ML algorithms: Regularized Regression, and Gradient Boosting

Miras Tolepbergen

2023-11-26

```
install.packages("glmnet", repos='http://cran.us.r-project.org')

install.packages("xgboost", repos='http://cran.us.r-project.org')

install.packages("Ckmeans.1d.dp", repos='http://cran.us.r-project.org')

install.packages("irr", repos='http://cran.us.r-project.org')

rm(list=ls(all=TRUE))
setwd("C:/Users/Miras/Desktop/u_m/1st/big_data_analytics/Labs/projects")
getwd()

## [1] "C:/Users/Miras/Desktop/u_m/1st/big_data_analytics/Labs/projects"

library(quanteda)

library(readtext)

library(glmnet)

library(xgboost)

library(Ckmeans.1d.dp)

library(dplyr)

library(iml)

library(future.callr)

library(ggplot2)
library(cowplot)
library(PerformanceAnalytics)

# let's prepare the training- and the validation-set

# Training-set
x_train <- read.csv("train_review23.csv", stringsAsFactors=FALSE)
str(x_train)

## 'data.frame':    495 obs. of  3 variables:
##  $ X        : int  1 16 19 22 29 37 39 40 41 43 ...
##  $ text     : chr  "plot : two teen couples go to a church party , drink
and then drive . \nthey get into an accident . \none of th"| __truncated__
"john carpenter makes b-movies . \nalways has ( \" halloween , \" \" escape
```

```
from new york , \" \" the thing \" )"| __truncated__ "the law of crowd
pleasing romantic movies states that the two leads must end up together by
film's end . \nif y"| __truncated__ " \" in dreams \" might keep you awake at
night , but not because of its creepy imagery , bizarre visual style o"|
__truncated__ ...
## $ Sentiment: chr  "neg" "neg" "neg" "neg" ...
```

```
x_train$Sentiment <- ifelse(x_train$Sentiment == 'neg', 0, 1)
x_train$Sentiment <- factor(x_train$Sentiment,  levels=c("0", "1"),
labels=c("negative", "positive"))
corpus_rain <- corpus(x_train)
tok_train <- tokens(corpus_rain , remove_punct = TRUE, remove_numbers=TRUE,
remove_symbols = TRUE, split_hyphens = TRUE, remove_separators = TRUE,
remove_URL = TRUE)
```

```
## Warning: remove_URL argument is not used.
```

```
tok_train <- tokens_remove(tok_train, stopwords("en"))
# let's also remove the unicode symbols
tok_train <- tokens_remove(tok_train, c("0*"))
tok_train <- tokens_wordstem (tok_train)
Dfm_train <- dfm(tok_train)

# let's keep only features with at least 2 characters
Dfm_train <- dfm_trim(Dfm_train , min_docfreq = 2, verbose=TRUE)
```

```
## Removing features occurring:

##   - in fewer than 2 documents: 6,217

##   Total features removed: 6,217 (43.0%).
```

```
Dfm_train  <- dfm_remove(Dfm_train , min_nchar = 2)
Dfm_train <- dfm_trim(Dfm_train, min_termfreq = 0.80, termfreq_type =
"quantile",
                 max_docfreq = 0.2, docfreq_type = "prop")
nfeat(Dfm_train)
```

```
## [1] 1438
```

```
topfeatures(Dfm_train , 20)  # 20 top words
```

```
##    alien   famili    night     girl    black   horror    money     seri
##      195      175      170      163      157      148      141      141
##      kid  special     case     anim    power     rate      got    death
##      140      140      137      137      136      134      133      132
##      sex      die question      men
##      132      130      129      129
```

```
train <- as(Dfm_train, "dgCMatrix")
colnames(train) <- make.names(colnames(train), unique=TRUE)
```

```r
# Validation-set

x_val <- read.csv("validation_review23.csv", stringsAsFactors=FALSE)
str(x_val)

## 'data.frame':    425 obs. of  3 variables:
##  $ X        : int  1 15 16 19 22 29 37 39 40 41 ...
##  $ text     : chr  "the happy bastard's quick movie review \ndamn that y2k
bug . \nit's got a head start in this movie starring jam"| __truncated__ "i'm
really starting to wonder about alicia silverstone . \nsure , she is one of
the most beautiful creatures on"| __truncated__ "so what do you get when you
mix together plot elements from various successful sci-fi films such as close
encou"| __truncated__ " \" knock off \" is exactly that : a cheap knock off
of an action movie . \nit's also the worst movie i have se"| __truncated__
...
##  $ Sentiment: chr  "neg" "neg" "neg" "neg" ...

x_val$Sentiment <- ifelse(x_val$Sentiment == "neg", 0, 1)
x_val$Sentiment <- factor(x_val$Sentiment,  levels=c("0", "1"),
labels=c("negative", "positive"))
corpus_val <- corpus(x_val)
tok_val <- tokens(corpus_val , remove_punct = TRUE, remove_numbers=TRUE,
remove_symbols = TRUE, split_hyphens = TRUE, remove_separators = TRUE)
tok_val <- tokens_remove(tok_val, stopwords("en"))
tok_val <- tokens_remove(tok_val, c("0*"))
tok_val <- tokens_wordstem (tok_val)
Dfm_val <- dfm(tok_val)
Dfm_val <- dfm_trim(Dfm_val , min_docfreq = 2, verbose=TRUE)

## Removing features occurring:

##   - in fewer than 2 documents: 6,222

##   Total features removed: 6,222 (45.4%).

Dfm_val <- dfm_remove(Dfm_val , min_nchar = 2)
Dfm_val <- dfm_trim(Dfm_val, min_termfreq = 0.80, termfreq_type = "quantile",
                  max_docfreq = 0.2, docfreq_type = "prop")
nfeat(Dfm_val)

## [1] 1311

# let's match the features included in the training and in the validation-set
setequal(featnames(Dfm_train), featnames(Dfm_val ))

## [1] FALSE

val_dfm  <- dfm_match(Dfm_val , features = featnames(Dfm_train))
setequal(featnames(Dfm_train), featnames(val_dfm ))

## [1] TRUE
```

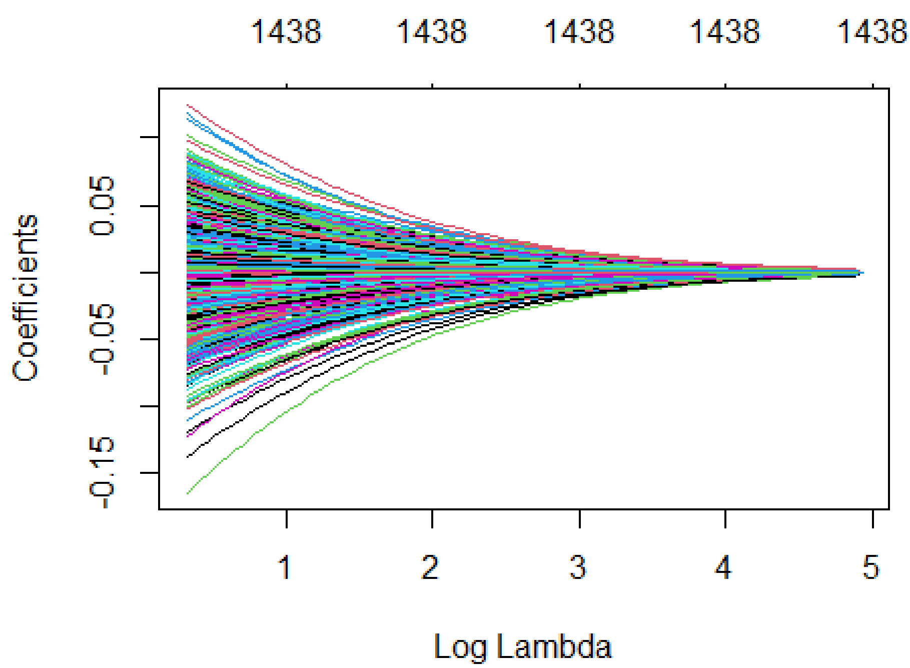
```r
ValM <- as(val_dfm , "dgCMatrix")
```

