STATS 762 Assignment 4

Francis Tang, ID 240887036

Due: 12 June 2019

Packages

```
library(MASS)
library(klaR)
library(nnet)
library(reshape2)
library(ggplot2)
```

```
## Registered S3 methods overwritten by 'ggplot2':
##   method         from
##   [.quosures     rlang
##   c.quosures     rlang
##   print.quosures rlang
```

```
library(glmnet)
```

```
## Loading required package: Matrix
## Loading required package: foreach
## Loaded glmnet 2.0-18
```

```
library(splines)
```

Question 1

(a)

First we read data and only subset the positions contain RDM, RCM and LS:

```
#read data
fifa=read.csv("~/Desktop/STATS 762/Fifa2019.csv")
fifa0 <- fifa[,-1]
fifa1 <- subset(fifa0, Position %in% c("RDM", "RCM", "LS"))
fifa1$Position <- factor(fifa1$Position)
```

Now we first try to fit a multinomial regression:

```
#fit a multinomial regression for the fifa data
fifa.mn <- multinom(Position ~ ., data = fifa1)
```

```
## # weights:  108 (70 variable)
## initial  value 929.425996
## iter  10 value 518.531804
## iter  20 value 504.714000
## iter  30 value 502.813422
## iter  40 value 493.590840
## iter  50 value 473.086244
```

```
## iter  60 value 448.736984
## iter  70 value 435.141344
## iter  80 value 413.418365
## iter  90 value 411.689334
## final  value 411.687916
## converged
```

Make predictions based on the multinomial regression model, the results are shown in the matrix:

```r
#prediction
mn.pred=predict(fifa.mn,fifa1)
#confusion matrix
table(mn.pred,fifa1$Position)
```

```
##
## mn.pred  LS RCM RDM
##     LS  203   4   2
##     RCM   4 320 123
##     RDM   0  67 123
```

Now let's try to fit LDA:

```r
#Fit the LDA for the data
lda.fifa <- lda(Position ~ ., fifa1)
```

Make predictions based on the LDA model, the results are shown in the matrix:

```r
#prediction
lda.pred=predict(lda.fifa,fifa1)
#confusion matrix
table(lda.pred$class,fifa1$Position)
```

```
##
##        LS RCM RDM
##   LS  195   8   3
##   RCM  12 310 116
##   RDM   0  73 129
```

Now let's try to fit QDA:

```r
#Fit the QDA for the train data
qda.fifa <- qda(Position ~ .,fifa1)
```

Make predictions based on the QDA model, the results are shown in the matrix:

```r
#prediction
qda.pred=predict(qda.fifa,fifa1)
#confusion matrix
table(qda.pred$class,fifa1$Position)
```

```
##
##        LS RCM RDM
##   LS  203   8   3
##   RCM   4 336  63
##   RDM   0  47 182
```

It is very obvious that QDA model achieved the best prediction accuracy. So in this case, we will pick QDA as our best model.

(b)

