ml_final_project'

Ruchen Cai

3/16/2022

import and clean the data

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.2     v dplyr   1.0.6
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
load(file='nhanes2003-2004.rda')

# remove participants younger than 50 yo
NHANES.data <- subset(nhanes2003_2004, as.numeric(nhanes2003_2004$RIDAGEEX) > (50*12-1))

# make a matrix that contains only important predictors, excluding participants with missing data
variables <- c("RIDAGEYR", "RIAGENDR", "BPQ010", "BPQ060", "DIQ010", "DIQ050", "DIQ090", "MCQ0

NHANES.data <- na.omit(NHANES.data[variables])

# change all values to type numeric
for(i in 1:length(NHANES.data[1,])){
  NHANES.data[,i] <- as.numeric(NHANES.data[,i])
}
```

```
summary(NHANES.data)
```

```
##     RIDAGEYR        RIAGENDR         BPQ010          BPQ060
##  Min.   :48.00   Min.   :1.000   Min.   :1.000   Min.   :1.000
##  1st Qu.:58.00   1st Qu.:1.000   1st Qu.:1.000   1st Qu.:1.000
##  Median :64.00   Median :1.000   Median :1.000   Median :1.000
##  Mean   :64.25   Mean   :1.494   Mean   :1.267   Mean   :1.136
##  3rd Qu.:72.00   3rd Qu.:2.000   3rd Qu.:1.000   3rd Qu.:1.000
##  Max.   :83.00   Max.   :2.000   Max.   :5.000   Max.   :3.000
##      DIQ010          DIQ050          DIQ090          MCQ010
##  Min.   :1.000   Min.   :1.000   Min.   :1.000   Min.   :1.000
##  1st Qu.:2.000   1st Qu.:2.000   1st Qu.:2.000   1st Qu.:2.000
##  Median :2.000   Median :2.000   Median :2.000   Median :2.000
##  Mean   :1.826   Mean   :1.956   Mean   :1.961   Mean   :1.899
```