RR Model

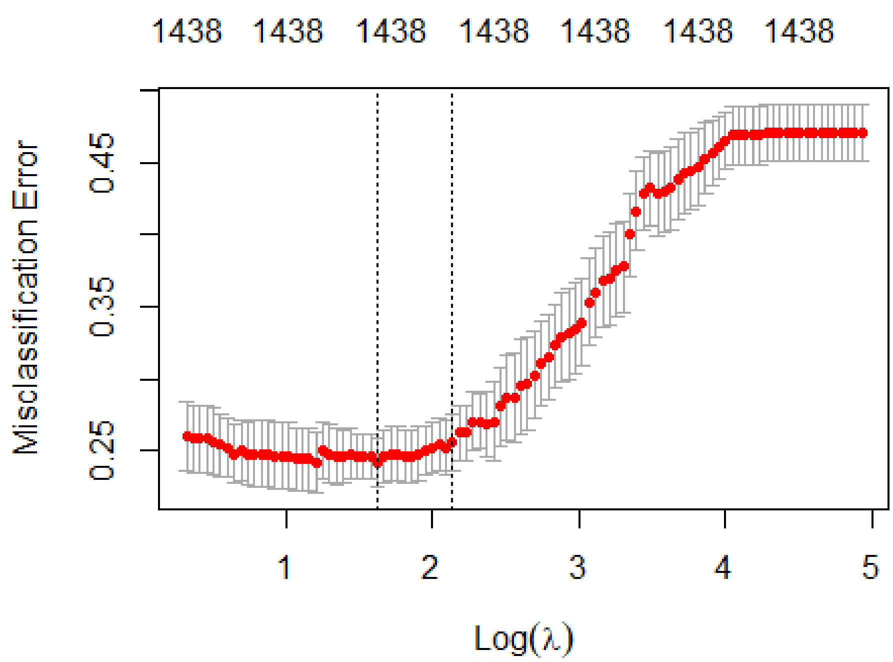
```r
# Let's start with a Regularized regression - RIDGE model

class(Dfm_train@docvars$Sentiment)

## [1] "factor"

system.time(ROLS <- glmnet(y= Dfm_train@docvars$Sentiment, x=train,
family="binomial", alpha=0, trace.it=1))

##

##    user  system elapsed
##    0.14    0.00    0.14

ROLS

##
## Call:  glmnet(x = train, y = Dfm_train@docvars$Sentiment, family =
"binomial",      alpha = 0, trace.it = 1)
##
##        Df  %Dev  Lambda
## 1    1438  0.00 138.500
## 2    1438  1.60 132.300
## 3    1438  1.68 126.200
## 4    1438  1.75 120.500
## 5    1438  1.83 115.000
## 6    1438  1.92 109.800
## 7    1438  2.01 104.800
## 8    1438  2.10 100.000
## 9    1438  2.19  95.500
## 10   1438  2.29  91.160
## 11   1438  2.40  87.010
## 12   1438  2.50  83.060
## 13   1438  2.62  79.280
## 14   1438  2.74  75.680
## 15   1438  2.86  72.240
## 16   1438  2.99  68.960
## 17   1438  3.12  65.820
## 18   1438  3.26  62.830
## 19   1438  3.40  59.970
## 20   1438  3.55  57.250
## 21   1438  3.71  54.650
## 22   1438  3.87  52.160
## 23   1438  4.04  49.790
## 24   1438  4.22  47.530
## 25   1438  4.40  45.370
## 26   1438  4.59  43.310
```

```
## 27  1438  4.79  41.340
## 28  1438  5.00  39.460
## 29  1438  5.21  37.670
## 30  1438  5.44  35.950
## 31  1438  5.67  34.320
## 32  1438  5.91  32.760
## 33  1438  6.15  31.270
## 34  1438  6.41  29.850
## 35  1438  6.68  28.490
## 36  1438  6.95  27.200
## 37  1438  7.24  25.960
## 38  1438  7.54  24.780
## 39  1438  7.84  23.650
## 40  1438  8.16  22.580
## 41  1438  8.49  21.550
## 42  1438  8.83  20.570
## 43  1438  9.17  19.640
## 44  1438  9.54  18.750
## 45  1438  9.91  17.890
## 46  1438 10.29  17.080
## 47  1438 10.69  16.300
## 48  1438 11.10  15.560
## 49  1438 11.52  14.860
## 50  1438 11.95  14.180
## 51  1438 12.39  13.540
## 52  1438 12.85  12.920
## 53  1438 13.32  12.330
## 54  1438 13.80  11.770
## 55  1438 14.30  11.240
## 56  1438 14.80  10.730
## 57  1438 15.33  10.240
## 58  1438 15.86   9.774
## 59  1438 16.41   9.330
## 60  1438 16.97   8.906
## 61  1438 17.54   8.501
## 62  1438 18.12   8.115
## 63  1438 18.72   7.746
## 64  1438 19.28   7.394
## 65  1438 19.90   7.058
## 66  1438 20.53   6.737
## 67  1438 21.18   6.431
## 68  1438 21.84   6.139
## 69  1438 22.51   5.860
## 70  1438 23.19   5.593
## 71  1438 23.88   5.339
## 72  1438 24.58   5.096
## 73  1438 25.30   4.865
## 74  1438 26.02   4.644
## 75  1438 26.76   4.433
## 76  1438 27.51   4.231
```

```
## 77  1438 28.26   4.039
## 78  1438 29.03   3.855
## 79  1438 29.80   3.680
## 80  1438 30.59   3.513
## 81  1438 31.38   3.353
## 82  1438 32.18   3.201
## 83  1438 32.99   3.055
## 84  1438 33.80   2.916
## 85  1438 34.63   2.784
## 86  1438 35.46   2.657
## 87  1438 36.29   2.536
## 88  1438 37.13   2.421
## 89  1438 37.98   2.311
## 90  1438 38.83   2.206
## 91  1438 39.69   2.106
## 92  1438 40.55   2.010
## 93  1438 41.41   1.919
## 94  1438 42.27   1.832
## 95  1438 43.14   1.748
## 96  1438 44.01   1.669
## 97  1438 44.89   1.593
## 98  1438 45.76   1.521
## 99  1438 46.63   1.451
## 100 1438 47.51   1.385

plot(ROLS, xvar = "lambda")
```

