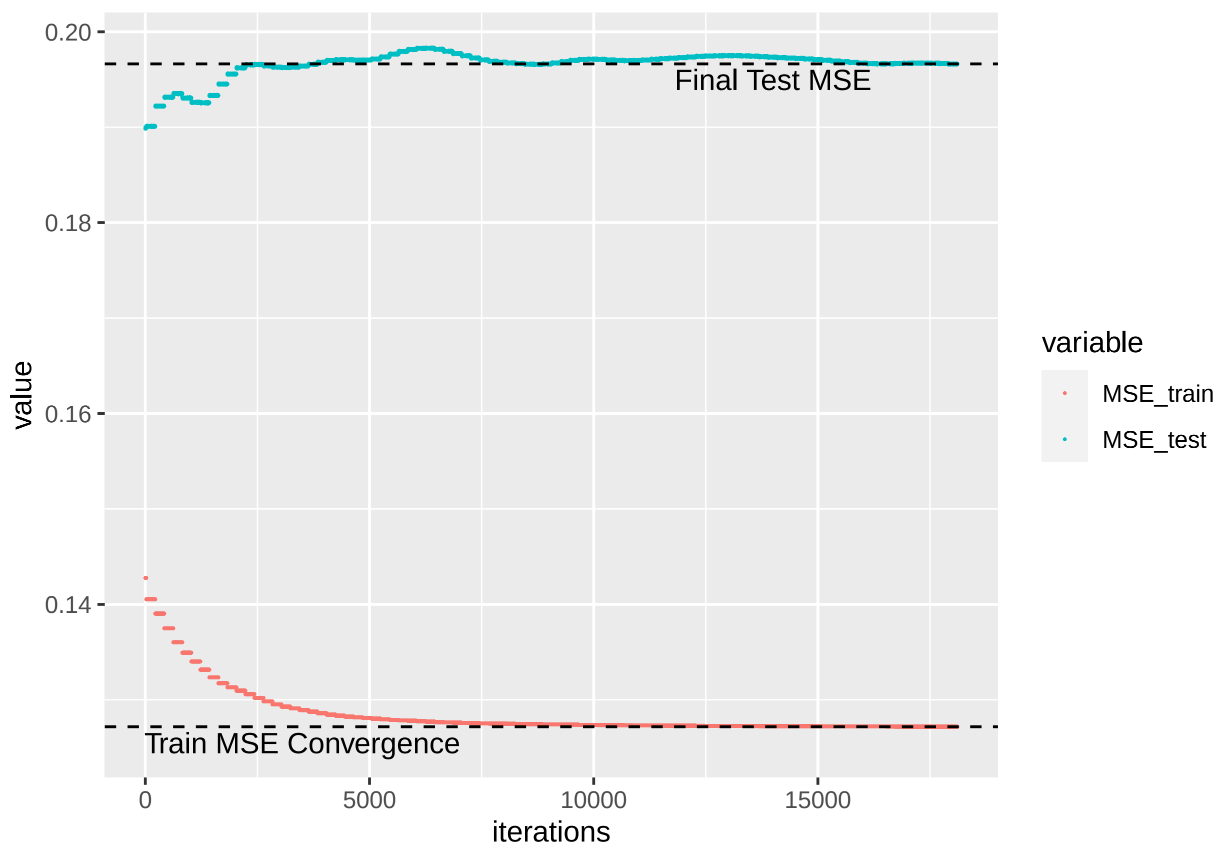
```
# define the optimization using BFGS method

optim_loss <- function(theta, train, test) {
  optim_result <- optim(par = theta,
                        fn = loss_function,
                        method = "BFGS",
                        X_train = train[,1:100],
                        Y_train = train[,101],
                        X_test = test[,1:100],
                        Y_test = test[,101])
  return(optim_result)
}
```

To test our implementation, we will set the initial values of $\theta_0 = 0$ and we will discard the first 2000 values to properly observe the optimization.