```
##  3rd Qu.:2.000   3rd Qu.:2.000   3rd Qu.:2.000   3rd Qu.:2.000
##  Max.   :3.000   Max.   :2.000   Max.   :3.000   Max.   :3.000
##      MCQ053          MCQ160A         MCQ160B         MCQ160K
##  Min.   :1.000   Min.   :1.000   Min.   :1.000   Min.   :1.000
##  1st Qu.:2.000   1st Qu.:1.000   1st Qu.:2.000   1st Qu.:2.000
##  Median :2.000   Median :2.000   Median :2.000   Median :2.000
##  Mean   :1.968   Mean   :1.547   Mean   :1.952   Mean   :1.921
##  3rd Qu.:2.000   3rd Qu.:2.000   3rd Qu.:2.000   3rd Qu.:2.000
##  Max.   :4.000   Max.   :3.000   Max.   :3.000   Max.   :3.000
##     MCQ160L         BMXWAIST          MCQ160M          MCQ220
##  Min.   :1.000   Min.   :  1.00   Min.   :1.000   Min.   :1.000
##  1st Qu.:2.000   1st Qu.: 93.25   1st Qu.:2.000   1st Qu.:2.000
##  Median :2.000   Median :308.00   Median :2.000   Median :2.000
##  Mean   :1.955   Mean   :478.11   Mean   :1.861   Mean   :1.848
##  3rd Qu.:2.000   3rd Qu.:895.00   3rd Qu.:2.000   3rd Qu.:2.000
##  Max.   :3.000   Max.   :981.00   Max.   :3.000   Max.   :3.000
##     MCQ245A         MCQ250A         MCQ250B         MCQ250C
##  Min.   :1.000   Min.   :1.000   Min.   :1.000   Min.   :1.000
##  1st Qu.:1.000   1st Qu.:1.000   1st Qu.:2.000   1st Qu.:2.000
##  Median :2.000   Median :2.000   Median :2.000   Median :2.000
##  Mean   :1.615   Mean   :1.523   Mean   :1.869   Mean   :1.836
##  3rd Qu.:2.000   3rd Qu.:2.000   3rd Qu.:2.000   3rd Qu.:2.000
##  Max.   :2.000   Max.   :3.000   Max.   :3.000   Max.   :3.000
##     MCQ250E         MCQ250F         MCQ250G          MCQ265
##  Min.   :1.000   Min.   :1.000   Min.   :1.000   Min.   :1.000
##  1st Qu.:2.000   1st Qu.:2.000   1st Qu.:2.000   1st Qu.:2.000
##  Median :2.000   Median :2.000   Median :2.000   Median :2.000
##  Mean   :1.877   Mean   :1.804   Mean   :1.903   Mean   :1.908
##  3rd Qu.:2.000   3rd Qu.:2.000   3rd Qu.:2.000   3rd Qu.:2.000
##  Max.   :3.000   Max.   :3.000   Max.   :3.000   Max.   :4.000
##      SSQ011          SSQ051          WHQ030          WHQ040
##  Min.   :1.000   Min.   :1.000   Min.   :1.000   Min.   :1.000
##  1st Qu.:1.000   1st Qu.:1.000   1st Qu.:1.000   1st Qu.:2.000
##  Median :1.000   Median :1.000   Median :1.000   Median :2.000
##  Mean   :1.093   Mean   :1.307   Mean   :1.779   Mean   :2.301
##  3rd Qu.:1.000   3rd Qu.:2.000   3rd Qu.:3.000   3rd Qu.:3.000
##  Max.   :4.000   Max.   :4.000   Max.   :5.000   Max.   :5.000
##      LBXRDW          HSD010          BPXPULS           BPXML1
##  Min.   : 5.00   Min.   :1.000   Min.   :1.000   Min.   : 2.000
##  1st Qu.:18.00   1st Qu.:2.000   1st Qu.:1.000   1st Qu.: 5.000
##  Median :22.00   Median :3.000   Median :1.000   Median : 6.000
##  Mean   :24.55   Mean   :2.923   Mean   :1.085   Mean   : 6.559
##  3rd Qu.:28.00   3rd Qu.:4.000   3rd Qu.:1.000   3rd Qu.: 8.000
##  Max.   :97.00   Max.   :5.000   Max.   :2.000   Max.   :14.000
##      VIQ200          BMXBMI         BPXSY1          BPXDI1
##  Min.   :1.000   Min.   : 119   Min.   : 1.00   Min.   : 1.00
##  1st Qu.:2.000   1st Qu.:1135   1st Qu.:12.00   1st Qu.:37.00
##  Median :2.000   Median :1424   Median :18.00   Median :40.00
##  Mean   :1.848   Mean   :1473   Mean   :20.64   Mean   :39.71
##  3rd Qu.:2.000   3rd Qu.:1800   3rd Qu.:26.00   3rd Qu.:44.00
##  Max.   :3.000   Max.   :2605   Max.   :77.00   Max.   :55.00
##     mortstat
##  Min.   :0.0000
##  1st Qu.:0.0000
```

```
##   Median :0.0000
##   Mean   :0.1796
##   3rd Qu.:0.0000
##   Max.   :1.0000
```

split the dataset into training and testing set

```r
library(caTools)
set.seed(209123)

train <- sample.split(NHANES.data$RIDAGEYR, SplitRatio = 0.7)
NHANES.training <- subset(NHANES.data,train==TRUE)
NHANES.testing <- subset(NHANES.data,train==FALSE)

testing.mortstat <- NHANES.testing$mortstat
```

Model1: logistic regression

```r
glm.fits <- glm(mortstat ~ ., NHANES.training, family=binomial)
glm.probs <- predict(glm.fits, NHANES.testing, type = "response")
glm.pred <- rep(0,410)
glm.pred[glm.probs > .5] <- 1
```

```r
summary(glm.fits)
```

```
##
## Call:
## glm(formula = mortstat ~ ., family = binomial, data = NHANES.training)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.7588  -0.6187  -0.3570  -0.1779   2.5864
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -4.4840607  2.5485249  -1.759  0.07850 .
## RIDAGEYR     0.0826033  0.0140330   5.886 3.95e-09 ***
## RIAGENDR    -0.5584385  0.2131970  -2.619  0.00881 **
## BPQ010       0.0666546  0.1578904   0.422  0.67291
## BPQ060      -0.3399005  0.2699451  -1.259  0.20798
## DIQ010      -0.3104766  0.2279732  -1.362  0.17323
## DIQ050      -0.4984604  0.4032705  -1.236  0.21644
## DIQ090      -0.4797651  0.4372214  -1.097  0.27251
## MCQ010       0.3813123  0.3433004   1.111  0.26669
## MCQ053       0.1093301  0.4560970   0.240  0.81056
## MCQ160A     -0.2799572  0.2037979  -1.374  0.16953
## MCQ160B     -0.6180269  0.3094232  -1.997  0.04579 *
## MCQ160K     -0.1882452  0.3461293  -0.544  0.58654
## MCQ160L     -0.3227434  0.4628207  -0.697  0.48559
## BMXWAIST     0.0002118  0.0003111   0.681  0.49610
## MCQ160M      0.2871458  0.2893780   0.992  0.32106
## MCQ220      -0.1515776  0.2455244  -0.617  0.53700
## MCQ245A      0.2517161  0.2449428   1.028  0.30411
```