Here we need to find those rows which result in predicting RDM and RCM. This means that the membership probability of LS must be minimised and RDM and RCM need to be as close as possible. In this case, we pick 0.001 as the threshold for LS, [0.4.0.6] for RDM and RCM.

```r
#print the class membership probability
qda.pred.prob.df <- as.data.frame.matrix(qda.pred$posterior)
#extract those rows which not resulting in LS
new1.df <- qda.pred.prob.df[(qda.pred.prob.df$LS < 0.001), ]
#one more step to extract those who result in both RDM and RCM (both probabilities between 0.4 to 0.6)
new2.df <- new1.df[new1.df$RCM > 0.4 & new1.df$RDM > 0.4, ]
new2.df
```

```
##                 LS       RCM       RDM
## 1260  2.189597e-13 0.4275358 0.5724642
## 1306  7.801973e-16 0.5987345 0.4012655
## 1949  5.470421e-19 0.5218050 0.4781950
## 1960  4.135051e-13 0.5030562 0.4969438
## 2098  1.823022e-14 0.4633033 0.5366967
## 2458  1.434826e-16 0.4957068 0.5042932
## 3598  1.708051e-09 0.5496794 0.4503206
## 3683  1.536073e-18 0.5092774 0.4907226
## 3718  1.417468e-10 0.4497646 0.5502354
## 4200  2.356951e-17 0.5214958 0.4785042
## 4305  3.334352e-26 0.4386582 0.5613418
## 4539  1.872576e-27 0.4692426 0.5307574
## 4608  2.440561e-22 0.5997135 0.4002865
## 4866  4.648895e-19 0.5436244 0.4563756
## 4999  9.849763e-07 0.5158382 0.4841608
## 5702  1.188624e-16 0.4997141 0.5002859
## 5743  6.483179e-07 0.5769393 0.4230601
## 5796  2.001461e-18 0.4865715 0.5134285
## 5907  3.065174e-09 0.5142871 0.4857129
## 7238  3.005751e-09 0.5502656 0.4497344
## 7510  9.734893e-08 0.5449031 0.4550968
## 7985  6.256405e-09 0.5033189 0.4966811
## 8685  3.904366e-07 0.5245165 0.4754831
## 9418  2.342970e-07 0.5658330 0.4341667
## 9665  1.915169e-17 0.4174697 0.5825303
## 10163 4.861618e-07 0.5429784 0.4570212
## 10232 7.731262e-15 0.4842512 0.5157488
## 10419 1.435424e-22 0.4796220 0.5203780
## 10581 3.109864e-23 0.4835385 0.5164615
## 10788 5.123872e-08 0.4716201 0.5283798
## 11030 7.123522e-13 0.4193666 0.5806334
## 11374 1.819433e-15 0.5789556 0.4210444
## 12681 3.742474e-16 0.5597038 0.4402962
## 12786 2.062864e-07 0.5979880 0.4020118
## 12823 1.257851e-11 0.5777964 0.4222036
## 12888 5.243587e-16 0.4942304 0.5057696
## 13172 3.071227e-07 0.4294524 0.5705473
## 13382 1.598966e-15 0.5223960 0.4776040
## 14334 2.052584e-10 0.5259151 0.4740849
## 15094 1.647273e-11 0.4060602 0.5939398
```

```
## 15947 3.627611e-08 0.5010714 0.4989285
```

```r
#match those performance scores in the original dataset and print out
subset(fifa1[,-1], rownames(fifa1) %in% rownames(new2.df))
```

```
##       Crossing Finishing HeadingAccuracy ShortPassing Volleys Dribbling
## 1260        70        64              41           78      68        73
## 1306        62        56              63           80      63        72
## 1949        52        47              50           71      40        69
## 1960        78        68              61           80      62        73
## 2098        42        65              73           76      71        64
## 2458        60        56              68           76      50        75
## 3598        73        59              70           74      57        71
## 3683        63        49              47           68      71        68
## 3718        59        63              48           75      48        67
## 4200        59        53              58           74      56        71
## 4305        58        37              58           69      46        63
## 4539        50        39              54           72      38        67
## 4608        42        59              51           79      31        69
## 4866        53        48              55           73      57        64
## 4999        73        66              49           71      63        67
## 5702        44        55              62           68      39        65
## 5743        66        61              60           71      60        69
## 5796        46        49              58           64      35        58
## 5907        59        59              46           70      49        69
## 7238        67        55              64           67      51        62
## 7510        59        63              55           68      60        63
## 7985        52        59              65           75      63        64
## 8685        62        61              66           63      56        65
## 9418        66        58              58           68      64        66
## 9665        48        46              61           72      43        66
## 10163       57        54              48           67      45        70
## 10232       54        56              55           68      56        64
## 10419       41        41              51           65      41        58
## 10581       41        33              60           69      41        61
## 10788       61        52              52           67      57        65
## 11030       53        45              57           68      50        63
## 11374       49        41              68           63      46        56
## 12681       41        35              65           60      39        51
## 12786       53        54              41           64      47        59
## 12823       49        46              53           68      42        72
## 12888       50        48              61           63      46        50
## 13172       57        59              66           62      53        58
## 13382       61        29              45           54      35        62
## 14334       44        44              53           63      41        58
## 15094       48        43              45           67      46        59
## 15947       48        49              55           56      44        56
##       Curve FKAccuracy LongPassing BallControl Acceleration SprintSpeed
## 1260     60         70          74          76           69          69
## 1306     69         67          76          76           67          66
## 1949     45         40          67          71           71          69
## 1960     83         85          79          78           55          45
## 2098     53         71          77          70           74          76
## 2458     42         45          70          77           67          72
## 3598     61         73          73          74           55          60
```

```
## 3683    68          68           70           72           83           74
## 3718    58          63           68           70           69           75
## 4200    66          56           70           74           61           60
## 4305    53          50           67           67           66           63
## 4539    42          41           69           71           70           72
## 4608    48          38           77           69           47           47
## 4866    49          48           66           68           64           65
## 4999    68          65           67           70           78           77
## 5702    53          39           67           67           47           41
## 5743    64          58           68           72           74           68
## 5796    47          42           60           63           63           66
## 5907    55          60           65           71           74           68
## 7238    59          62           65           64           76           72
## 7510    59          66           66           68           66           63
## 7985    66          65           68           68           52           54
## 8685    55          56           60           65           76           73
## 9418    68          68           61           67           65           64
## 9665    48          43           61           70           67           60
## 10163   64          54           64           67           68           68
## 10232   58          53           63           67           64           60
## 10419   38          41           59           64           54           41
## 10581   37          40           66           69           61           68
## 10788   62          57           59           72           70           60
## 11030   48          45           63           67           71           65
## 11374   48          29           58           60           48           44
## 12681   48          49           57           57           67           66
## 12786   46          43           63           62           70           71
## 12823   47          35           62           69           79           66
## 12888   56          45           62           56           54           58
## 13172   62          60           58           62           57           54
## 13382   54          36           50           58           77           76
## 14334   48          38           58           62           64           62
## 15094   49          41           61           61           59           62
## 15947   49          44           55           57           60           64
##       Agility Reactions Balance ShotPower Jumping Stamina Strength
## 1260       71        76      61        70      58      78       66
## 1306       70        73      66        66      53      80       69
## 1949       70        66      75        63      85      90       78
## 1960       58        70      61        78      68      66       65
## 2098       68        82      68        82      68      77       69
## 2458       68        73      67        73      68      76       76
## 3598       63        71      59        77      76      68       75
## 3683       91        74      92        76      82      86       36
## 3718       72        70      71        72      66      88       70
## 4200       64        73      69        73      82      72       67
## 4305       66        68      71        68      68      82       71
## 4539       69        66      62        65      63      77       66
## 4608       64        70      65        73      63      78       71
## 4866       71        67      68        65      71      79       72
## 4999       82        67      67        76      59      80       66
## 5702       58        67      67        67      68      81       78
## 5743       78        64      70        64      64      68       74
## 5796       63        64      62        53      53      94       76
## 5907       69        70      63        71      71      84       72
```

```
## 7238        71         66      64          62       73       82         66
## 7510        73         66      75          67       68       78         62
## 7985        61         64      58          73       61       68         65
## 8685        75         65      70          65       66       73         68
## 9418        73         61      72          65       79       67         74
## 9665        66         65      71          62       65       68         56
## 10163       80         61      77          61       74       68         64
## 10232       73         60      71          72       67       83         73
## 10419       59         71      63          64       58       76         66
## 10581       65         62      67          60       70       82         78
## 10788       72         61      74          59       49       70         64
## 11030       77         60      69          59       63       68         59
## 11374       34         60      45          51       67       69         81
## 12681       62         53      66          57       71       80         73
## 12786       71         62      73          59       63       66         67
## 12823       78         56      53          52       59       74         63
## 12888       50         61      49          65       52       71         80
## 13172       61         56      66          66       67       69         72
## 13382       73         45      67          52       68       86         60
## 14334       63         58      65          53       60       70         66
## 15094       62         55      60          52       58       58         67
## 15947       54         63      50          53       70       79         70
##       LongShots Aggression Interceptions Positioning Vision Penalties
## 1260         70         60            72          76     74        61
## 1306         64         71            76          55     76        51
## 1949         44         95            74          57     63        42
## 1960         78         60            62          63     80        84
## 2098         77         66            66          70     63        70
## 2458         68         80            73          66     71        49
## 3598         70         77            68          67     73        59
## 3683         78         63            74          68     68        57
## 3718         68         76            66          68     75        55
## 4200         66         73            72          61     66        57
## 4305         63         76            72          46     65        49
## 4539         59         74            70          47     70        47
## 4608         54         82            67          56     62        36
## 4866         55         80            70          49     60        48
## 4999         67         53            65          70     73        61
## 5702         63         81            64          55     65        63
## 5743         65         68            68          68     68        69
## 5796         54         77            70          55     60        44
## 5907         66         65            58          56     68        50
## 7238         54         67            70          63     65        68
## 7510         64         74            64          63     66        68
## 7985         70         69            66          65     69        55
## 8685         56         67            64          63     65        54
## 9418         59         59            63          66     72        65
## 9665         51         69            64          60     64        55
## 10163        54         54            60          59     65        48
## 10232        68         74            65          51     58        57
## 10419        58         64            64          54     57        40
## 10581        46         70            61          57     60        40
## 10788        45         60            58          66     67        47
## 11030        51         67            57          63     63        50
```

```
## 11374        39        66             61           49      55         39
## 12681        43        79             61           48      54         52
## 12786        53        62             61           52      66         49
## 12823        41        53             47           57      61         44
## 12888        62        67             54           41      57         58
## 13172        61        63             56           58      62         52
## 13382        43        56             56           42      49         44
## 14334        46        60             58           54      62         43
## 15094        39        57             52           61      61         48
## 15947        50        61             59           54      53         48
##       Composure Marking StandingTackle SlidingTackle GKDiving GKHandling
## 1260         76      76             72            68       12          8
## 1306         70      71             73            68       12         13
## 1949         72      73             75            74        8         15
## 1960         75      70             65            58        9          7
## 2098         69      81             76            68        7         12
## 2458         70      68             70            66       15          8
## 3598         70      57             65            66        6         15
## 3683         67      70             66            62       10          8
## 3718         73      62             66            54        8          9
## 4200         72      72             73            67        9          9
## 4305         66      63             71            68        9         15
## 4539         68      66             70            67        9          6
## 4608         72      69             65            61        8         15
## 4866         69      64             69            61        8          9
## 4999         71      59             59            63        8         15
## 5702         64      62             70            66        7          8
## 5743         69      58             67            64       16         12
## 5796         67      69             70            68       12         12
## 5907         63      58             62            53        9         12
## 7238         64      70             64            63       14         10
## 7510         64      60             64            60        7         14
## 7985         66      65             59            56       15         14
## 8685         66      65             65            62        8         16
## 9418         72      53             61            54       10          9
## 9665         67      64             69            66        6         12
## 10163        62      60             62            58        8         15
## 10232        56      62             66            61        9         12
## 10419        66      58             62            58       11         10
## 10581        58      63             64            63        9         14
## 10788        62      56             55            52        8         11
## 11030        59      60             61            60        9         12
## 11374        52      63             64            59        7          8
## 12681        55      66             61            58        9         13
## 12786        51      60             61            60       11          8
## 12823        68      51             52            50        9          7
## 12888        53      59             65            62       12          6
## 13172        61      54             63            57        9          6
## 13382        55      59             58            59       12          8
## 14334        63      59             58            54        7         10
## 15094        59      58             56            55        8         11
## 15947        56      58             58            57        7          8
##       GKKicking GKPositioning GKReflexes
## 1260          9            14         13
```

```
## 1306          8            12          10
## 1949          9            16          11
## 1960          8            11          16
## 2098         13             7           8
## 2458          5            13          10
## 3598          9            15           7
## 3683         14            10           9
## 3718         14            15          15
## 4200         11             8           9
## 4305         14            10           8
## 4539         13             9          13
## 4608         14             9           6
## 4866          7            15           9
## 4999         16            10          14
## 5702         13            10          14
## 5743         12            11          16
## 5796         10             8          13
## 5907         15            14           7
## 7238         13            14          11
## 7510          8             7           9
## 7985         15            11          13
## 8685         12             8          14
## 9418         16             6          16
## 9665          9             5          12
## 10163        11            13          14
## 10232         8             9          12
## 10419        12            13           7
## 10581        14            12          12
## 10788         9            10           7
## 11030        10             7          12
## 11374        13            13          12
## 12681        13            13           9
## 12786         8             7           6
## 12823         8            11           8
## 12888        12             6           8
## 13172        10             6           9
## 13382        10            13          11
## 14334        10            11           5
## 15094        14             6          11
## 15947        14            13           8
```

(c)

We substitute the number given from the question then use QDA to predict the result:

```
qc.df = data.frame(Crossing = 57.487, Finishing = 57.71277, HeadingAccuracy = 58.64657,
                   ShortPassing = 68.83688, Volleys = 54.40426, Dribbling = 65.74468,
                   Curve = 57.09456, FKAccuracy = 53.16312, LongPassing = 63.4539,
                   BallControl = 68.76123, Acceleration = 67.00591, SprintSpeed = 66.63475,
                   Agility = 68.67376, Reactions = 66.62648, Balance = 67.78369,
                   ShotPower = 67.30378, Jumping = 67.24232, Stamina = 73.51773,
                   Strength =69.20331, LongShots = 61.43735, Aggression = 65.65839,
                   Interceptions = 55.4669, Positioning = 62.02719, Vision = 63.98818,
```

```
                     Penalties = 57.40189, Composure = 65.89835, Marking = 54.90898,
                     StandingTackle = 55.4669, SlidingTackle = 51.90544, GKDiving = 10.69267,
                     GKHandling = 10.63357, GKKicking = 10.83333, GKPositioning = 10.65248,
                     GKReflexes = 10.69031)
qda.pred1=predict(qda.fifa,qc.df)
qda.pred1
```

```
## $class
## [1] RCM
## Levels: LS RCM RDM
##
## $posterior
##             LS       RCM       RDM
## 1 1.964279e-06 0.7651683 0.2348298
```

(d)