```
## $par
##   [1] -0.088519888  0.236590078 -0.014876074  0.300489244 -0.019718428
##   [6] -0.002617776  0.023456514  0.184346870 -0.277187384 -0.165442318
##  [11] -0.028544422 -0.285058634  0.165209609 -0.014055606 -0.485948221
##  [16]  0.085068444 -0.086652771  0.180661236 -0.128613237 -0.012568047
##  [21]  0.128830856  0.321604222 -0.623112597 -0.045137910 -0.029454254
##  [26]  0.019501064 -0.001064140  0.090570553 -0.169647759 -0.063403651
##  [31] -0.005854105  0.181063554 -0.027571018  0.505809391 -0.112695363
##  [36] -0.125508513  0.219708542  0.201672805  0.607057649  0.372479469
##  [41] -0.119312195 -0.370280346  0.079165465 -0.358262087 -0.223433666
##  [46] -0.011372412  0.061110593  0.161730641 -0.443450634 -0.059524775
##  [51]  0.058931870 -0.056185411  0.046921794 -0.149683858  0.273053289
##  [56] -0.175112242 -0.261836589  0.125038048  0.263986818 -0.275474167
##  [61]  0.410906707 -0.036926386 -0.110143301 -0.191130711  0.660025121
##  [66] -0.165242936 -0.188012215 -0.395775920  0.391047954  0.151998756
##  [71]  0.077975599  0.035798248 -0.130359884  0.165785271  0.037173332
```

```
##  [76] -0.038912252 -0.001224700  0.096550184  0.027793722 -0.294044921
##  [81] -0.097339922  0.252105457 -0.205262214 -0.131230492 -0.087233901
##  [86]  0.497350800  0.155459283 -0.045454322 -0.057768248  0.056067607
##  [91]  0.039600322  0.370402529  0.272240059  0.026850445  0.095820087
##  [96] -0.180803538  0.010865666  0.017344119 -0.028304137  0.027677608
##
## $value
## [1] 0.1271727
##
## $counts
## function gradient
##      109      100
##
## $convergence
## [1] 1
##
## $message
## NULL
```



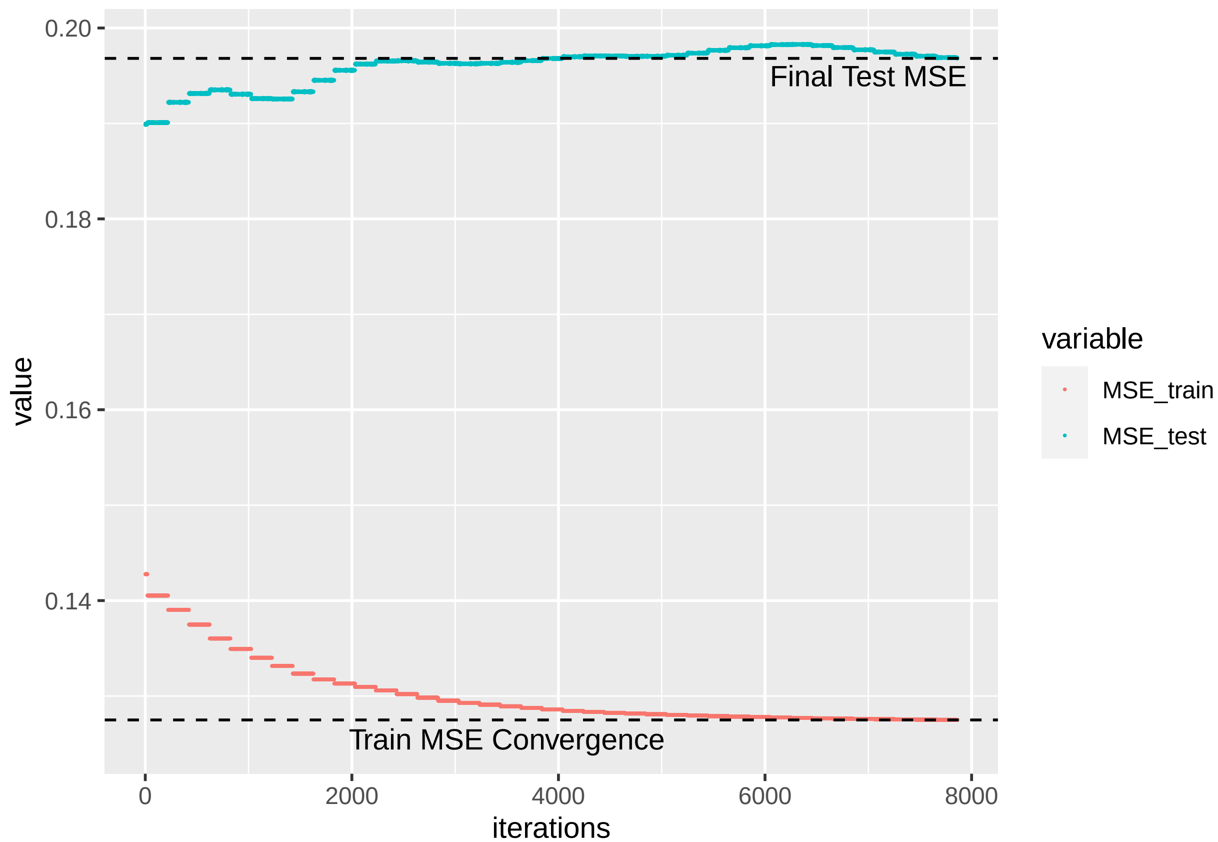

We see that the optimization converges to give us optimum values of $\theta$ that minimizes the MSE of the train data.