```
## MCQ250A      0.1605086  0.1906151   0.842  0.39976
## MCQ250B     -0.1213568  0.2533617  -0.479  0.63195
## MCQ250C     -0.2622214  0.2344844  -1.118  0.26344
## MCQ250E     -0.0956645  0.2484308  -0.385  0.70018
## MCQ250F      0.2430167  0.2156945   1.127  0.25988
## MCQ250G     -0.4689425  0.2637078  -1.778  0.07536 .
## MCQ265      -0.2208634  0.2037583  -1.084  0.27839
## SSQ011      -0.3263575  0.3006307  -1.086  0.27767
## SSQ051      -0.0063933  0.1638842  -0.039  0.96888
## WHQ030       0.2124070  0.1517947   1.399  0.16172
## WHQ040      -0.0547762  0.2356038  -0.232  0.81616
## LBXRDW       0.0347990  0.0077760   4.475 7.63e-06 ***
## HSD010       0.4345622  0.1045889   4.155 3.25e-05 ***
## BPXPULS      0.1183853  0.2976498   0.398  0.69083
## BPXML1       0.0642645  0.0635249   1.012  0.31171
## VIQ200       0.2628171  0.2490539   1.055  0.29131
## BMXBMI       0.0003109  0.0002749   1.131  0.25808
## BPXSY1      -0.0050710  0.0109207  -0.464  0.64240
## BPXDI1      -0.0035235  0.0118823  -0.297  0.76682
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 932.39  on 959  degrees of freedom
## Residual deviance: 729.25  on 923  degrees of freedom
## AIC: 803.25
##
## Number of Fisher Scoring iterations: 5
```

According to logistic regression, RIDAGEYR, RIAGENDR, MCQ160B, LBXRDW, and BPXPULS significantly affect the results.

```r
# test for sensitivity and specificity
conf.logistic <- table(glm.pred,testing.mortstat)
conf.logistic
```

```
##         testing.mortstat
## glm.pred   0   1
##        0 337  55
##        1   9   9
```

```r
sens.logistic <- conf.logistic[1,1]/(conf.logistic[1,1]+conf.logistic[1,2])
sens.logistic
```

```
## [1] 0.8596939
```

```r
spec.logistic <- conf.logistic[2,2]/(conf.logistic[2,1]+conf.logistic[2,2])
spec.logistic
```

```
## [1] 0.5
```

```r
# testing error rate
err.logistic <- mean(glm.pred!=testing.mortstat)
err.logistic
```

```
## [1] 0.1560976
```

The sensiticity of logistic regression model is 85.97%. The specificity of logistic regression model is 50.00%. The testing error rate of logistic regression model is 15.61%

Model2: LDA

```
library(MASS)
```

```
##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select
```

```
lda.fits <- lda(mortstat ~ ., data = NHANES.training)
lda.pred <- predict(lda.fits, NHANES.testing)
lda.class <- lda.pred$class
```

```
# test for sensitivity and specificity
conf.lda <- table(lda.class,testing.mortstat)
conf.lda
```

```
##          testing.mortstat
## lda.class   0   1
##         0 340  56
##         1   6   8
```

```
sens.lda <- conf.lda[1,1]/(conf.lda[1,1]+conf.lda[1,2])
sens.lda
```

```
## [1] 0.8585859
```

```
spec.lda <- conf.lda[2,2]/(conf.lda[2,1]+conf.lda[2,2])
spec.lda
```

```
## [1] 0.5714286
```

```
# testing error rate
testing.err.lda <- mean(testing.mortstat!=lda.class)
testing.err.lda
```

```
## [1] 0.1512195
```

The sensiticity of LDA model is 85.86%. The specificity of LDA model is 57.14%. The testing error rate of LDA model is 15.12%

Model3: QDA

```
qda.fits <- qda(mortstat ~ ., data = NHANES.training)
qda.pred <- predict(qda.fits, NHANES.testing)
qda.class <- qda.pred$class
```

```
# test for sensitivity and specificity
conf.qda <- table(qda.class,testing.mortstat)
conf.qda
```