Here we fit a classification tree:

```
library(rpart); library(rpart.plot); library(rattle); library(gbm)
```

```
## Rattle: A free graphical interface for data science with R.
## Version 5.2.0 Copyright (c) 2006-2018 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

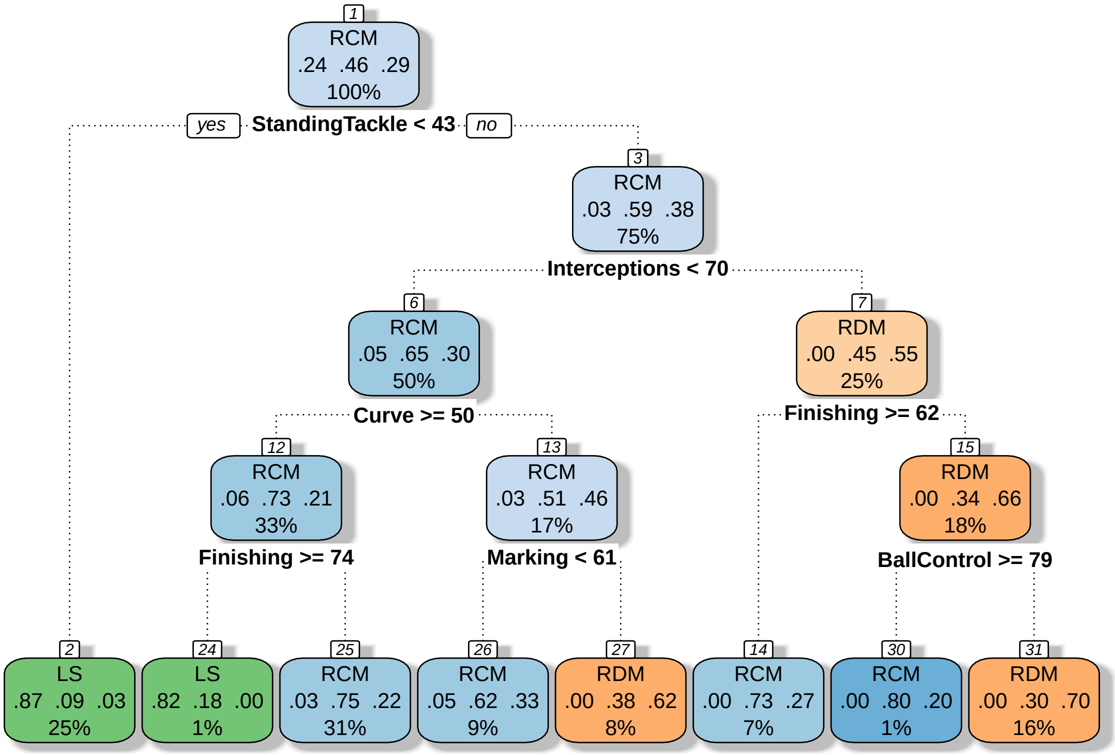
```
## Loaded gbm 2.1.5
```

```
#Fit a classifcation tree
set.seed(1e5)
fifa.cart0 <- rpart(Position~., data=fifa1,method='class',cp=0.001)
fifa.cart0$cptable
```

```
##              CP nsplit rel error    xerror       xstd
## 1  0.364835165      0 1.0000000 1.0000000 0.03187113
## 2  0.053846154      1 0.6351648 0.6417582 0.03039133
## 3  0.017582418      3 0.5274725 0.6109890 0.03002617
## 4  0.015384615      5 0.4923077 0.6153846 0.03008089
## 5  0.013186813      6 0.4769231 0.6131868 0.03005364
## 6  0.008791209      7 0.4637363 0.6021978 0.02991415
## 7  0.007692308      9 0.4461538 0.6043956 0.02994248
## 8  0.007326007     11 0.4307692 0.6087912 0.02999849
## 9  0.006593407     14 0.4087912 0.6065934 0.02997060
## 10 0.004395604     17 0.3846154 0.6021978 0.02991415
## 11 0.003296703     22 0.3626374 0.6087912 0.02999849
## 12 0.003076923     24 0.3560440 0.6153846 0.03008089
## 13 0.002197802     30 0.3318681 0.6175824 0.03010792
## 14 0.001000000     33 0.3252747 0.6373626 0.03034169
```

Now we prune the tree with a particular cp. We noticed that 0.008791209 has the smallest cross validation error 0.6021978, so we use this one to prune the tree. We will also compare this with the full model.

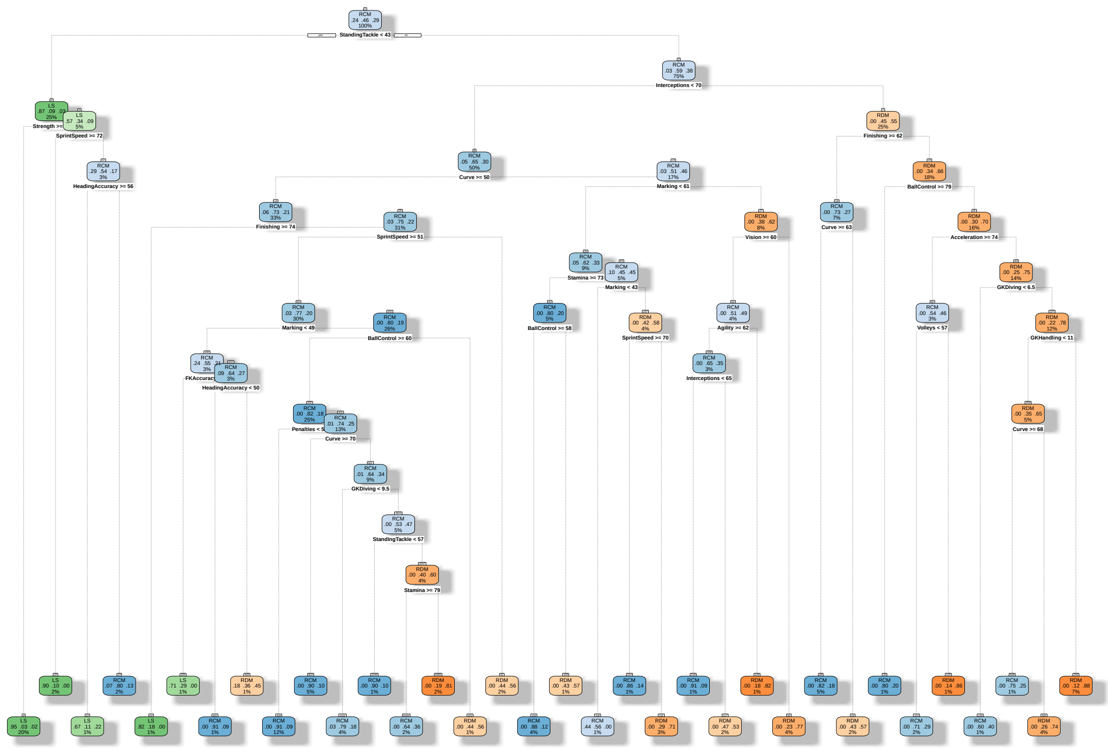
```
#Prune the tree with a particular complexity paramter (cp)

fifa.prune0 <-prune(fifa.cart0,cp=fifa.cart0$cptable[6,1])
fancyRpartPlot(fifa.prune0, uniform=TRUE,main=" ")
```



Rattle 2019−Jun−12 23:59:05 francistang

```r
fifa.prune0.full <-prune(fifa.cart0,cp=fifa.cart0$cptable[14,1])
fancyRpartPlot(fifa.prune0.full, uniform=TRUE,main=" ")
```

```
## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```



Rattle 2019−Jun−12 23:59:05 francistang

(e)

Now we use pruned tree to make predictions also comparing the full model. Although the full model has better predictions, it may suffer from the problem of overfitting.

```
fifa.pred0 <- predict(fifa.prune0,newdata=fifa1[,-1],type='class')
fifa.pred0.full <- predict(fifa.prune0.full,newdata=fifa1[,-1],type='class')
table(fifa.pred0,fifa1$Position)
```

```
##
## fifa.pred0  LS RCM RDM
##        LS  195  22   7
##        RCM  12 301 102
##        RDM   0  68 139
```

```
table(fifa.pred0.full,fifa1$Position)
```

```
##
## fifa.pred0.full  LS RCM RDM
##             LS  199  12   5
##             RCM   6 311  55
##             RDM   2  68 188
```

Comparing to the QDA model, the classification tree does not achieve a better prediction accuracy. So the best model will remain as the QDA model in (a).

```
#confusion matrix
table(qda.pred$class,fifa1$Position)
```

```
##
```

```
##        LS RCM RDM
##   LS  203   8   3
##   RCM   4 336  63
##   RDM   0  47 182
```

Question 2:

(a)

In this question, instead of using Position in Question 1, we use Overall for processing.

```
#keep Overall but get rid of Position
fifa3 <- subset(fifa, Position %in% c("RDM", "RCM", "LS"))
fifa3$Position <- factor(fifa3$Position)
fifa2 <- fifa3[,-2]
```

First, we fit a regression tree with cp=0.001

```
#Setup random numbers
set.seed(1e5)

#Fit a regression tree with cp=0.001
fifa2.cart <- rpart(Overall~. , data=fifa2,method='anova',cp=0.001)
fifa2.cart$cptable
```