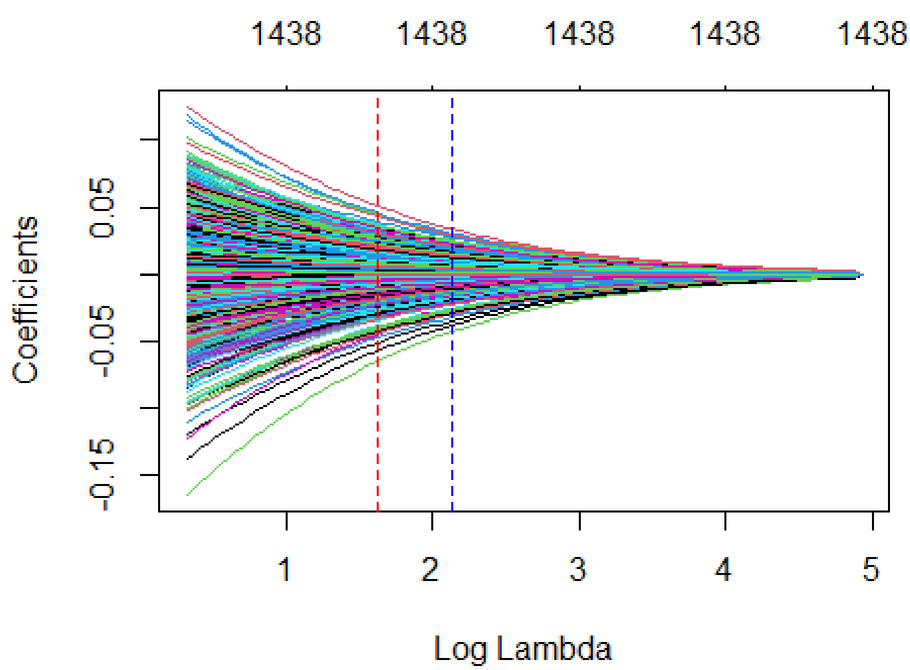


```r
set.seed(123)
system.time(ridge <-  cv.glmnet(y= Dfm_train@docvars$Sentiment, x=train,
    family="binomial", alpha=0, nfolds=10, type.measure="class", trace.it=1
))

## Training
##    |
##    user  system elapsed
##    1.73    0.15    1.90

plot(ridge)
```




```r
min(ridge $cvm) # minimum miss-classification error (i.e., 1-accuracy)

## [1] 0.2424242

ridge $lambda.min # lambda for this minimum

## [1] 5.096305

log(ridge $lambda.min )

## [1] 1.628516

ridge $cvm[ridge $lambda == ridge $lambda.1se]  # 1 st.error of minimum miss-
classification error

## [1] 0.2565657
```

```
ridge $lambda.1se  # lambda for this error

## [1] 8.501149

log(ridge$lambda.1se)

## [1] 2.140201

lse <- as.numeric(ridge$lambda.1se) # let's save the lambda in this latter
case
lse

## [1] 8.501149

plot(ROLS, xvar = "lambda")
abline(v = log(ridge $lambda.min), col = "red", lty = "dashed")
abline(v = log(ridge $lambda.1se), col = "blue", lty = "dashed")
```



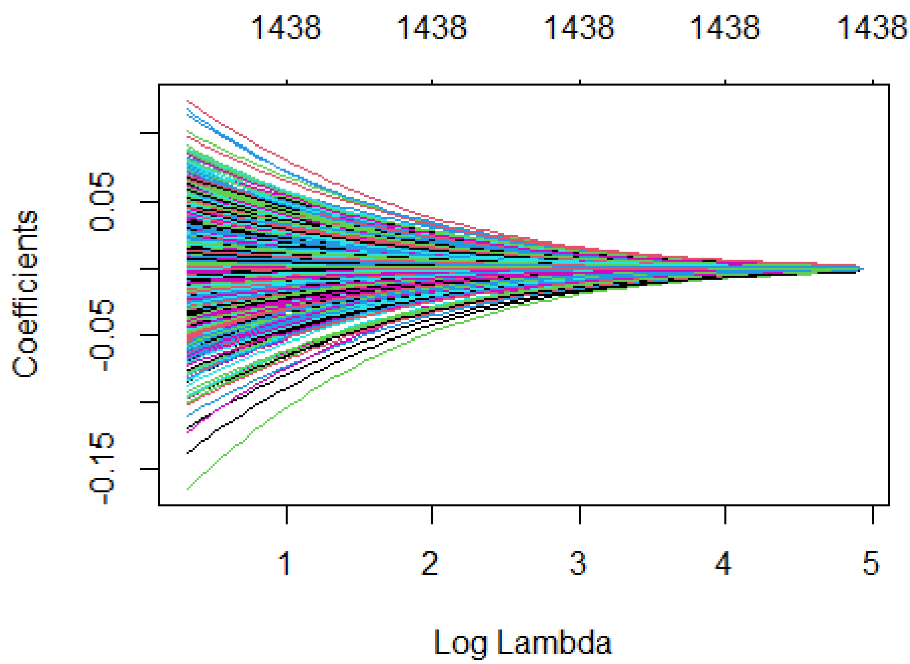

```
# Lasso regression model

system.time(ROLS_lasso <- glmnet(y= Dfm_train@docvars$Sentiment, x=train,
family="binomial", alpha=1, trace.it=1))

##   |
##    user  system elapsed
##    0.08    0.00    0.08

ROLS_lasso
```

```
##
## Call:  glmnet(x = train, y = Dfm_train@docvars$Sentiment, family =
"binomial",      alpha = 1, trace.it = 1)
##
##      Df  %Dev   Lambda
## 1     0  0.00 0.138500
## 2     1  0.49 0.132300
## 3     1  0.94 0.126200
## 4     2  1.47 0.120500
## 5     3  2.29 0.115000
## 6     4  3.21 0.109800
## 7     4  4.19 0.104800
## 8     5  5.19 0.100000
## 9     5  6.13 0.095500
## 10    7  7.13 0.091160
## 11   10  8.35 0.087010
## 12   12  9.68 0.083060
## 13   13 11.04 0.079280
## 14   14 12.43 0.075680
## 15   20 13.88 0.072240
## 16   22 15.49 0.068960
## 17   24 17.09 0.065820
## 18   31 18.90 0.062830
## 19   32 20.77 0.059970
## 20   36 22.57 0.057250
## 21   40 24.43 0.054650
## 22   43 26.23 0.052160
## 23   48 28.03 0.049790
## 24   53 29.84 0.047530
## 25   61 31.66 0.045370
## 26   65 33.50 0.043310
## 27   70 35.35 0.041340
## 28   72 37.18 0.039460
## 29   74 38.93 0.037670
## 30   79 40.65 0.035950
## 31   84 42.39 0.034320
## 32   87 44.07 0.032760
## 33   89 45.72 0.031270
## 34   95 47.34 0.029850
## 35   99 48.93 0.028490
## 36  105 50.58 0.027200
## 37  110 52.25 0.025960
## 38  115 53.89 0.024780
## 39  121 55.49 0.023650
## 40  127 57.06 0.022580
## 41  131 58.59 0.021550
## 42  139 60.08 0.020570
## 43  145 61.58 0.019640
## 44  147 63.03 0.018750
## 45  148 64.44 0.017890
```