```
##          testing.mortstat
## qda.class   0   1
##         0 311  46
```

```
##         1  35  18
```

```
sens.qda <- conf.qda[1,1]/(conf.qda[1,1]+conf.qda[1,2])
sens.qda
```

```
## [1] 0.8711485
```

```
spec.qda <- conf.qda[2,2]/(conf.qda[2,1]+conf.qda[2,2])
spec.qda
```

```
## [1] 0.3396226
```

```
# testing error rate
testing.err.qda <- mean(testing.mortstat!=qda.class)
testing.err.qda
```

```
## [1] 0.197561
```

The sensiticity of QDA model is 87.11%. The specificity of QDA model is 33.96%. The testing error rate of QDA model is 19.76%

Model4: Lasso

```
library(glmnet)
```

```
## Loading required package: Matrix
```

```
##
## Attaching package: 'Matrix'
```

```
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
```

```
## Loaded glmnet 4.1-3
```

```
x.train <- model.matrix(mortstat ~., data = NHANES.training)[,-1]
y.train <- NHANES.training$mortstat

x.test <- model.matrix(mortstat ~ ., data = NHANES.testing)[,-1]
y.test <- NHANES.testing$mortstat

#grid <- 10^seq(10,-2,length=100)

cv.model <- cv.glmnet(x.train,y.train,alpha=1)
best.lambda.lasso <- cv.model$lambda.min
```

```
lasso.mod <- glmnet(x.train, y.train,alpha=1,lambda=best.lambda.lasso)
lasso.val <- as.numeric(predict(lasso.mod, x.test,s=best.lambda.lasso,type="class"))
lasso.pred <- rep(0,410)
lasso.pred[lasso.val > .5] <- 1
```

```
# test for sensitivity and specificity
 conf.lasso <- table(lasso.pred,y.test)
 conf.lasso
```

```
##           y.test
## lasso.pred   0   1
##          0 345  62
##          1   1   2
```

```r
sens.lasso <- conf.lasso[1,1]/(conf.lasso[1,1]+conf.lasso[1,2])
sens.lasso
```

```
## [1] 0.8476658
```

```r
spec.lasso <- conf.lasso[2,2]/(conf.lasso[2,1]+conf.lasso[2,2])
spec.lasso
```

```
## [1] 0.6666667
```

```r
# MSE
err.lasso <- mean(lasso.pred!=y.test)
err.lasso
```

```
## [1] 0.1536585
```

The sensiticity of QDA model is 84.77%. The specificity of QDA model is 66.67%. The testing error rate of QDA model is 15.37%

Model 5-7: SVM

```r
library(e1071)
set.seed(283)

# linear kernel
tune.out.linear <- tune(svm, mortstat~., data=NHANES.training, kernel="linear",ranges = list(cost=c(0.0

summary(tune.out.linear)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##  1000
##
## - best performance: 0.1743665
##
## - Detailed performance results:
##    cost     error dispersion
## 1 1e-03 0.1761247 0.02178138
## 2 1e-02 0.1758409 0.02225474
## 3 1e-01 0.1757542 0.02238576
## 4 1e+00 0.1757910 0.02241351
## 5 1e+01 0.1757957 0.02240054
## 6 1e+02 0.1756740 0.02248178
## 7 1e+03 0.1743665 0.02212332
```

```r
tune.out.linear$best.model
```

```
##
## Call:
## best.tune(method = svm, train.x = mortstat ~ ., data = NHANES.training,
##     ranges = list(cost = c(0.001, 0.01, 0.1, 1, 10, 100, 1000)),
##     kernel = "linear")
```

```
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  linear
##        cost:  1000
##       gamma:  0.02777778
##     epsilon:  0.1
##
##
## Number of Support Vectors:  499
```

```r
# svm, linear, cost = 1000
svm.linear <- svm(mortstat ~., data=NHANES.training, kernel ="linear",cost=1000,type="C-classification")
svm.linear.pred <- predict(svm.linear, NHANES.testing)

# test for sensitivity and specificity
conf.svm.lin <- table(svm.linear.pred,testing.mortstat)
conf.svm.lin
```

```
##                testing.mortstat
## svm.linear.pred   0   1
##               0 344  62
##               1   2   2
```

```r
sens.svm.lin <- conf.svm.lin[1,1]/(conf.svm.lin[1,1]+conf.svm.lin[1,2])
sens.svm.lin
```

```
## [1] 0.8472906
```

```r
spec.svm.lin <- conf.svm.lin[2,2]/(conf.svm.lin[2,1]+conf.svm.lin[2,2])
spec.svm.lin
```