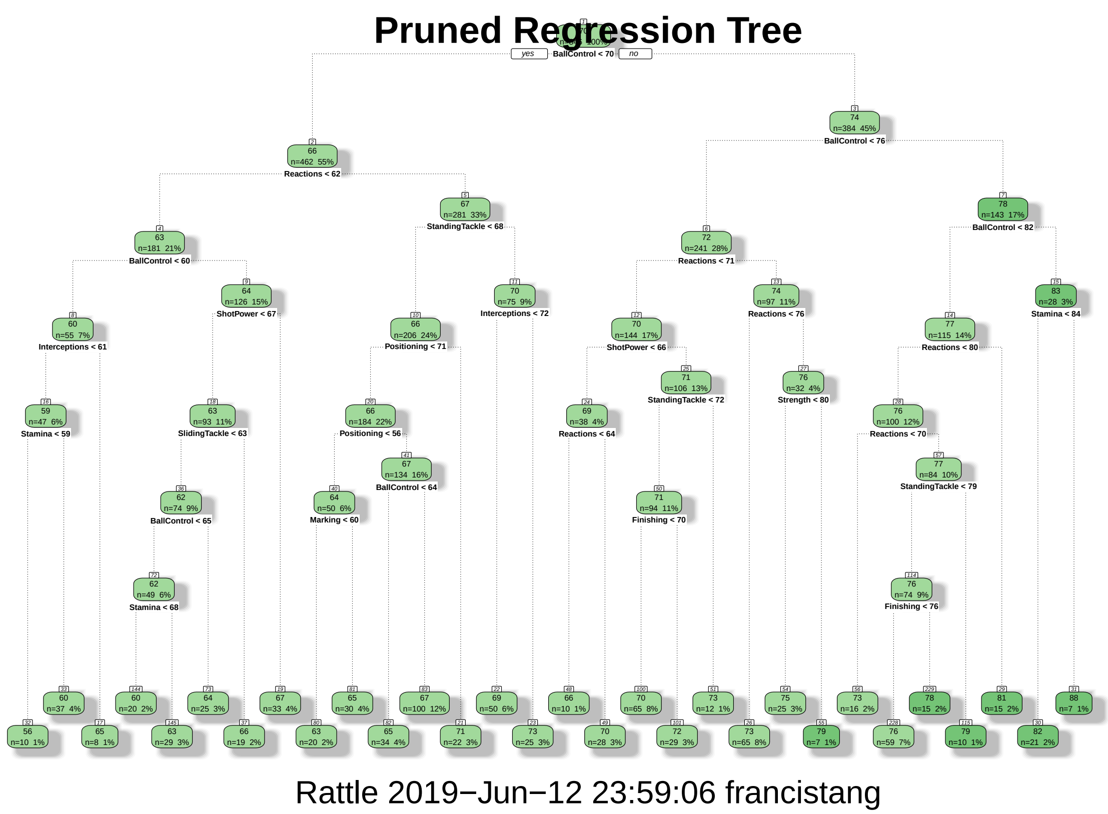
```
##             CP nsplit  rel error    xerror       xstd
## 1  0.492527507      0 1.00000000 1.0009109 0.05147726
## 2  0.106461467      1 0.50747249 0.5334657 0.03031492
## 3  0.078900624      2 0.40101103 0.4292515 0.02326115
## 4  0.033098990      3 0.32211040 0.3581378 0.01978659
## 5  0.025668595      4 0.28901141 0.3174626 0.01859831
## 6  0.025577783      5 0.26334282 0.2955118 0.01666606
## 7  0.022393129      6 0.23776503 0.2835535 0.01655685
## 8  0.016256616      7 0.21537190 0.2717535 0.01612412
## 9  0.010207695      8 0.19911529 0.2585760 0.01562668
## 10 0.009165922      9 0.18890759 0.2453542 0.01490193
## 11 0.008946992     10 0.17974167 0.2419035 0.01427666
## 12 0.006272348     11 0.17079468 0.2357554 0.01387268
## 13 0.005889702     12 0.16452233 0.2294103 0.01359778
## 14 0.005864620     13 0.15863263 0.2265236 0.01361185
## 15 0.005738690     14 0.15276801 0.2265236 0.01361185
## 16 0.005533938     15 0.14702932 0.2231265 0.01314817
## 17 0.005172275     16 0.14149538 0.2241645 0.01337391
## 18 0.004455127     17 0.13632311 0.2219853 0.01333352
## 19 0.003858016     18 0.13186798 0.2199174 0.01297966
## 20 0.003848973     19 0.12800996 0.2160672 0.01292953
## 21 0.003436386     20 0.12416099 0.2131558 0.01290195
## 22 0.003333182     21 0.12072461 0.2095240 0.01280840
## 23 0.003284612     22 0.11739142 0.2076158 0.01238015
## 24 0.002440085     23 0.11410681 0.2034690 0.01217081
## 25 0.002419914     25 0.10922664 0.2002660 0.01229456
## 26 0.002405192     26 0.10680673 0.1996279 0.01228839
## 27 0.002318690     27 0.10440154 0.1997131 0.01229406
## 28 0.001965770     29 0.09976416 0.1980072 0.01214966
```

```
## 29 0.001944549     30 0.09779839 0.1910233 0.01183062
## 30 0.001871194     31 0.09585384 0.1911958 0.01183486
## 31 0.001839383     32 0.09398264 0.1912890 0.01179788
## 32 0.001832131     33 0.09214326 0.1912890 0.01179788
## 33 0.001529021     34 0.09031113 0.1886008 0.01123284
## 34 0.001518805     35 0.08878211 0.1890735 0.01125954
## 35 0.001501875     36 0.08726330 0.1881291 0.01125729
## 36 0.001310566     37 0.08576143 0.1875399 0.01123321
## 37 0.001289367     38 0.08445086 0.1876222 0.01120510
## 38 0.001255896     39 0.08316150 0.1871208 0.01117898
## 39 0.001249564     40 0.08190560 0.1876353 0.01119065
## 40 0.001239840     41 0.08065604 0.1876353 0.01119065
## 41 0.001233527     42 0.07941620 0.1879520 0.01118441
## 42 0.001213148     43 0.07818267 0.1873484 0.01112875
## 43 0.001115453     45 0.07575637 0.1891926 0.01119668
## 44 0.001000000     46 0.07464092 0.1889483 0.01115605
```

When $\alpha = 0.001255896$, the cv-error is minimised: 0.1871208. $\alpha = 0.001965770$ is the largest value in which the corresponding cv-error: 0.1980072 is within the one standard deviation around the minimum error: $0.1871208 + 0.01117898 = 0.19829978$.

Now we prue the trees and plot it out together:

```r
#Prune trees
fifa2.opt=prune(fifa2.cart,cp=fifa2.cart$cptable[28,1])
fancyRpartPlot(fifa2.opt, uniform=TRUE,main="Pruned Regression Tree")
```




```r
fifa2.opt
```

```
## n= 846
```