We will now observe the MSE for both train and test data to see how it stack up against our earlier calculation using alternate methods

```
## LM Train MSE =  0.1271727
```

```
##
##  LM test MSE =  0.1966377
```

We see that the computed MSE values match those from step 3 pretty closely although the training error is fractionally lower in step 3 vs here and the opposite is true for the test data in this step vs step3. If we look at the MSE plot closely we can see that a potentially acceptable solution may also lie much earlier than convergence when the test error is also lower that the final error value after convergence of $\theta$. This would lie somewhere close to iteration #9000, which would actually mean nearly half the computation time for the optimization, because as we can see from the data, that the convergence happened after ~20000 iterations. This is also true because our training error is well within control if compare it to the quartiles of the actual data. To check this we will update our optim function to specify a control argument with a relative tolerance of $10^{\{-4\}}$ and we see that we are able to achieve a convergence with 9862 iterations.




```
## LM Controlled Train MSE =  0.1274937

##
##  LM Controlled test MSE =  0.1968191
```

As we postulated, we haven't lost too much accuracy on the MSE by using the early stopping criterion.

# APPENDIX - CODE

```
knitr::opts_chunk$set(echo = TRUE)
# loading packages
library(glmnet)
library(car)
library(ROCR)
library(tree)
library(ggplot2)
library(ggfortify)
library(dplyr)
library(reshape2)

# ASSIGNMENT 1

# Load the data
fat_data <- read.csv("data/tecator.csv")
# split data train/test (50/50)
n <- dim(fat_data)[1]
set.seed(12345)
id <- sample(1:n, floor(n*0.5))
train <- as.data.frame(fat_data[id,])
test <- as.data.frame(fat_data[-id,])
channels <- as.data.frame(train[,-1])
channels <- as.data.frame(channels[,-(102:103)])
# fit model on train data
fit_train <- lm(train$Fat ~ ., channels)
summary(fit_train)
scatterplot(train[,102], fit_train$fitted.values, boxplots = FALSE,  smooth = FALSE, xlab = "Predicted"
# test model on test data
p_test <- predict(fit_train, test)
# mean square error train
n2 <- nrow(train)
actual_fat_train <- train[,102]
MSE_train <- 1/n2*sum((actual_fat_train-fitted(fit_train))^2)
MSE_train
# mean square error test
n1 <- nrow(test)
actual_fat_test <- test[,102]
MSE_test <- 1/n1*sum((actual_fat_test-p_test)^2)
MSE_test
# scatter plot of test data
scatterplot(p_test, test[,102], boxplots = FALSE,  smooth = FALSE, xlab = "Predicted", ylab = "Actual fa
x <- as.matrix(channels[,-101])
y <- as.matrix(train$Fat)
fit_lasso <- glmnet(x, y, family = "gaussian", alpha=1) # LASSO
# plot the LASSO
plot(fit_lasso, xvar = "lambda", label = TRUE, main = "LASSO")
print(fit_lasso)
fit_ridge <- glmnet(x, y, family = "gaussian", alpha = 0)
plot(fit_ridge, xvar = "lambda", label = TRUE, main = "Ridge")

set.seed(12345)
```

```r
fit_cv <- cv.glmnet(x, y, alpha=1, family="gaussian")
# plot(fit_cv)
lambda_min <- fit_cv$lambda.min
lambda_min
no_lambda <- coef(fit_cv, s="lambda.min")
# no_lambda
# print(fit_cv)
opt_lambda <- predict(fit_cv, newx = as.matrix(test[,2:101]), type="response", s = lambda_min)
scatterplot(opt_lambda, test[,102], boxplots = FALSE,  smooth = FALSE, xlab = "Predicted", ylab = "Actua

# mean square error LASSO
MSE_opt_lambda <- 1/n2*sum((actual_fat_test-opt_lambda)^2)
MSE_opt_lambda


# ASSIGNMENT 2

# Data import
df <- read.csv("data/bank-full.csv", stringsAsFactors = TRUE, sep=";", header=TRUE)