```
## [1] 0.5
```

```r
# testing error rate
err.svm.lin <- mean((svm.linear.pred!=testing.mortstat))
err.svm.lin
```

```
## [1] 0.1560976
```

The sensiticity of SVM model using linear kernel is 84.73%. The specificity of SVM model using linear kernel is 50.00%. The testing error of SVM model using linear kernel is 15.61%

```r
# radial kernel
set.seed(32989)
tune.out.radial <- tune(svm, mortstat~., data=NHANES.training, kernel ="radial", ranges =list(cost=c(0.0

summary(tune.out.radial)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##    10     3
##
```

```
## - best performance: 0.1546424
##
## - Detailed performance results:
##     cost gamma     error dispersion
## 1  1e-03   0.5 0.1762250 0.04128445
## 2  1e-02   0.5 0.1760117 0.04123115
## 3  1e-01   0.5 0.1736311 0.04058846
## 4  1e+00   0.5 0.1574270 0.03356227
## 5  1e+01   0.5 0.1546898 0.02555857
## 6  1e+02   0.5 0.1546898 0.02555857
## 7  1e+03   0.5 0.1546898 0.02555857
## 8  1e-03   1.0 0.1762251 0.04128443
## 9  1e-02   1.0 0.1760122 0.04123087
## 10 1e-01   1.0 0.1736418 0.04058795
## 11 1e+00   1.0 0.1574586 0.03353923
## 12 1e+01   1.0 0.1546960 0.02543562
## 13 1e+02   1.0 0.1546960 0.02543562
## 14 1e+03   1.0 0.1546960 0.02543562
## 15 1e-03   2.0 0.1762251 0.04128453
## 16 1e-02   2.0 0.1760119 0.04123084
## 17 1e-01   2.0 0.1736399 0.04058787
## 18 1e+00   2.0 0.1574365 0.03352892
## 19 1e+01   2.0 0.1546469 0.02540100
## 20 1e+02   2.0 0.1546469 0.02540100
## 21 1e+03   2.0 0.1546469 0.02540100
## 22 1e-03   3.0 0.1762251 0.04128452
## 23 1e-02   3.0 0.1760119 0.04123084
## 24 1e-01   3.0 0.1736396 0.04058779
## 25 1e+00   3.0 0.1574341 0.03352662
## 26 1e+01   3.0 0.1546424 0.02539442
## 27 1e+02   3.0 0.1546424 0.02539442
## 28 1e+03   3.0 0.1546424 0.02539442
## 29 1e-03   4.0 0.1762251 0.04128452
## 30 1e-02   4.0 0.1760119 0.04123084
## 31 1e-01   4.0 0.1736396 0.04058777
## 32 1e+00   4.0 0.1574341 0.03352617
## 33 1e+01   4.0 0.1546424 0.02539304
## 34 1e+02   4.0 0.1546424 0.02539304
## 35 1e+03   4.0 0.1546424 0.02539304
```

```r
tune.out.radial$best.model
```

```
##
## Call:
## best.tune(method = svm, train.x = mortstat ~ ., data = NHANES.training,
##     ranges = list(cost = c(0.001, 0.01, 0.1, 1, 10, 100, 1000), gamma = c(0.5,
##         1, 2, 3, 4)), kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  radial
##        cost:  10
##       gamma:  3
##     epsilon:  0.1
```

```
##
##
## Number of Support Vectors:  960
```

```
set.seed(427)

# svm, radial, cost = 10, gamma=3
svm.radial <- svm(mortstat ~., data=NHANES.training, kernel ="radial",cost=10,gamma=3,type="C-classifica
svm.radial.pred <- predict(svm.radial, NHANES.testing)

# test for sensitivity and specificity
conf.svm.rad <- table(svm.radial.pred,testing.mortstat)
conf.svm.rad
```

```
##                  testing.mortstat
## svm.radial.pred   0   1
##                 0 346  64
##                 1   0   0
```

```
sens.svm.rad <- conf.svm.rad[1,1]/(conf.svm.rad[1,1]+conf.svm.rad[1,2])
sens.svm.rad
```

```
## [1] 0.8439024
```

```
spec.svm.rad <- conf.svm.rad[2,2]/(conf.svm.rad[2,1]+conf.svm.rad[2,2])
spec.svm.rad
```

```
## [1] NaN
```

```
# testing error rate
err.svm.rad <- mean((svm.radial.pred!=testing.mortstat))
err.svm.rad
```