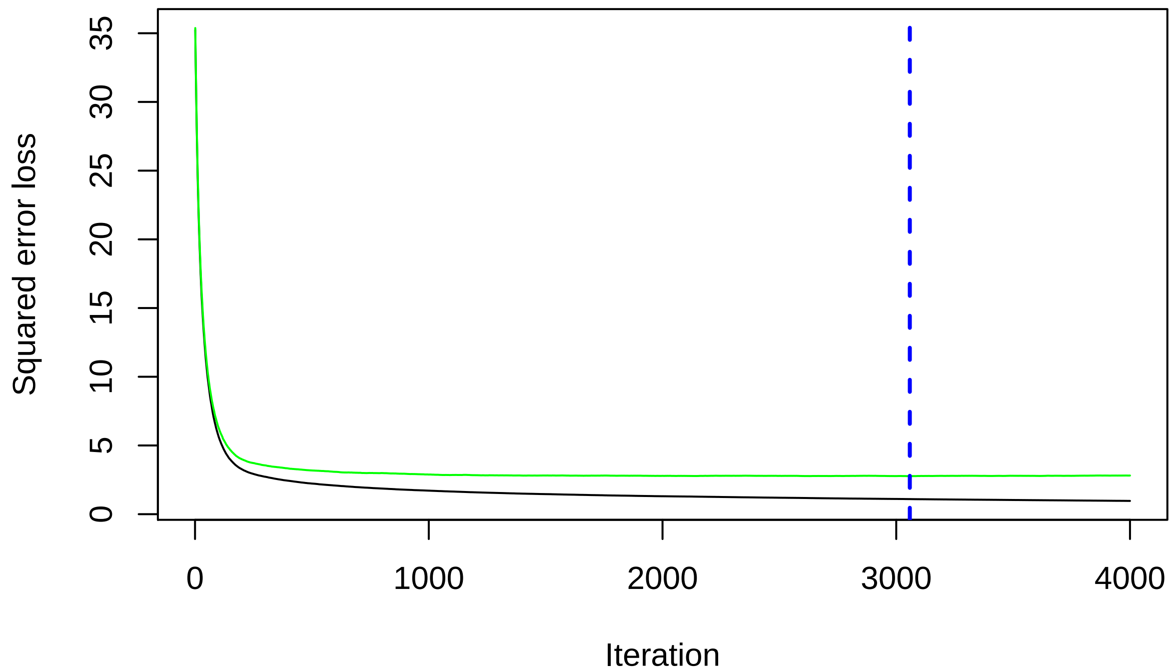
```
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
##   1) root 846 31079.27000 69.51655
##     2) BallControl< 69.5 462  7806.63400 65.63853
##       4) Reactions< 61.5 181  2490.25400 62.76796
##         8) BallControl< 59.5 55   722.80000 59.80000
##          16) Interceptions< 60.5 47   353.23400 58.87234
##            32) Stamina< 58.5 10    53.60000 55.80000 *
##            33) Stamina>=58.5 37   179.72970 59.70270 *
##          17) Interceptions>=60.5 8    91.50000 65.25000 *
##         9) BallControl>=59.5 126  1071.49200 64.06349
##          18) ShotPower< 66.5 93   611.69890 63.11828
##            36) SlidingTackle< 62.5 74   432.50000 62.50000
##              72) BallControl< 64.5 49   255.83670 61.59184
##               144) Stamina< 67.5 20   135.80000 60.10000 *
##               145) Stamina>=67.5 29    44.82759 62.62069 *
##              73) BallControl>=64.5 25    57.04000 64.28000 *
##            37) SlidingTackle>=62.5 19    40.73684 65.52632 *
##          19) ShotPower>=66.5 33   142.54550 66.72727 *
##       5) Reactions>=61.5 281  2864.20600 67.48754
##        10) StandingTackle< 67.5 206  1623.32500 66.47087
##          20) Positioning< 70.5 184  1052.08200 65.92935
##            40) Positioning< 55.5 50   284.50000 64.30000
##              80) Marking< 59.5 20   102.95000 62.55000 *
##              81) Marking>=59.5 30    79.46667 65.46667 *
##            41) Positioning>=55.5 134   585.31340 66.53731
##              82) BallControl< 63.5 34   174.97060 65.02941 *
##              83) BallControl>=63.5 100   306.75000 67.05000 *
##          21) Positioning>=70.5 22    66.00000 71.00000 *
##        11) StandingTackle>=67.5 75   443.12000 70.28000
##          22) Interceptions< 71.5 50   164.02000 69.14000 *
##          23) Interceptions>=71.5 25    84.16000 72.56000 *
##     3) BallControl>=69.5 384  7965.24000 74.18229
##       6) BallControl< 75.5 241  2273.50200 71.92116
##        12) Reactions< 70.5 144   969.30560 70.43056
##          24) ShotPower< 65.5 38   259.07890 68.60526
##            48) Reactions< 63.5 10    45.60000 65.80000 *
##            49) Reactions>=63.5 28   106.67860 69.60714 *
##          25) ShotPower>=65.5 106   538.23580 71.08491
##            50) StandingTackle< 71.5 94   417.15960 70.79787
##             100) Finishing< 69.5 65   229.13850 70.16923 *
##             101) Finishing>=69.5 29   104.75860 72.20690 *
##            51) StandingTackle>=71.5 12    52.66667 73.33333 *
##        13) Reactions>=70.5 97   509.25770 74.13402
##          26) Reactions< 75.5 65   207.53850 73.23077 *
##          27) Reactions>=75.5 32   140.96880 75.96875
##            54) Strength< 79.5 25    43.36000 75.16000 *
##            55) Strength>=79.5 7    22.85714 78.85714 *
##       7) BallControl>=75.5 143  2382.99300 77.99301
##        14) BallControl< 81.5 115   993.44350 76.66957
##          28) Reactions< 79.5 100   639.64000 76.06000
##            56) Reactions< 69.5 16    78.00000 73.00000 *
```

```
##            57) Reactions>=69.5 84   383.28570 76.64286
##             114) StandingTackle< 78.5 74   290.21620 76.32432
##               228) Finishing< 75.5 59   169.55930 75.79661 *
##               229) Finishing>=75.5 15    39.60000 78.40000 *
##             115) StandingTackle>=78.5 10    30.00000 79.00000 *
##          29) Reactions>=79.5 15    68.93333 80.73333 *
##        15) BallControl>=81.5 28   360.85710 83.42857
##          30) Stamina< 83.5 21   138.95240 81.95238 *
##          31) Stamina>=83.5 7    38.85714 87.85714 *
```

(b)

This question requires us to fit a gradient boosting regression tree:

```
#Fit a boosting reg tree
fifa.gbm <- gbm(Overall~., data = fifa2, distribution='gaussian',
                shrinkage = 0.04, n.trees = 4000, cv.folds = 10)
fifa.gbm.perf = gbm.perf(fifa.gbm, method = "cv")
```




```
fifa.gbm.perf
```

```
## [1] 3058
```

From the result above we can conclude that the optimal number of trees are 3058.

(c)

```
#Predict values and find the MSE using optimal regression tree
fifa2.opt.pred <- predict(fifa2.opt,newdata=fifa2[,-1],type='vector')
opt.res=fifa2.opt.pred-fifa2$Overall;
mean(opt.res^2)
```

```
## [1] 3.665008
```

```
#Predict values and find the MSE using optimal gradient boosting regression tree
fifa.gbm.pred <- predict(fifa.gbm,newdata = fifa2[,-1],n.trees = fifa.gbm.perf,type = "response")
#mse
fifa.res=fifa.gbm.pred-fifa2$Overall;
mean(fifa.res^2)
```

```
## [1] 1.097266
```

```
#Predict values and find the MSE using optimal linear regression with lasso
set.seed(1e5)
cv.lasso=cv.glmnet(as.matrix(fifa2[,-1]),as.matrix(fifa2[,1]),alpha=1,standardize=TRUE)
fifa.lasso.pred <- predict(cv.lasso, as.matrix(fifa2[,-1]),
type='response',lambda=cv.lasso$lambda.1se)
lasso.res = fifa.lasso.pred - fifa2$Overall
mse3 = mean(lasso.res^2)
mse3
```

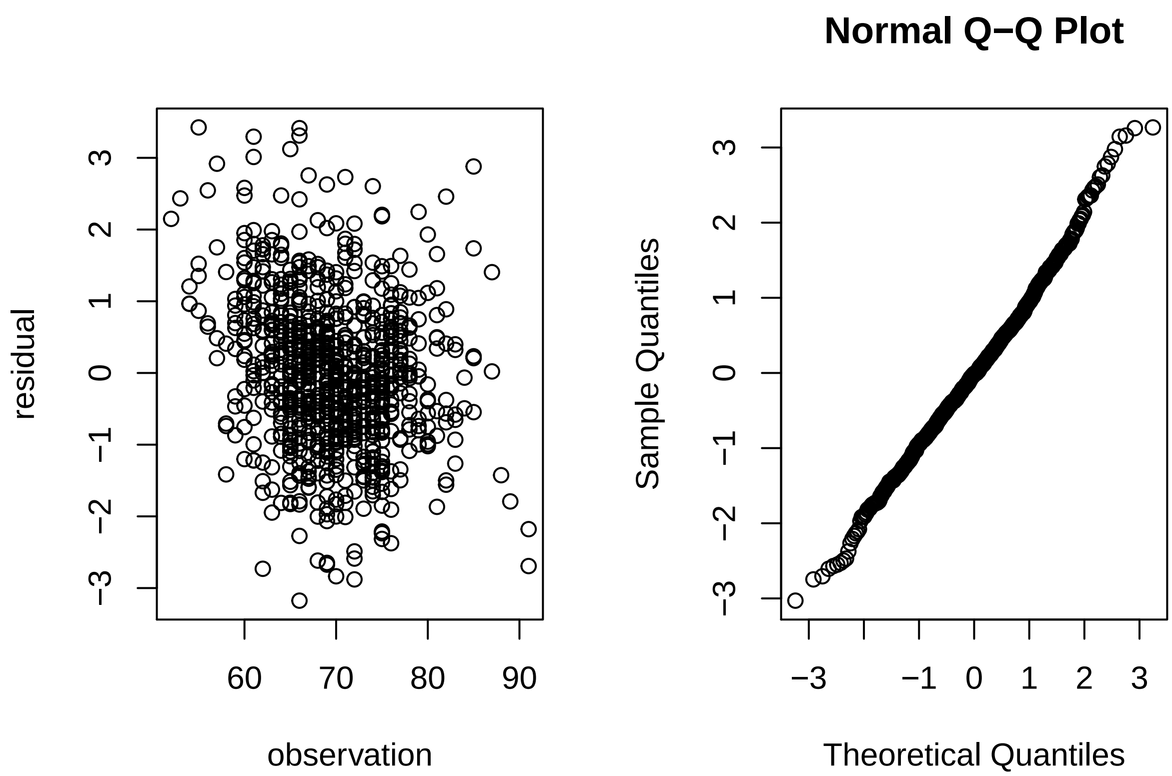
```
## [1] 3.909375
```

From the comparison of MSE among those three model above, we found out that the optimal gradient boosting regression tree model achieved the smallest MSE, so we are able to conclude that the optimal gradient boosting regression tree model is the best model for this case.

(d)

In this question, we are required to plot residual against overall score for the optimal gradient boosting regression tree:

```
par(mfrow=c(1,2)); plot(fifa2$Overall,fifa.res,xlab='observation',ylab='residual');
qqnorm(fifa.res/sd(fifa.res))
```



The residual VS observation plot shows no obvious pattern and most of the points concentrate between [-2, 2] which is actually a good result for modelling. Normal Q-Q plots also proves the same conclusion as almost all points stand on the line together.