# remove variable "duration"
df=df[,-12]
# Split data
n=dim(df)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.4))
train=df[id,]
id1=setdiff(1:n, id)
set.seed(12345)
id2=sample(id1, floor(n*0.3))
valid=df[id2,]
id3=setdiff(id1,id2)
test=df[id3,]
library(rpart)
library(tree)
# Training models using training data and different parameters
model_1 <- tree(y~., train)
model_2 <- tree(y~., train, minsize=7000)
model_3 <- tree(y~., train, mindev=0.0005)
pred_train_1 <- predict(model_1, train, type="class")
pred_train_2 <- predict(model_2, train, type="class")
pred_train_3 <- predict(model_3, train, type="class")
pred_val_1 <- predict(model_1, valid, type="class")
pred_val_2 <- predict(model_2, valid, type="class")
pred_val_3 <- predict(model_3, valid, type="class")
confm_train_1 <- table(predicted = pred_train_1, actual = train$y)
confm_train_2 <- table(predicted = pred_train_2, actual = train$y)
confm_train_3 <- table(predicted = pred_train_3, actual = train$y)
confm_val_1 <- table(predicted = pred_val_1, actual = valid$y)
confm_val_2 <- table(predicted = pred_val_2, actual = valid$y)
confm_val_3 <- table(predicted = pred_val_3, actual = valid$y)
mmce_train_1 <- 1 - (sum(diag(confm_train_1)) / sum(confm_train_1))
mmce_train_2 <- 1 - (sum(diag(confm_train_2)) / sum(confm_train_2))
mmce_train_3 <- 1 - (sum(diag(confm_train_3)) / sum(confm_train_3))
```

```r
mmce_training <- c(mmce_train_1 = mmce_train_1, mmce_train_2 = mmce_train_2, mmce_train_3 = mmce_train_3
mmce_val_1 <- 1 - (sum(diag(confm_val_1)) / sum(confm_val_1))
mmce_val_2 <- 1 - (sum(diag(confm_val_2)) / sum(confm_val_2))
mmce_val_3 <- 1 - (sum(diag(confm_val_3)) / sum(confm_val_3))
mmce_validation <- c(mmce_val_1 = mmce_val_1, mmce_val_2 = mmce_val_2, mmce_val_3 = mmce_val_3)
mmce_training
mmce_validation
train_trees <- c()
valid_trees <- c()
for(i in 2:50) {
  prune_train <- prune.tree(model_3, best = i)
  prune_pred <- predict(prune_train, newdata = valid, type = "tree")
  train_trees[i] <- deviance(prune_train)
  valid_trees[i] <- deviance(prune_pred)
}
plot(2:50, train_trees[2:50], type ="b", col="red", ylim = c(8000,12000), xlab="50 leaves of tree" )
points(2:50, valid_trees[2:50], type="b", col="blue")
optimal_node<-which.min(valid_trees)
optimal_node
final_opt_tree <- prune.tree(model_3, best = optimal_node)
pred_optimal <- predict(final_opt_tree, newdata = test, type="class")
as.character(summary(final_opt_tree)$used)
confm_test <- table(predicted = pred_optimal, actual = test$y)
confm_test
mmce_test <- 1 - sum(diag(confm_test)) / sum(confm_test)
mmce_test
TN <- confm_test[1,1]; TP <- confm_test[2,2]; FN <- confm_test[1,2]; FP <- confm_test[2,1]
precision <- (TP)/(TP+FP); recall_score <- (TP)/(TP+FN)
f1_score <- 2*((precision*recall_score)/(precision+recall_score))
f1_score
accuracy_model  <- (TP+TN)/(TP+TN+FP+FN)
accuracy_model
loss_tree <- rpart(y~., data = train, method="class",
                   parm = list(loss = matrix(c(0,5,1,0), byrow=TRUE, nrow=2)))
pred_loss <- predict(loss_tree, test, type="class")
confm_loss <- table(predicted = pred_loss, actual = test$y)
confm_loss
mmce_loss <- 1 - sum(diag(confm_loss))/sum(confm_loss)
mmce_loss
TN_loss <- confm_loss[1,1]; TP_loss <- confm_loss[2,2]; FN_loss <- confm_loss[1,2]; FP_loss <- confm_los
accuracy_model_loss  <- (TP_loss+TN_loss)/(TP_loss+TN_loss+FP_loss+FN_loss)
accuracy_model_loss
library(ROCR)
model_log <- glm(y~.,
               family = binomial(link = "logit"),
               data = train)
model_log$call
pred_log <- predict(model_log, test, type = "response")