```
## [1] 0.1560976
```

The sensiticity of SVM model using radial kernel is 84.39%. The specificity of SVM model using radial kernel is unknown since the model classify all cases as alive. The testing error of SVM model using radial kernel is 15.61%

```
# polynomial kernel
set.seed(2987)
tune.out.poly <- tune(svm, mortstat~., data=NHANES.training, , kernel ="polynomial", ranges = list(cost

summary(tune.out.poly)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost degree
##     1      3
##
## - best performance: 0.1605922
##
## - Detailed performance results:
##     cost degree     error dispersion
## 1  1e-03      1 0.1762409 0.03137452
```

```
## 2  1e-02      1 0.1761873 0.03141165
## 3  1e-01      1 0.1758970 0.03179227
## 4  1e+00      1 0.1754025 0.03251923
## 5  5e+00      1 0.1753910 0.03248814
## 6  1e+01      1 0.1753903 0.03250287
## 7  1e+02      1 0.1753745 0.03249319
## 8  1e-03      2 0.1761841 0.03136474
## 9  1e-02      2 0.1754535 0.03128604
## 10 1e-01      2 0.1694481 0.03049072
## 11 1e+00      2 0.1704060 0.02605340
## 12 5e+00      2 0.2321557 0.03014433
## 13 1e+01      2 0.2887254 0.04164993
## 14 1e+02      2 0.7398740 0.16085113
## 15 1e-03      3 0.1761005 0.03134433
## 16 1e-02      3 0.1745599 0.03106061
## 17 1e-01      3 0.1639244 0.02911744
## 18 1e+00      3 0.1605922 0.02318568
## 19 5e+00      3 0.1821862 0.02972759
## 20 1e+01      3 0.1996163 0.03122904
## 21 1e+02      3 0.2721005 0.06066044
## 22 1e-03      4 0.1760316 0.03132022
## 23 1e-02      4 0.1739593 0.03087355
## 24 1e-01      4 0.1633753 0.02895433
## 25 1e+00      4 0.1623101 0.02073873
## 26 5e+00      4 0.1698467 0.02643086
## 27 1e+01      4 0.1790407 0.03106939
## 28 1e+02      4 0.2106294 0.03532869
## 29 1e-03      5 0.1759473 0.03130311
## 30 1e-02      5 0.1733903 0.03080272
## 31 1e-01      5 0.1647294 0.02928668
## 32 1e+00      5 0.1631628 0.02242649
## 33 5e+00      5 0.1659554 0.02470952
## 34 1e+01      5 0.1735138 0.03044052
## 35 1e+02      5 0.2184457 0.08200631
```

```
tune.out.poly$best.model
```

```
##
## Call:
## best.tune(method = svm, train.x = mortstat ~ ., data = NHANES.training,
##     ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100), degree = c(1,
##         2, 3, 4, 5)), kernel = "polynomial")
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  polynomial
##        cost:  1
##      degree:  3
##       gamma:  0.02777778
##      coef.0:  0
##     epsilon:  0.1
##
##
## Number of Support Vectors:  680
```

```r
# svm, polynomial, cost = 1, degree=3
svm.poly <- svm(mortstat ~., data=NHANES.training, kernel ="polynomial",cost=1,degree=3,type="C-classifi
svm.poly.pred <- predict(svm.poly, NHANES.testing)

# test for sensitivity and specificity
conf.svm.poly <- table(svm.poly.pred,testing.mortstat)
conf.svm.poly
```

```
##              testing.mortstat
## svm.poly.pred   0   1
##             0 338  60
##             1   8   4
```

```r
sens.svm.poly <- conf.svm.poly[1,1]/(conf.svm.poly[1,1]+conf.svm.poly[1,2])
sens.svm.poly
```

```
## [1] 0.8492462
```

```r
spec.svm.poly <- conf.svm.poly[2,2]/(conf.svm.poly[2,1]+conf.svm.poly[2,2])
spec.svm.poly
```

```
## [1] 0.3333333
```

```r
# testing error rate
err.svm.poly <- mean((svm.poly.pred!=testing.mortstat))
err.svm.poly
```

```
## [1] 0.1658537
```

The sensiticity of SVM model using polynomial kernel is 84.92%. The specificity of SVM model using polynomial kernel is 33.33%. The testing error of SVM model using polynomial kernel is 16.59%

Model 8&9: Tree-based methods

bagging

```r
library(tree)
library(randomForest)
```

```
## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:dplyr':
##
##     combine