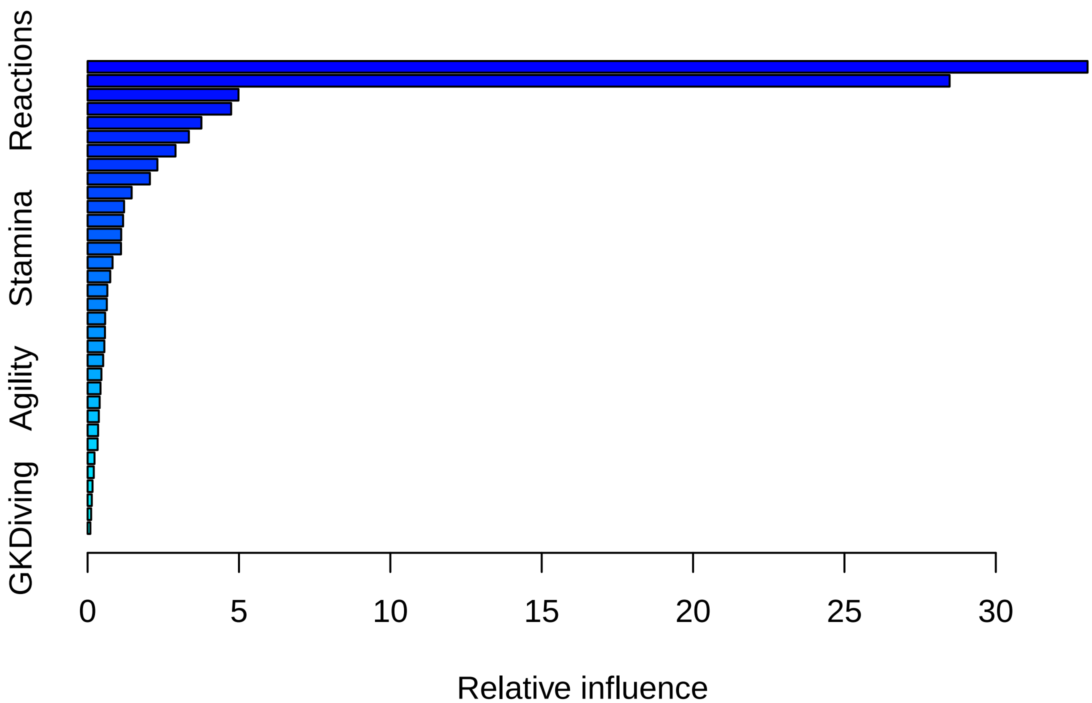
(e)

This question requires us to compare the relative variable importance of both my trees in (a) and (b).

```
#reltive variable importance of optimal regression tree
fifa2.cart$variable.importance/sum(fifa2.cart$variable.importance)
```

```
##      BallControl        Reactions        Dribbling     ShortPassing
##     0.2305382257     0.1369057188     0.1318139069     0.1261103201
##           Vision      LongPassing      Positioning  StandingTackle
##     0.1134453557     0.0944542590     0.0201117160     0.0196792053
##    Interceptions        LongShots        Composure          Marking
##     0.0169788065     0.0151097339     0.0146184851     0.0122423988
##     SlidingTackle        Finishing       Aggression        ShotPower
##     0.0117693760     0.0090674022     0.0085215933     0.0081859016
##          Stamina HeadingAccuracy         Crossing          Volleys
##     0.0055878179     0.0047447899     0.0040151015     0.0035855561
##         Strength        Penalties          Balance          Agility
##     0.0033779482     0.0021655553     0.0016112820     0.0011732980
##     Acceleration       FKAccuracy            Curve       GKHandling
##     0.0010376560     0.0009955272     0.0008877492     0.0004246820
##      SprintSpeed          Jumping          GKDiving
##     0.0003646797     0.0002660874     0.0002098648
```

```
#reltive variable importance of optimal gradient boosting regression tree
fifa.gbm.summary <- summary.gbm(fifa.gbm)
```



In order to compare them, we combine them together to compare:

```
ort.rvi <- as.matrix(fifa2.cart$variable.importance/sum(fifa2.cart$variable.importance) * 100)
gbm.rvi <- fifa.gbm.summary$rel.inf
compare.rvi = cbind(ort.rvi, gbm.rvi[match(rownames(ort.rvi), rownames(fifa.gbm.summary))])
colnames(compare.rvi) = c("regression tree", "gradient boosting tree")
compare.rvi
```

```
##                  regression tree gradient boosting tree
## BallControl          23.05382257            33.03098084
## Reactions            13.69057188            28.47428063
## Dribbling            13.18139069             1.20486159
## ShortPassing         12.61103201             3.75838490
## Vision               11.34453557             0.33137879
## LongPassing           9.44542590             0.57950165
## Positioning           2.01117160             3.34531457
## StandingTackle        1.96792053             2.30487147
## Interceptions         1.69788065             1.17245649
## LongShots             1.51097339             0.63527299
## Composure             1.46184851             4.98385002
## Marking               1.22423988             0.82443586
## SlidingTackle         1.17693760             1.45527133
## Finishing             0.90674022             2.90463613
## Aggression            0.85215933             1.10960971
## ShotPower             0.81859016             4.74214808
## Stamina               0.55878179             1.10378380
## HeadingAccuracy       0.47447899             2.05773145
## Crossing              0.40151015             0.34733525
## Volleys               0.35855561             0.55332566
## Strength              0.33779482             0.37048659
## Penalties             0.21655553             0.39807456
## Balance               0.16112820             0.51466982
```

```
## Agility                0.11732980             0.42650598
## Acceleration           0.10376560             0.57380160
## FKAccuracy             0.09955272             0.45623188
## Curve                  0.08877492             0.22790807
## GKHandling             0.04246820             0.13829641
## SprintSpeed            0.03646797             0.74744805
## Jumping                0.02660874             0.65216298
## GKDiving               0.02098648             0.09076354
```

To find out which variables have similar importance score, we allow them to have 10% difference in this case:

```
compare.rvi[0.9< compare.rvi[,1]/compare.rvi[,2] & compare.rvi[,1]/compare.rvi[,2]<1.11, ]
```

```
##          regression tree gradient boosting tree
##                0.3377948               0.3704866
```

In this case we have "Strength" has similar results between tree models in (a) and (b). Let's loose the boundary of similarity:

```
compare.rvi[0.75< compare.rvi[,1]/compare.rvi[,2] & compare.rvi[,1]/compare.rvi[,2]<1.33, ]
```

```
##                  regression tree gradient boosting tree
## StandingTackle         1.9679205              2.3048715
## SlidingTackle          1.1769376              1.4552713
## Aggression             0.8521593              1.1096097
## Crossing               0.4015102              0.3473353
## Strength               0.3377948              0.3704866
```

Now we have four more! "StandingTackle", "SlidingTackle", "Aggression", "Crossing" and "Strength". So we can conclude that these five are roughly equally important.

Now it comes with the question: why all variables are NOT equally important? To answer this question, we need to go back to what actually determines relative variable importances - it depends on how many times this variable has been chosen for splitting the tree, which means a variable achieves higher importance score only because it has been chosen for more time than others. Rather than just calculate the times when the variable got chosen for splitting for optimal regression trees, optimal gradient boosting regression tree sums its importance of each trees separately. The sum will be divided by the total number of trees, which makes it different from normal regression trees, so the difference comes out.

Question 3

(a)

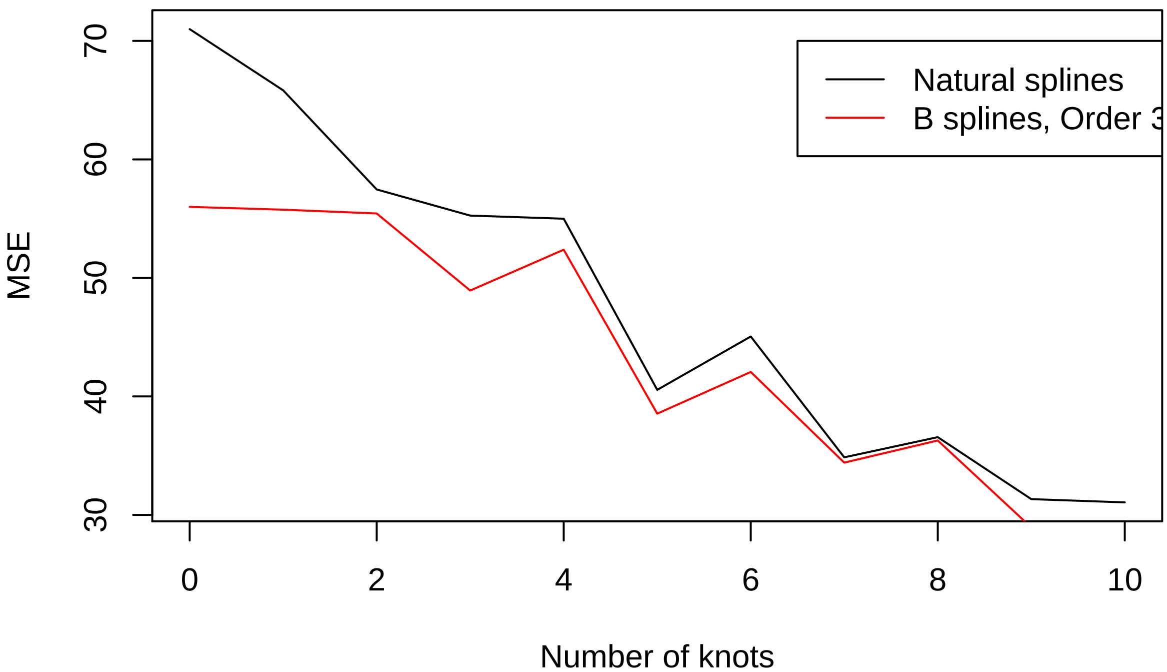
First, we read the data and give indexes to the rows:

```
accident <- read.csv("~/Desktop/STATS 762/airliner_accidents-1.csv")
accident$index <- 1:72
```