summary(final_opt_tree)
pred_opt_tree <- predict(final_opt_tree, newdata = test)
classify_tree <- list(); classify_log <- list(); confm_log <- list(); confm_opt_tree <- list(); tpr_log
for(i in 1:length(pi)){
  classify_tree <- factor(ifelse(pred_opt_tree[,2] > pi[i],  "yes", "no"), levels= c("no", "yes"))
```

```
  classify_log <- factor(ifelse(pred_log > pi[i], "yes", "no"), levels= c("no", "yes"))
  confm_opt_tree <- table(predicted = classify_tree, actual = test$y)
  confm_log <- table(predicted = classify_log, actual = test$y)
  fpr_tree[i] <- confm_opt_tree[2,1]/(confm_opt_tree[1,1]+confm_opt_tree[2,1])
  tpr_tree[i] <- confm_opt_tree[2,2]/(confm_opt_tree[2,2]+confm_opt_tree[1,2])
  fpr_log[i] <- confm_log[2,1]/(confm_log[1,1]+confm_log[2,1])
  tpr_log[i] <- confm_log[2,2]/(confm_log[2,2]+confm_log[1,2])
}
plot(fpr_tree, tpr_tree, type="l", col="red", main = "ROC curve", xlab = "FPR", ylab = "TPR")
lines(fpr_log, tpr_log, col="blue")
legend(x=0.4, y=0.2 , legend=c('Opt_tree_Model', 'Log_Model'), lty=1:2, col=c("red", "blue"))

# ASSIGNMENT 3

data <- read.csv("data/communities.csv")

# scale all variables except ViolentCrimesPerPop
scaled_data <- data %>% mutate(across(.cols = c(1:100), .fns = scale))

# PCA using eigen()
sigma_cov <- cov(scaled_data)
eigen_check <- eigen(sigma_cov)
pc.eigen <- as.matrix(scaled_data) %*% eigen_check$vectors
# Proportion of variance by first 2 components
varexplained <- eigen_check$values / sum(eigen_check$values)
varexplained[1:2]
# How many PC required to explain 95% variance
var_cumsum <- cumsum(varexplained)
pc_95 <- min(which(var_cumsum >=.95))
pc_var_95 <- var_cumsum[pc_95]
pc_plot_df <- data.frame(PC = c(1:101), var_cumsum)
ggplot(pc_plot_df, aes(x = PC, y = var_cumsum)) +
  geom_line() +
  geom_point() +
  geom_hline(yintercept=pc_var_95, linetype = 2) +
  annotate(geom = "text", x = 14, y = 0.98, label = ">95% Variance") +
  geom_vline(xintercept = pc_95, linetype = 2) +
  annotate(geom = "text", x = 42, y = 0.75, label = "35 PCs") +
  geom_text(check_overlap = TRUE,
    label = ">95% Variance explained by 35 PCs",
    x = pc_95,
    y = pc_var_95,
    vjust = 32.2,
    hjust = -0.25
  )
# PCA using princomp()

pc.prc <- princomp(scaled_data, cor = TRUE)
# featurewise contribution to PC1
pc1_prc <- pc.prc$loadings[,1]
pc1_prc_sorted <- sort(abs(pc1_prc), decreasing = TRUE)

# top 5 features of PC1
```

```
plot(pc1_prc_sorted)
pc1_prc_top5 <- head(pc1_prc_sorted,5)
pc1_prc_top5
# plot of PC scores
autoplot(pc.prc, colour = "ViolentCrimesPerPop")
# Scale the original data

fully_scaled_data <- scale(data, center = FALSE)

#Partition into train & test (50/50)
n=dim(fully_scaled_data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=fully_scaled_data[id,]
test=fully_scaled_data[-id,]
##

rm(id, n)


# fit linear regression model to train data
train_lm <- lm(ViolentCrimesPerPop ~ 0 + ., as.data.frame(train))

# top 5 coefficients by magnitude
top5_lm_features <- head(sort(abs(train_lm$coefficients), decreasing = TRUE), 5)
top5_lm_features
# verify most significant coefficients by p-values
summary(train_lm)

# MSE of train and test data
MSE_train <- mean((fitted.values(train_lm) - train[,101])^2)
MSE_test <- mean((predict(train_lm, newdata = as.data.frame(test)) - test[,101])^2)

summary(train[,101])
cat("\n MSE training:", MSE_train, "\n")

summary(test[,101])
cat("\n MSE test", MSE_test, "\n")
# optimize theta

# define the loss function to calculate MSE for both train & test

loss_function <- function(theta, X_train, Y_train, X_test, Y_test) {