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```r
set.seed(7321)

bag.mod <- randomForest(mortstat~., NHANES.training, mtry=36, ntree=100)
```

```
## Warning in randomForest.default(m, y, ...): The response has five or fewer
```

```
## unique values. Are you sure you want to do regression?
```

```r
yhat.bag <- predict(bag.mod, NHANES.testing)

yhat.pred <- rep(0,410)
yhat.pred[yhat.bag > .5] <- 1
```

```r
# test for sensitivity and specificity
conf.bag <- table(yhat.pred,testing.mortstat)
conf.bag
```

```
##          testing.mortstat
## yhat.pred   0   1
##         0 333  47
##         1  13  17
```

```r
sens.bag <- conf.bag[1,1]/(conf.bag[1,1]+conf.bag[1,2])
sens.bag
```

```
## [1] 0.8763158
```

```r
spec.bag <- conf.bag[2,2]/(conf.bag[2,1]+conf.bag[2,2])
spec.bag
```

```
## [1] 0.5666667
```

```r
# testing error rate
err.bag <- mean(yhat.pred!=testing.mortstat)
err.bag
```

```
## [1] 0.1463415
```

The sensiticity using bagging is 87.63%. The specificity using bagging is 56.67%. The testing error using bagging is 14.63%.

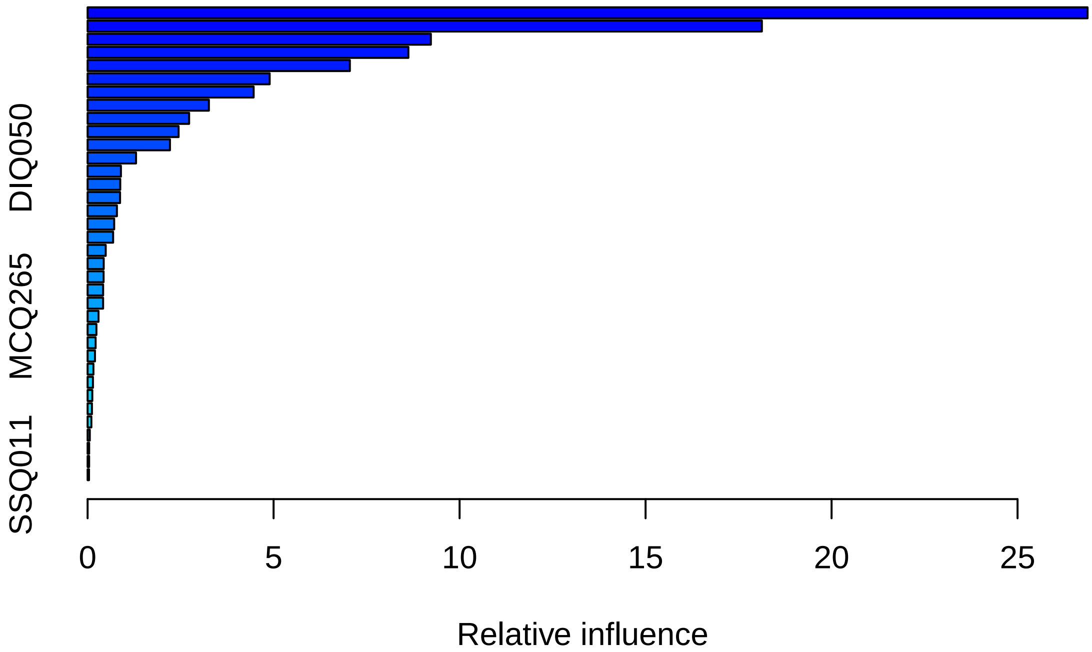
boosting

```r
library(gbm)
```

```
## Loaded gbm 2.1.8
```

```r
set.seed(3253)

a <- seq(-10, -1, by=0.5)
lambdas <- 10^a
MSE.training <- c()
MSE.testing <- c()

for (i in 1:length(lambdas)){
boosting.mod <- gbm(mortstat~., data = NHANES.training, distribution = "gaussian", n.trees = 1000, inte

boosting.training.pred <- predict(boosting.mod, NHANES.training,n.trees=1000)
boosting.testing.pred <- predict(boosting.mod, NHANES.testing,n.trees=1000)

MSE.training[i] <- mean((NHANES.training$mortstat-boosting.training.pred)^2)
MSE.testing[i] <- mean((testing.mortstat-boosting.testing.pred)^2)
}
```