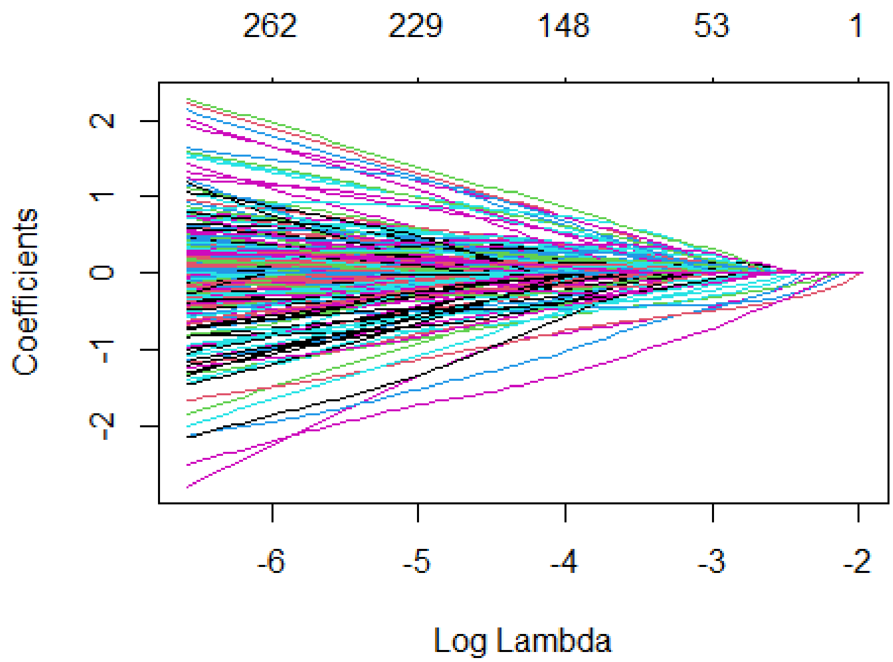
```
## 46  152 65.81 0.017080
## 47  160 67.14 0.016300
## 48  166 68.46 0.015560
## 49  172 69.74 0.014860
## 50  176 70.99 0.014180
## 51  180 72.20 0.013540
## 52  183 73.37 0.012920
## 53  192 74.51 0.012330
## 54  196 75.61 0.011770
## 55  200 76.66 0.011240
## 56  201 77.67 0.010730
## 57  203 78.64 0.010240
## 58  204 79.57 0.009774
## 59  208 80.46 0.009330
## 60  210 81.31 0.008906
## 61  216 82.14 0.008501
## 62  219 82.94 0.008115
## 63  224 83.70 0.007746
## 64  227 84.43 0.007394
## 65  226 85.13 0.007058
## 66  229 85.80 0.006737
## 67  232 86.44 0.006431
## 68  232 87.06 0.006139
## 69  233 87.64 0.005860
## 70  232 88.20 0.005593
## 71  232 88.73 0.005339
## 72  234 89.24 0.005096
## 73  233 89.72 0.004865
## 74  234 90.19 0.004644
## 75  239 90.63 0.004433
## 76  239 91.05 0.004231
## 77  241 91.46 0.004039
## 78  246 91.85 0.003855
## 79  251 92.22 0.003680
## 80  251 92.57 0.003513
## 81  252 92.91 0.003353
## 82  253 93.23 0.003201
## 83  255 93.54 0.003055
## 84  255 93.84 0.002916
## 85  258 94.12 0.002784
## 86  259 94.39 0.002657
## 87  260 94.65 0.002536
## 88  262 94.89 0.002421
## 89  262 95.12 0.002311
## 90  263 95.35 0.002206
## 91  264 95.56 0.002106
## 92  265 95.76 0.002010
## 93  265 95.96 0.001919
## 94  265 96.14 0.001832
## 95  266 96.32 0.001748
```

```
## 96  266 96.48 0.001669
## 97  268 96.64 0.001593
## 98  268 96.80 0.001521
## 99  272 96.94 0.001451
## 100 274 97.08 0.001385
```

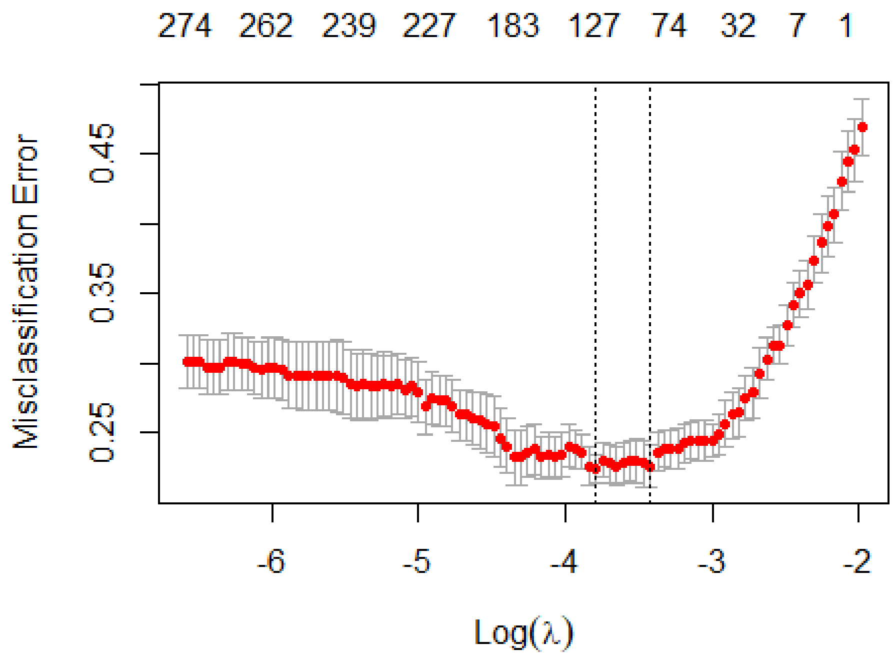
```r
# as you can see as lambda increases, the penalty becomes large and forces
our coefficients to zero
plot(ROLS, xvar = "lambda") # with ridge
```




```r
plot(ROLS_lasso, xvar = "lambda") # with lasso
```