  error_train <- as.matrix(Y_train) - as.matrix(X_train) %*% theta
  error_test <- as.matrix(Y_test) - as.matrix(X_test) %*% theta
  optim_values_train[i] <<- mean(error_train^2)
  optim_values_test[i] <<- mean(error_test^2)
  i <<- i + 1
  return(mean(error_train^2))
}
# define the optimization using BFGS method

optim_loss <- function(theta, train, test) {
```

```r
  optim_result <- optim(par = theta,
                        fn = loss_function,
                        method = "BFGS",
                        X_train = train[,1:100],
                        Y_train = train[,101],
                        X_test = test[,1:100],
                        Y_test = test[,101])
  return(optim_result)
}
# testing optimization

theta0 <- rep(0, 100)
i <- 1
optim_values_train <- c()
optim_values_test <- c()
optim_call <- optim_loss(theta = theta0, train, test)
optim_call
x_plot <- c(1:(i-2000))
y_plot_train <- optim_values_train[2000:length(optim_values_train)]
y_plot_test <- optim_values_test[2000:length(optim_values_test)]
plot_error <- data.frame(iterations = x_plot, MSE_train = y_plot_train, MSE_test = y_plot_test)
plot_MSE_melt <- melt(plot_error, id.vars = 1)

# plot the MSE for train and test for each iteration of theta optimization

ggplot(plot_MSE_melt, aes(x = iterations, y = value, colour = variable)) +
  geom_point(size = 0.01) +
  geom_hline(yintercept = optim_call$value, linetype = 2) +
  annotate(geom = "text", x = 3500, y = 0.1255, label = "Train MSE Convergence") +
  geom_hline(yintercept = optim_values_test[length(optim_values_test)], linetype = 2) +
  annotate(geom = "text", x = 14000, y = 0.195, label = "Final Test MSE")
# calculate MSE for optimized theta
optim_MSE_train <- optim_call$value
optim_MSE_test <- optim_values_test[length(optim_values_test)]

cat("LM Train MSE = ", optim_MSE_train)
cat("\n LM test MSE = ", optim_MSE_test)
optim_loss_control <- function(theta, train, test) {
  optim_result <- optim(par = theta,
                        fn = loss_function,
                        method = "BFGS",
                        control = list(reltol = 1e-4),
                        X_train = train[,1:100],
                        Y_train = train[,101],
                        X_test = test[,1:100],
                        Y_test = test[,101])
  return(optim_result)
}

# testing optimization

theta0 <- rep(0, 100)
i <- 1
```

```
optim_values_train <- c()
optim_values_test <- c()
optim_call <- optim_loss_control(theta = theta0, train, test)
x_plot <- c(1:(i-2000))
y_plot_train <- optim_values_train[2000:length(optim_values_train)]
y_plot_test <- optim_values_test[2000:length(optim_values_test)]
plot_error <- data.frame(iterations = x_plot, MSE_train = y_plot_train, MSE_test = y_plot_test)
plot_MSE_melt <- melt(plot_error, id.vars = 1)

# plot the MSE for train and test for each iteration of theta optimization

ggplot(plot_MSE_melt, aes(x = iterations, y = value, colour = variable)) +
  geom_point(size = 0.01) +
  geom_hline(yintercept = optim_call$value, linetype = 2) +
  annotate(geom = "text", x = 3500, y = 0.1255, label = "Train MSE Convergence") +
  geom_hline(yintercept = optim_values_test[length(optim_values_test)], linetype = 2) +
  annotate(geom = "text", x = 7000, y = 0.195, label = "Final Test MSE")

# calculate MSE for optimized theta
optim_MSE_train <- optim_call$value
optim_MSE_test <- optim_values_test[length(optim_values_test)]

cat("LM Controlled Train MSE = ", optim_MSE_train)
cat("\n LM Controlled test MSE = ", optim_MSE_test)
```