The goal is to model the fatal accidents with year using both natural splines and B-splines to fit. We fit the temperature data using natrual splines and B-splines with the degree of 3. Various number of inner knots are considered, 0-10.

```
set.seed(1e5)
acci.mse.ns=acci.mse.bs=c(0:10)
n.knots=c(0:10)
for(j in 1:length(acci.mse.ns)){
```

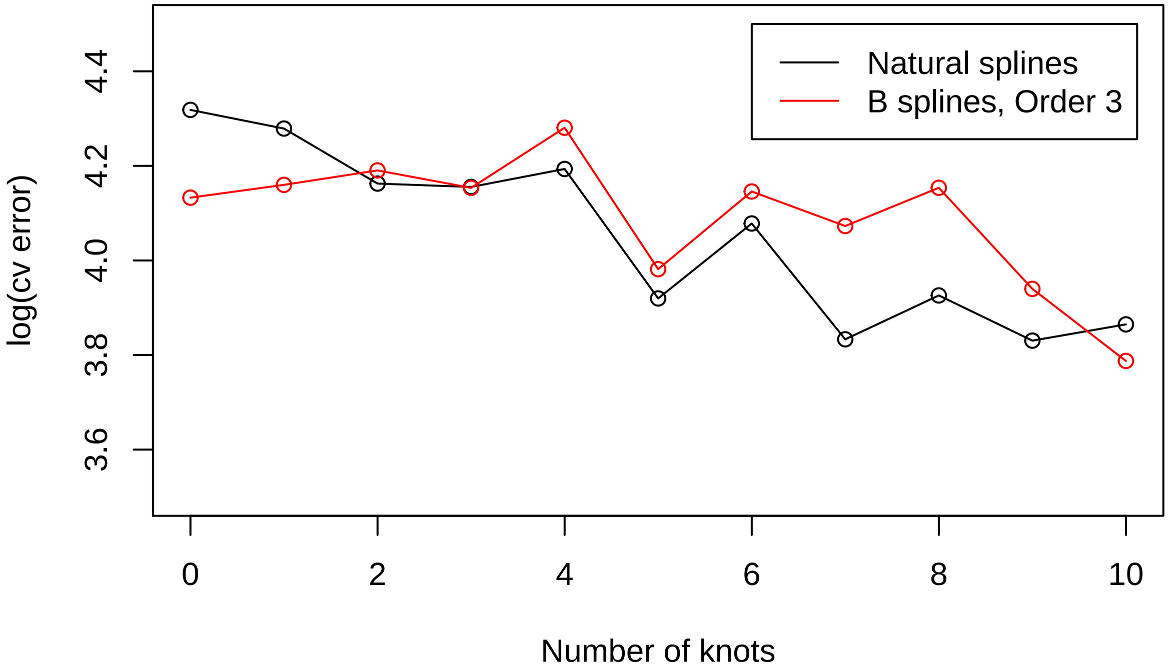
```
  #natural splines
  a.ns=ns(accident$index,df=n.knots[j]+1,intercept=FALSE)
  #B splines
  a.bs=bs(accident$index,df=n.knots[j]+3,intercept=FALSE)
  #predict accidents
  pre.acci.ns=predict(lm(accident$Fatal~a.ns), interval='confidence');
  pre.acci.bs=predict(lm(accident$Fatal~a.bs), interval='confidence');
  #MSE
  acci.mse.ns[j]=mean((accident$Fatal-pre.acci.ns[,1])^2)
  acci.mse.bs[j]=mean((accident$Fatal-pre.acci.bs[,1])^2)
}

plot(n.knots,acci.mse.ns,'l',ylab='MSE',xlab='Number of knots'); lines(n.knots,acci.mse.bs,col='red');
legend(6.5,70,legend=c("Natural splines","B splines, Order 3"),col=c(1,2),lty=1);
```



LOOCV error -vs- number of knots:

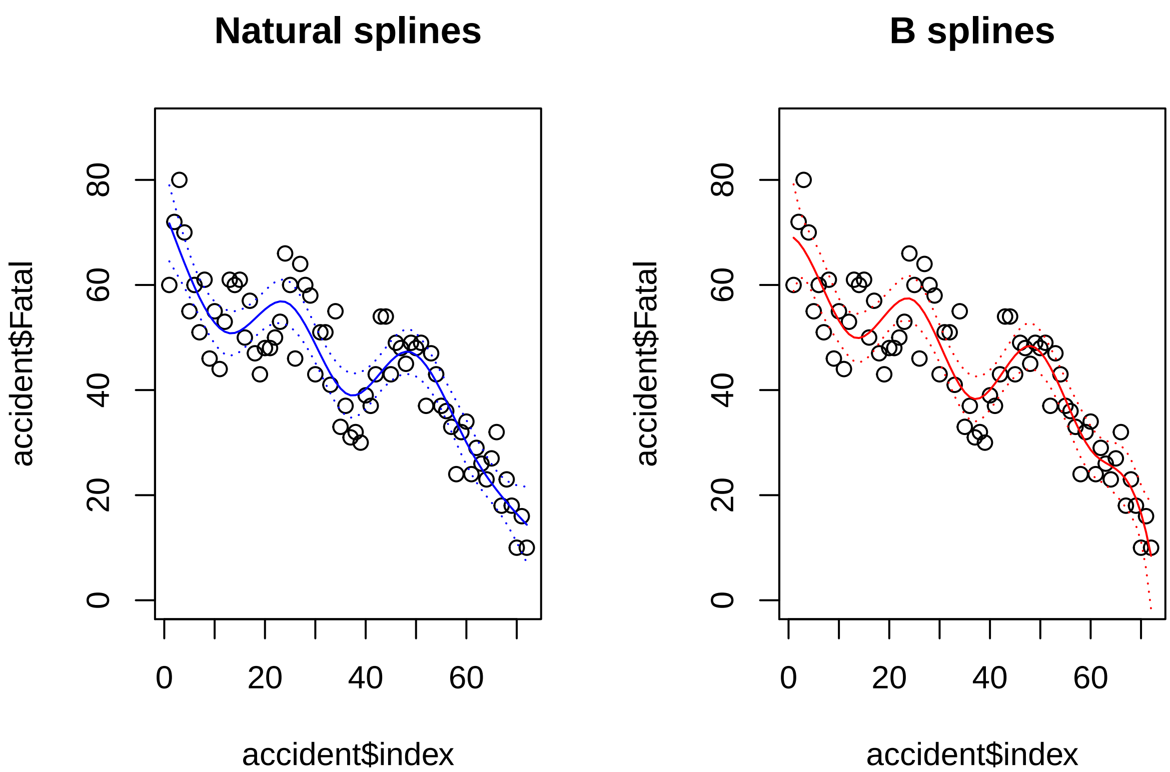
```
#LOOCV error -vs- number of knots
acci.cv.ns=acci.cv.bs=rep(0,length(n.knots))
for(j in 1:length(n.knots)){ for(l in 1:length(accident$index)){
  #predict accidents
  pre.a.ns=predict(lm(Fatal~ns(index,df=n.knots[j]+1,
                               intercept=FALSE,Boundary.knots=c(1,72)),
                              data=accident[-l,]),newdata=accident[l,])
  pre.a.bs=predict(lm(Fatal~bs(index,df=n.knots[j]+3,intercept=FALSE,
                               Boundary.knots=c(1,72)),data=accident[-l,]),
                              newdata=accident[l,])
  #cumulative sum of error
  acci.cv.ns[j]=acci.cv.ns[j]+(accident$Fatal[l]-pre.a.ns)^2
  acci.cv.bs[j]=acci.cv.bs[j]+(accident$Fatal[l]-pre.a.bs)^2
}}
acci.cv.ns=acci.cv.ns/length(accident$index)
acci.cv.bs=acci.cv.bs/length(accident$index)
```

```
plot(n.knots,log(acci.cv.ns),ylab='log(cv error)',xlab='Number of knots','o',ylim=c(3.5,4.5)); lines(n.k
legend(6,4.5,legend=c("Natural splines","B splines, Order 3"),col=c(1,2), lty=1);
```



It is not hard to figure out that when the natural splines has the smallest error when number of knots equals 7 and B splines has the smallest errir when number of knots equals to 10.

```
acci0.ns=predict(lm(Fatal~ns(index,df=5+1,intercept=FALSE),data=accident),
                 newdata=accident,interval = 'confidence')
acci0.bs=predict(lm(Fatal~bs(index,df=5+3,intercept=FALSE),data=accident),
                 newdata=accident,interval = 'confidence')
par(mfrow=c(1,2));
plot(accident$index,accident$Fatal,ylim=c(0,90),main='Natural splines');
matlines(accident$index,acci0.ns,lty=c(1,3,3),col=c(4,4,4));
plot(accident$index,accident$Fatal,ylim=c(0,90),main='B splines');
matlines(accident$index,acci0.bs,lty=c(1,3,3),col=c(2,2,2));
```



We can conclude that B splines achieves a better result than natural splines, it has smaller LOOCVSE and MSE.