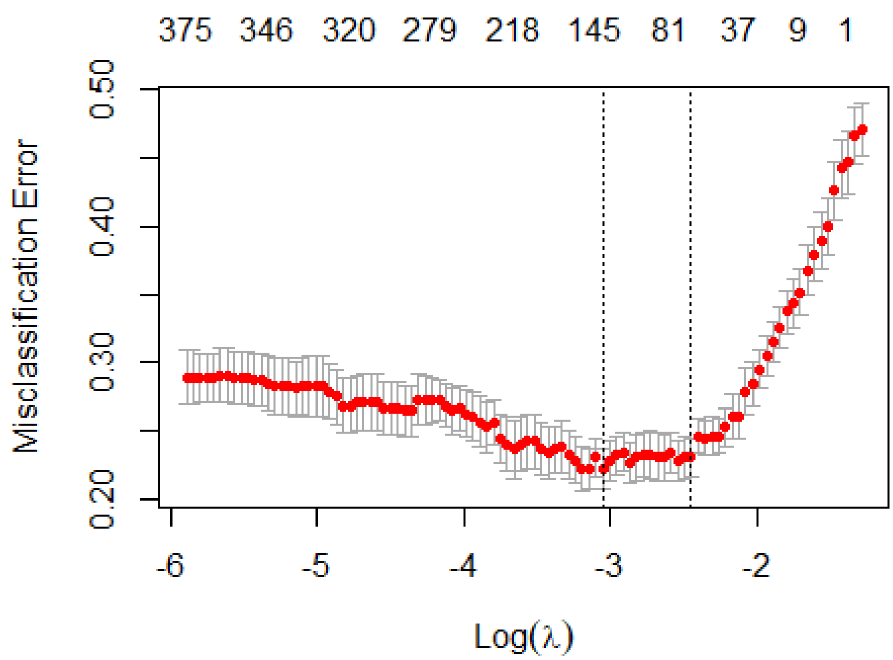



```
set.seed(123)
system.time(lasso <-  cv.glmnet(y= Dfm_train@docvars$Sentiment, x=train,
    family="binomial", alpha=1, nfolds=10, type.measure="class", trace.it=1
))

## Training
##    |
##    user  system elapsed
##    0.86    0.03    0.89

plot(lasso )
```

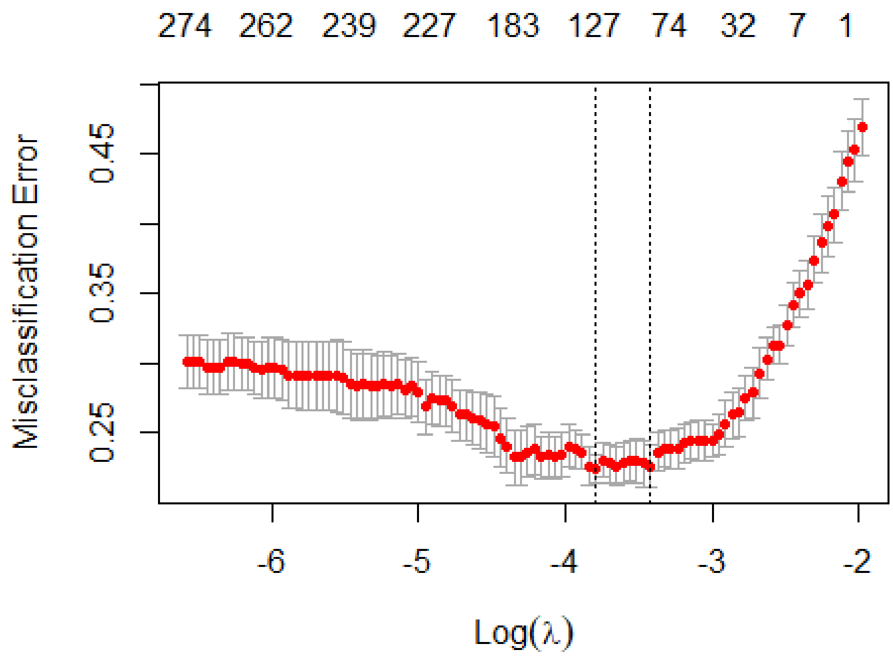



```
min(lasso $cvm)        # minimum miss-classification error of the lasso
regression

## [1] 0.2242424

min(ridge $cvm)        # minimum miss-classification error of the ridge
regression

## [1] 0.2424242

lasso $lambda.min      # lambda for this value

## [1] 0.0225798

log(lasso $lambda.min)

## [1] -3.7907

lasso $lambda.1se

## [1] 0.03275942

log(lasso $lambda.1se)

## [1] -3.418565

#elastic nets regression model

set.seed(123)
```

```r
system.time(elastic<-  cv.glmnet(y= Dfm_train@docvars$Sentiment, x=train,
    family="binomial", alpha=0.5, nfolds=10,  trace.it=1,
    type.measure="class"))

## Training
##   |
##    user  system elapsed
##    0.88    0.06    0.96

plot(elastic)
```




```r
# note the difference (in the number of features) with...
plot(lasso )
```




```
min(elastic$cvm)        # minimum miss-classification error of the elastic
nets regression model

## [1] 0.2222222

elastic$lambda.min      # lambda for this value

## [1] 0.0473099

log(elastic$lambda.min)

## [1] -3.051036

elastic$lambda.1se

## [1] 0.08661205

log(elastic$lambda.1se)

## [1] -2.446316

# it seems that the ridge model is the one performing better in our case

min(ridge $cvm)        # minimum miss-classification error of the ridge
regression

## [1] 0.2424242
```

```r
min(lasso $cvm)       # minimum miss-classification error of the lasso
regression

## [1] 0.2242424

min(elastic$cvm)      # minimum miss-classification error of the elastic nets
regression model

## [1] 0.2222222

# GB model

class(Dfm_train@docvars$Sentiment)

## [1] "factor"

num <- as.numeric(Dfm_train@docvars$Sentiment)
num

##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
##  [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [112] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [149] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [186] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [223] 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
## [260] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
## [297] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
## [334] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
## [371] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
## [408] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
## [445] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
## [482] 2 2 2 2 2 2 2 2 2 2 2 2 2 2

num[ num ==1 ] <-0
num[ num ==2 ] <-1
table(num)
```

```
## num
##   0   1
## 233 262
```

```
table(Dfm_train@docvars$Sentiment)
```

```
##
## negative positive
##      233      262
```

```
set.seed(123)
system.time(xgb.fit1 <- xgboost(
  data = train,
  label = num,
  nrounds = 500,
  objective = "binary:hinge",
  eval_metric = "error", # binary classification error rate
  verbose = 1  # not silent; if you want it silent "verbose=0"
))
```

```
## [1]  train-error:0.470707
## [2]  train-error:0.224242
## [3]  train-error:0.206061
## [4]  train-error:0.177778
## [5]  train-error:0.149495
##    user  system elapsed
##    5.41    0.40    2.19
```

```
predicted_xgb <- predict(xgb.fit1, train)
table(predicted_xgb)
```

```
## predicted_xgb
##   0   1
## 233 262
```

```
prop.table(table(predicted_xgb))
```

```
## predicted_xgb
##         0         1
## 0.4707071 0.5292929
```