```r
boosting.best <- gbm(mortstat~., data = NHANES.training, distribution = "gaussian", n.trees = 1000, inte

summary(boosting.best)
```




```
##                var     rel.inf
## RIDAGEYR RIDAGEYR 26.88195989
## LBXRDW     LBXRDW 18.12641161
## HSD010     HSD010  9.22796158
## BMXBMI     BMXBMI  8.62360862
## BPXDI1     BPXDI1  7.04995409
## BMXWAIST BMXWAIST  4.89434807
## BPXSY1     BPXSY1  4.46338145
## MCQ160B   MCQ160B  3.26020083
## RIAGENDR RIAGENDR  2.72896902
## BPXML1     BPXML1  2.44635559
## DIQ010     DIQ010  2.21414646
## DIQ050     DIQ050  1.30179636
## MCQ245A   MCQ245A  0.89538498
## MCQ160A   MCQ160A  0.87741648
## MCQ250A   MCQ250A  0.87220896
## MCQ220     MCQ220  0.78822142
## WHQ030     WHQ030  0.71426773
## DIQ090     DIQ090  0.68618030
## SSQ051     SSQ051  0.48661673
## WHQ040     WHQ040  0.43280434
## BPXPULS   BPXPULS  0.42937757
## MCQ250F   MCQ250F  0.41754843
## VIQ200     VIQ200  0.41624430
## MCQ265     MCQ265  0.29331061
## MCQ250G   MCQ250G  0.23668833
## MCQ250E   MCQ250E  0.21502059
## MCQ250C   MCQ250C  0.19787488
## MCQ250B   MCQ250B  0.15638291
```

```
## MCQ160M   MCQ160M  0.14331229
## MCQ010     MCQ010  0.12635022
## BPQ060     BPQ060  0.11641954
## MCQ160K   MCQ160K  0.10094283
## BPQ010     BPQ010  0.05856488
## MCQ053     MCQ053  0.04124174
## MCQ160L   MCQ160L  0.04063063
## SSQ011     SSQ011  0.03789570
```

```
boosting.probs <- predict(boosting.best, NHANES.testing, n.trees=1000)
boosting.pred <- rep(0,410)
boosting.pred[boosting.probs > .5] <- 1
```

```
# test for sensitivity and specificity
conf.boost <- table(boosting.pred,testing.mortstat)
conf.boost
```

```
##              testing.mortstat
## boosting.pred   0   1
##             0 342  51
##             1   4  13
```

```
sens.boost <- conf.boost[1,1]/(conf.boost[1,1]+conf.boost[1,2])
sens.boost
```

```
## [1] 0.870229
```

```
spec.boost <- conf.boost[2,2]/(conf.boost[2,1]+conf.boost[2,2])
spec.boost
```

```
## [1] 0.7647059
```

```
# testing error rate
err.boost <- mean(boosting.pred!=testing.mortstat)
err.boost
```

```
## [1] 0.1341463
```

The sensiticity using boosting is 87.02%. The specificity using boosting is 76.47%. The testing error using boosting is 13.41%.

Comparing model evaluations

```
a <- rbind(c(err.logistic, testing.err.lda, testing.err.qda, err.lasso, err.svm.lin, err.svm.rad, err.sv
               c(sens.logistic, sens.lda, sens.qda, sens.lasso, sens.svm.lin, sens.svm.rad, sens.svm.pol
               c(spec.logistic, spec.lda, spec.qda, spec.lasso, spec.svm.lin, spec.svm.rad, spec.svm.pol
print(a)
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.1560976 0.1512195 0.1975610 0.1536585 0.1560976 0.1560976 0.1658537
## [2,] 0.8596939 0.8585859 0.8711485 0.8476658 0.8472906 0.8439024 0.8492462
## [3,] 0.5000000 0.5714286 0.3396226 0.6666667 0.5000000       NaN 0.3333333
##           [,8]      [,9]
## [1,] 0.1463415 0.1341463
## [2,] 0.8763158 0.8702290
## [3,] 0.5666667 0.7647059
```

The results show testing error rate, sensetivity, and specificity for the 9 models respectively, using the list of 36 predictors. Model 8 (bagging) has the highest sensitivity, model 9 (boosting) has the highest specificity as well as the lowest error rate (highest accuracy). Overall, boosting method is optimal for classifing mortatily in the NHANES dataset.

The boosting model achieves sensitivity of 87.02% and specificity of 76.47%, with a testing error rate of 13.41%. Using this model, the top 5 important variables in prediction are: “RIDAGEYR”, “LBXRDW”, “HSD010”, “BMXBMI”, and “BPXDI1”.