(b)

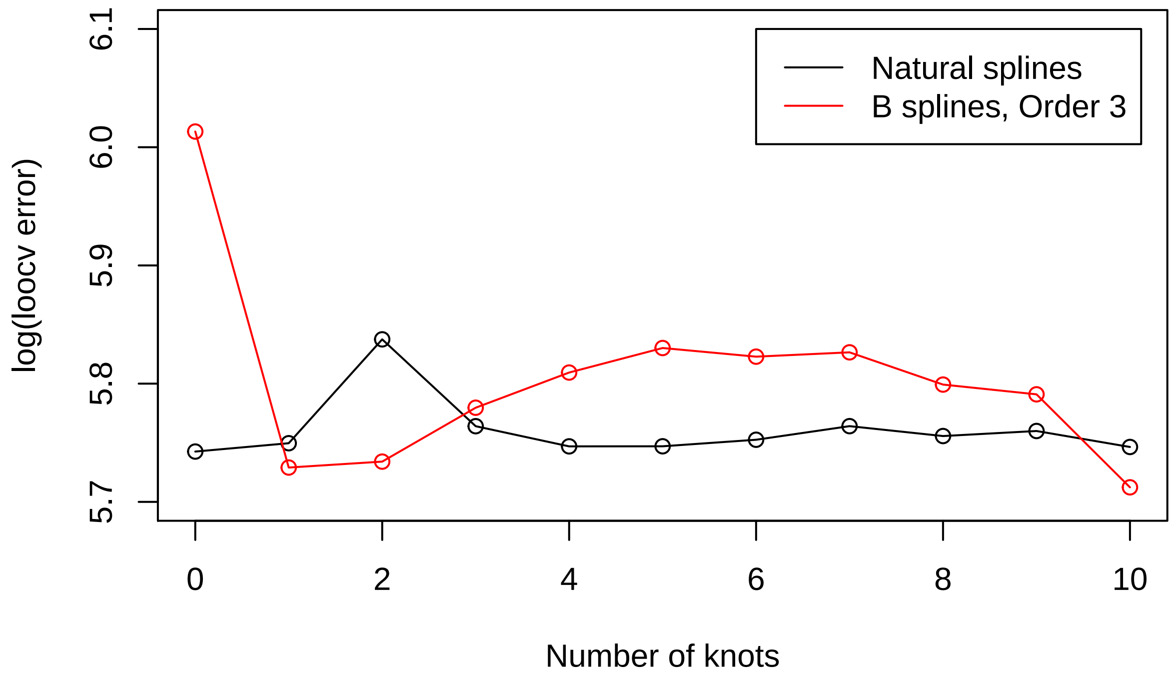
The goal is to model the hijacking insidents with fatal accidents using both natural splines and B-splines to fit. We fit the temperature data using natrual splines and B-splines with the degree of 3. Various number of inner knots are considered, 0-10.

```
# dataset has been sorted based on the fatal accidents
accident<-accident[sort(accident$Fatal,index=TRUE)$ix,]
```

LOOCV for natrual splines and B splines

```
accident.cv.ns1=accident.cv.bs1=rep(0,length(n.knots))
for(j in 1:length(n.knots)){ for(l in 1:length(accident$Fatal)){
  #predict hijacking insidents
  a.ns.pre1=predict(glm(Hijacking~ns(Fatal,df=n.knots[j]+1,
                                    intercept=FALSE,Boundary.knots=c(10,80)),
                                   data=accident[-l,], family = poisson),
                                   newdata=accident[l,],type = "response")
  a.bs.pre1=predict(glm(Hijacking~bs(Fatal,df=n.knots[j]+3,
                                    intercept=FALSE,Boundary.knots=c(10,80)),
                                   data=accident[-l,],family = poisson),
                                   newdata=accident[l,],type = "response")
  #cumulative sum of error
  accident.cv.ns1[j]=accident.cv.ns1[j]+(accident$Hijacking[l]-a.ns.pre1)^2
  accident.cv.bs1[j]=accident.cv.bs1[j]+(accident$Hijacking[l]-a.bs.pre1)^2
}}
accident.cv.ns1=accident.cv.ns1/length(accident$Fatal)
```

```
accident.cv.bs1=accident.cv.bs1/length(accident$Fatal)

plot(n.knots,log(accident.cv.ns1),ylab='log(loocv error)',
     xlab='Number of knots','o',ylim = c(5.7, 6.1))
lines(n.knots,log(accident.cv.bs1),col='red','o')
legend(6,6.1,legend=c("Natural splines","B splines, Order 3"),
       col=c(1,2), lty=1)
```

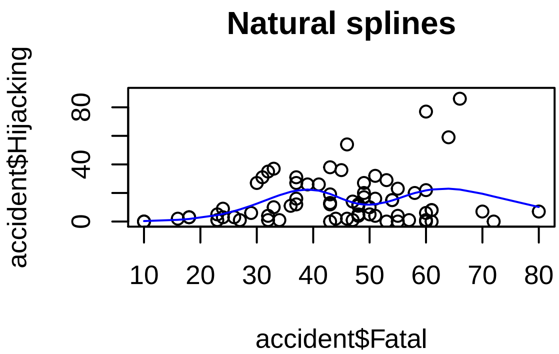


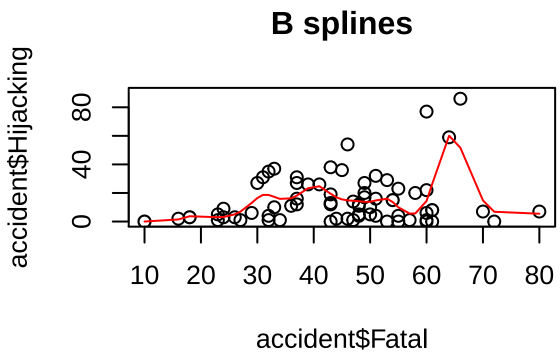
It is not hard to figure out that when the natural splines has the smallest error when number of knots equals 4 and B splines has the smallest errir when number of knots equals to 10.

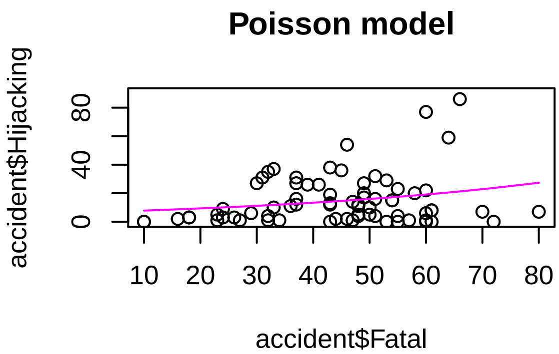
Rather than fitting only natual splines and B splines, we also fit a possion regression based on that.

```
n.knots.ns <- 4
n.knots.bs <- 10
# Natrual splines
acci0.ns1=predict(glm(Hijacking~ns(Fatal,df = n.knots.ns+1,intercept=FALSE),
                      data=accident,family = poisson),
                      type = "response", newdata=accident,
                      interval = 'confidence')
# B splines
acci0.bs1=predict(glm(Hijacking~bs(Fatal,df = n.knots.bs+3,intercept=FALSE),
                      data=accident, family = poisson),
                      type = "response", newdata=accident,
                      interval = 'confidence')
# Possion model
acci0.p=predict(glm(Hijacking~Fatal,data=accident,family = poisson),
                      type = "response",interval = 'confidence')
par(mfrow=c(2,2))
plot(accident$Fatal,accident$Hijacking, ylim = c(0,90),
     main = "Natural splines")
matlines(accident$Fatal,acci0.ns1,lty=c(1,3,3),col=c(4,4,4))
plot(accident$Fatal,accident$Hijacking, ylim = c(0,90),
```

```r
     main = "B splines")
matlines(accident$Fatal,acci0.bs1,lty=c(1,3,3),col=c(2,2,2))
plot(accident$Fatal,accident$Hijacking, ylim = c(0,90),
     main = "Poisson model")
matlines(accident$Fatal,acci0.p,lty=c(1,3,3),col=c(6,6,6))
```







We can conclude that B splines achieves a better result than natural splines, it has smaller LOOCVSE and MSE, the same as question 2(b).

Description of the relationship of those incidents: it is obvious that most of the large Hijacking points concentrates between 25 and 70, Hijacking points are relatively very small in [10,25] and [70,80]. There are some very high Hijacking points locating between 60 and 70. Also, most of the Hijacking points locate between [30,60] fatal interval. There can be a potential relationship between Hijacking and Fatal as in [30,60] of Fatal, Hijacking keeps increasing.