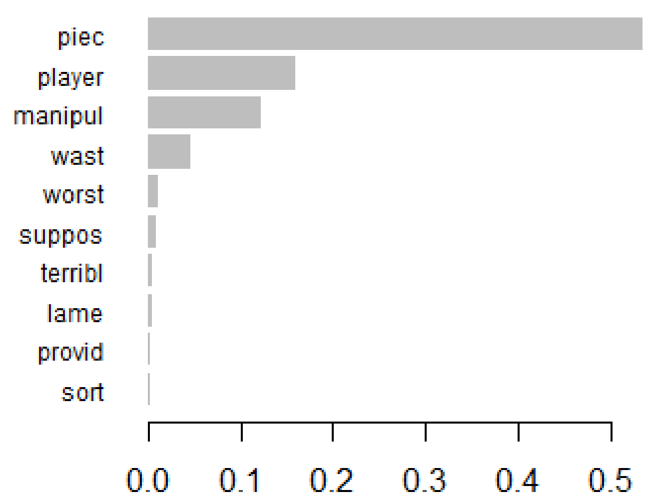
```
# Compute feature importance matrix
importance_matrix <- xgb.importance(model = xgb.fit1)
head(importance_matrix)
```

```
##    Feature        Gain      Cover   Frequency
## 1:    piec 0.535781733 0.27395474 0.290577508
## 2:  player 0.159050114 0.15407760 0.223708207
## 3: manipul 0.122300355 0.13966025 0.223100304
## 4:    wast 0.046963969 0.06494076 0.054103343
## 5:   worst 0.010880401 0.02890992 0.003039514
## 6:  suppos 0.008543705 0.02711716 0.003039514
```

```r
# Compute feature importance matrix
importance <- importance_matrix[order(importance_matrix$Gain,
decreasing=TRUE),]
head(importance, n=10)
```

```
##     Feature        Gain       Cover   Frequency
##  1:    piec 0.535781733 0.273954742 0.290577508
##  2:  player 0.159050114 0.154077603 0.223708207
##  3: manipul 0.122300355 0.139660252 0.223100304
##  4:    wast 0.046963969 0.064940763 0.054103343
##  5:   worst 0.010880401 0.028909923 0.003039514
##  6:  suppos 0.008543705 0.027117157 0.003039514
##  7: terribl 0.005329597 0.021688711 0.003039514
##  8:    lame 0.005189577 0.021964521 0.003039514
##  9:  provid 0.003322674 0.003735974 0.003647416
## 10:    sort 0.002881917 0.004438037 0.003039514
```

```r
xgb.plot.importance(importance_matrix, top_n = 10, measure = "Gain")
```



Internal validity

```r
# Ridge model

# let's clean the features's name
colnames(ValM)
```

```
##    [1] "teen"            "coupl"           "parti"           "drink"
##    [5] "drive"           "accid"           "die"             "girlfriend"
##    [9] "continu"         "deal"            "generat"         "touch"
##   [13] "cool"            "present"         "review"          "write"


# let's convert now the matrix into a data frame
valDF <- as.data.frame(as.matrix(ValM))
class(valDF)

## [1] "data.frame"

# Let's estimate the Ridge on the "cleaned" matrix

set.seed(123)
system.time(ridge <-  cv.glmnet(y= Dfm_val@docvars$Sentiment, x=ValM,
    family="binomial", alpha=0, nfolds=5, type.measure="class", trace.it=1 ))

## Training
##    |

##    user  system elapsed
##    0.71    0.06    0.78

newROLS <-  glmnet(y= Dfm_val@docvars$Sentiment, x=ValM,
    family="binomial", alpha=0, trace.it=1, lambda=ridge $lambda.min)

##    |
|                                                                       |   0%



# Then employ the below predictive function:

predRidge <-function(model, newdata){
newData_x <- as.matrix(newdata)
results<- predict(model, newData_x, type="class")
return(results)
 }


modRid <-  Predictor$new(newROLS, data = valDF, y =Dfm_val@docvars$Sentiment,
predict.fun = predRidge)
modRid $predict(valDF[1:10, ])

##    negative positive
## 1         1        0
## 2         1        0
## 3         1        0
## 4         1        0
## 5         1        0
## 6         1        0
```

```
## 7          1        0
## 8          1        0
## 9          1        0
## 10         0        1
```

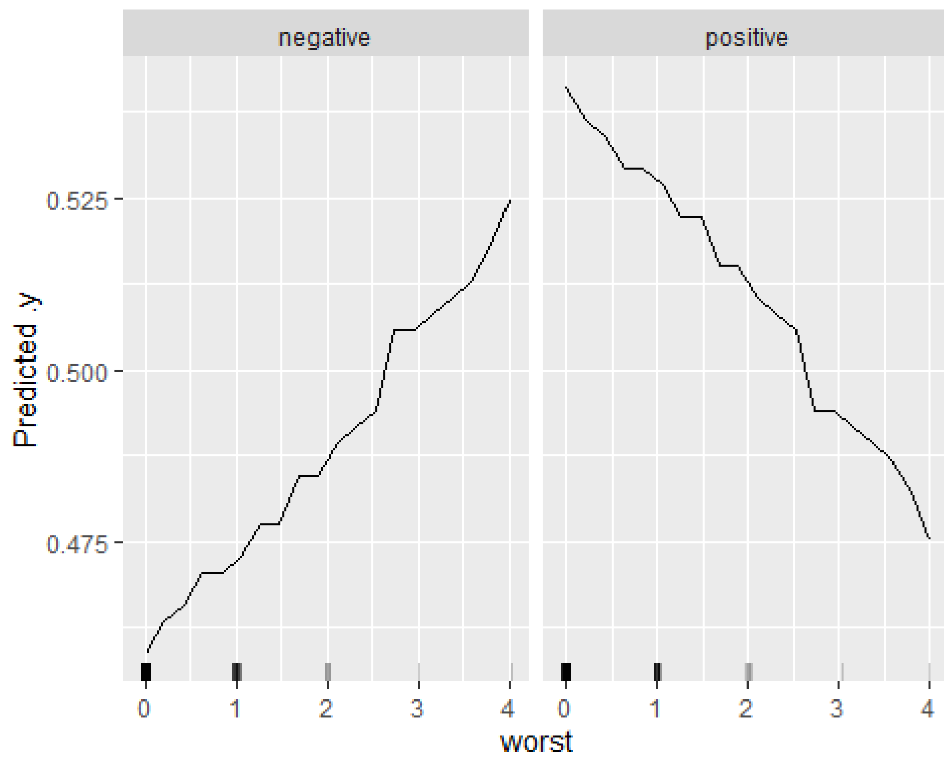
```r
system.time({
  plan("callr", workers = 4)
 set.seed(123)
impRidge <- FeatureImp$new(modRid , loss = "ce", n.repetitions=3)
})
```

```
##    user  system elapsed
##   11.02    0.89  319.06
```

```r
head(impRidge $results[,c(1:4)],10)
```

```
##    feature importance.05 importance importance.95
## 1     titl      1.084615   1.153846      1.153846
## 2    found      1.153846   1.153846      1.153846
## 3    desir      1.153846   1.153846      1.153846
## 4    worst      1.153846   1.153846      1.223077
## 5     mike      1.015385   1.153846      1.153846
## 6     lost      1.076923   1.076923      1.076923
## 7   explan      1.076923   1.076923      1.076923
## 8      els      1.076923   1.076923      1.076923
## 9    carri      1.007692   1.076923      1.076923
## 10    view      1.007692   1.076923      1.076923
```

```r
# Let's compute a Partial Dependence Plot (PDP) for the word "worst"
plot(FeatureEffect$new(modRid , "worst", method = "pdp"))
```




```
# XGBOOST model

num <- as.factor(Dfm_val@docvars$Sentiment)
num

##   [1] negative negative negative negative negative negative negative
negative
##   [9] negative negative negative negative negative negative negative
negative
##  [17] negative negative negative negative negative negative negative
negative
##  [25] negative negative negative negative negative negative negative
negative
##  [33] negative negative negative negative negative negative negative
negative
##  [41] negative negative negative negative negative negative negative
negative
##  [49] negative negative negative negative negative negative negative
negative
##  [57] negative negative negative negative negative negative negative
negative
##  [65] negative negative negative negative negative negative negative
negative
##  [73] negative negative negative negative negative negative negative
negative
##  [81] negative negative negative negative negative negative negative
negative
```

```
##  [89] negative negative negative negative negative negative negative
negative
##  [97] negative negative negative negative negative negative negative
negative
## [105] negative negative negative negative negative negative negative
negative
## [113] negative negative negative negative negative negative negative
negative
## [121] negative negative negative negative negative negative negative
negative
## [129] negative negative negative negative negative negative negative
negative
## [137] negative negative negative negative negative negative negative
negative
## [145] negative negative negative negative negative negative negative
negative
## [153] negative negative negative negative negative negative negative
negative
## [161] negative negative negative negative negative negative negative
negative
## [169] negative negative negative negative negative negative negative
negative
## [177] negative negative negative negative negative negative negative
negative
## [185] negative negative negative negative negative negative negative
negative
## [193] negative negative negative negative negative negative negative
negative
## [201] positive positive positive positive positive positive positive
positive
## [209] positive positive positive positive positive positive positive
positive
## [217] positive positive positive positive positive positive positive
positive
## [225] positive positive positive positive positive positive positive
positive
## [233] positive positive positive positive positive positive positive
positive
## [241] positive positive positive positive positive positive positive
positive
## [249] positive positive positive positive positive positive positive
positive
## [257] positive positive positive positive positive positive positive
positive
## [265] positive positive positive positive positive positive positive
positive
## [273] positive positive positive positive positive positive positive
positive
## [281] positive positive positive positive positive positive positive
positive
```

```
## [289] positive positive positive positive positive positive positive
positive
## [297] positive positive positive positive positive positive positive
positive
## [305] positive positive positive positive positive positive positive
positive
## [313] positive positive positive positive positive positive positive
positive
## [321] positive positive positive positive positive positive positive
positive
## [329] positive positive positive positive positive positive positive
positive
## [337] positive positive positive positive positive positive positive
positive
## [345] positive positive positive positive positive positive positive
positive
## [353] positive positive positive positive positive positive positive
positive
## [361] positive positive positive positive positive positive positive
positive
## [369] positive positive positive positive positive positive positive
positive
## [377] positive positive positive positive positive positive positive
positive
## [385] positive positive positive positive positive positive positive
positive
## [393] positive positive positive positive positive positive positive
positive
## [401] positive positive positive positive positive positive positive
positive
## [409] positive positive positive positive positive positive positive
positive
## [417] positive positive positive positive positive positive positive
positive
## [425] positive
## Levels: negative positive
```

```r
table(num)
```

```
## num
## negative positive
##      200      225
```

```r
num <- as.numeric(num)
num
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
##  [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```
1 1 1
## [112] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [149] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [186] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
## [223] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
## [260] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
## [297] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
## [334] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
## [371] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
## [408] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

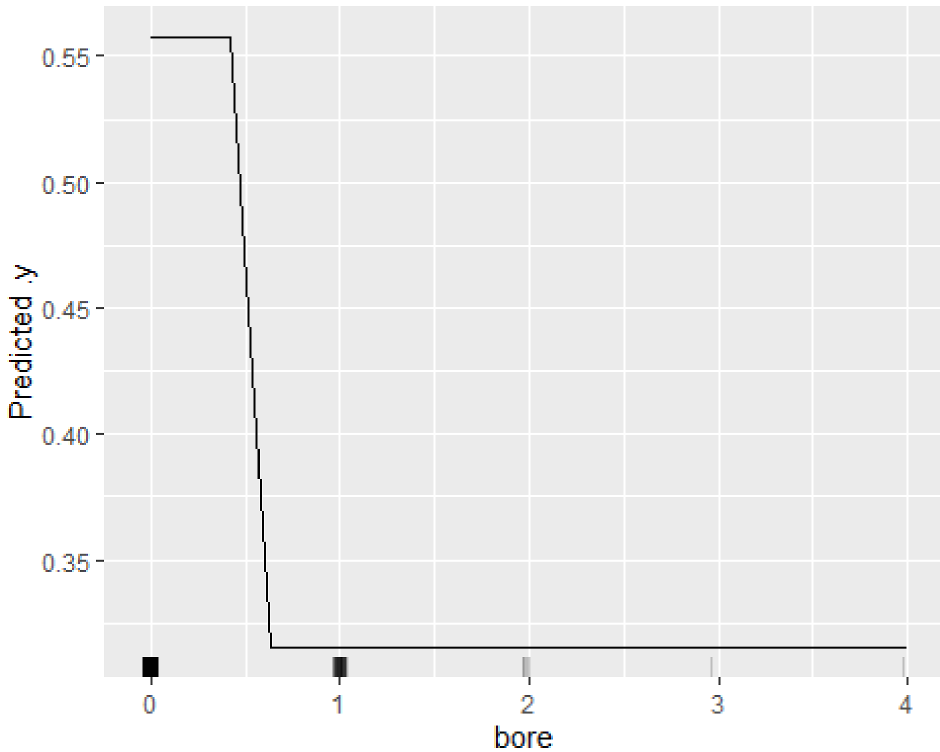
```
num[ num ==1 ] <-0
num[ num ==2 ] <-1
table(num)
```

```
## num
##   0   1
## 200 225
```

```
val2 <- as.matrix(valDF)

set.seed(123)
system.time(xgb.fit1 <- xgboost(
  data = val2 ,
  label = num,
  nrounds = 500,
eta = 1, nthread = 4,
  objective = "binary:hinge",  # for binary
 eval_metric = "error", # binary classification error rate
  verbose = 1  # not silent; if you want it silent "verbose=0"
))
```

```
## [1]  train-error:0.155294
```

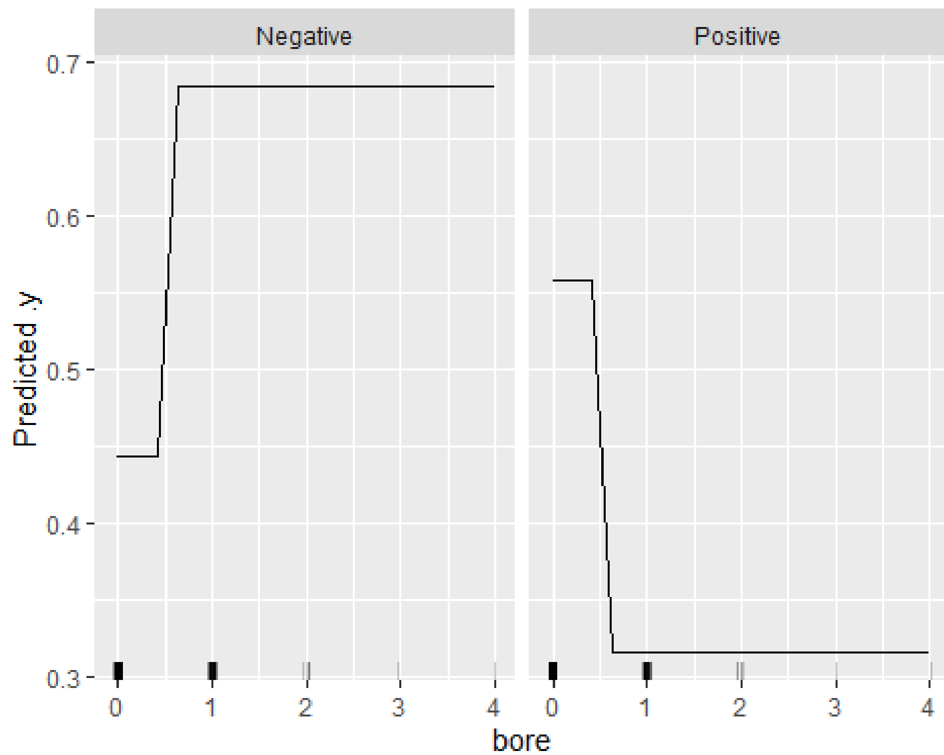
```r
# Let's compute a Partial Dependence Plot (PDP) for "bore"
plot(FeatureEffect$new(modXGB , "bore", method = "pdp"))
```




```r
predXGB2 <-function(model, newdata){
newData_x <- as.matrix(newdata)
predict <- predict(model, newData_x)
first <- ifelse(predict==0, 1, 0)
second <- ifelse(predict==1, 1, 0)
results <- as.data.frame(cbind(first, second))
colnames(results)[1] ="Negative"
colnames(results)[2] ="Positive"
return(results)
}

# prediction of both class-labels
modXGB2 <- Predictor$new(xgb.fit1, data = valDF, y =num, predict.fun =
predXGB2)
modXGB2 $predict(valDF[1:10, ])

##    Negative Positive
## 1         1        0
## 2         1        0
## 3         1        0
## 4         1        0
## 5         1        0
## 6         1        0
## 7         1        0
```

```
## 8          1        0
## 9          1        0
## 10         1        0
```

```
plot(FeatureEffect$new(modXGB2 , "bore", method = "pdp"))
```




```
library(cvTools)

## Warning: package 'cvTools' was built under R version 4.3.2

## Loading required package: lattice

## Loading required package: robustbase

## Warning: package 'robustbase' was built under R version 4.3.2

library(caret)

## Warning: package 'caret' was built under R version 4.3.2

##
## Attaching package: 'caret'

## The following object is masked from 'package:future':
##
##     cluster

library(reshape2)
```

```
## Warning: package 'reshape2' was built under R version 4.3.2
```

Internal Validity

```
table(Dfm_val@docvars$Sentiment)

##
## negative positive
##      200      225

prop.table(table(Dfm_val@docvars$Sentiment))

##
##  negative  positive
## 0.4705882 0.5294118

# CV with xgboost

# By default, the xgboost will be computed with nrounds=500 in the below
function
source("Function_CV2XG.R")
Function_CV2XG

## function (input, dt, k, DV, ML, nrounds = 500, eta = 0.3, max_depth = 6,
##     gamma = 0, min_child_weight = 1, subsample = 1, colsample_bytree = 1,
##     lambda = 1, alpha = 0)
## {
##     for (i in 1:k) {
##         train <- input[folds$subsets[folds$which != i], ]
##         validation <- input[folds$subsets[folds$which == i],
##             ]
##         set.seed(123)
##         model <- ML(label = DV[folds$subsets[folds$which != i]],
##             data = train, nrounds = nrounds, eta = eta, max_depth =
max_depth,
##             gamma = gamma, min_child_weight = min_child_weight,
##             subsample = subsample, colsample_bytree = colsample_bytree,
##             lambda = lambda, alpha = alpha, early_stopping_rounds = 100,
##             objective = "binary:hinge", eval_metric = "error")
##         pred <- predict(model, validation)
##         class_table <- table(Predictions = pred, Actual =
DV[folds$subsets[folds$which ==
##             i]])
##         print(class_table)
##         df <- confusionMatrix(class_table, mode = "everything")
##         dt[i, 1] <- df$overall[1]
##         dt[i, 2] <- df$byClass[11]
##         dt[i, 3] <- ((2 * (df)$byClass[1] *
(df)$byClass[3])/(df$byClass[1] +
##             (df)$byClass[3]) + (2 * (df)$byClass[2] *
```

```
(df)$byClass[4])/(df$byClass[2] +
##             (df)$byClass[4]))/2
##         dt[i, 4] <- (2 * (df)$byClass[1] * (df)$byClass[3])/(df$byClass[1]
+
##             (df)$byClass[3])
##         dt[i, 5] <- (2 * (df)$byClass[2] * (df)$byClass[4])/(df$byClass[2]
+
##             (df)$byClass[4])
##         colnames(dt)[1] <- "Accuracy"
##         colnames(dt)[2] <- "Avg. Balanced Accuracy"
##         colnames(dt)[3] <- "Avg. F1"
##         colnames(dt)[4] <- paste0("F1 ", levels(y)[1])
##         colnames(dt)[5] <- paste0("F1 ", levels(y)[2])
##         result <- dt
##     }
##     result
## }
```

```
ttt <- as(Dfm_val, "dgCMatrix")

data2 <- data.frame()
str(data2)
```

```
## 'data.frame':    0 obs. of  0 variables
```

```
set.seed(123)
folds <- cvFolds(NROW(ttt), K=5)
str(folds)
```

```
## List of 5
##  $ n      : num 425
##  $ K      : num 5
##  $ R      : num 1
##  $ subsets: int [1:425, 1] 415 179 14 195 306 118 299 229 244 423 ...
##  $ which  : int [1:425] 1 2 3 4 5 1 2 3 4 5 ...
##  - attr(*, "class")= chr "cvFolds"
```

```
class(Dfm_val@docvars$Sentiment)
```

```
## [1] "factor"
```

```
y <- Dfm_val@docvars$Sentiment
y2 <- as.numeric(y)
y2
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
##  [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [112] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```
1 1 1
## [149] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [186] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
## [223] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
## [260] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
## [297] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
## [334] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
## [371] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
## [408] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2

y2[ y2 ==1 ] <-0
y2[ y2 ==2 ] <-1
table(y2)

## y2
##   0   1
## 200 225

table(y)

## y
## negative positive
##      200      225

Function_CV2XG(input=ttt, dt=data2, k=5, DV=num, ML=xgboost)

## [1]  train-error:0.455882
## Will train until train_error hasn't improved in 100 rounds.


##             Actual
## Predictions  0  1
##           0 26 14
##           1 14 31
```

```r
# CV with glmnet

source("Function_CV2Glmnet.R")
Function_CV2Glmnet

## function (input, dt, DV, ML, foldid = folds$which, alpha)
## {
##     model <- ML(y = DV, x = input, family = "binomial", alpha = alpha,
##         foldid = folds$which, type.measure = "class", keep = TRUE,
##         trace.it = 1)
##     cnf <- confusion.glmnet(model$fit.preval, newy = y, family =
"binomial")
##     best <- model$index["min", ]
##     class_table <- cnf[[best]]
##     print(class_table)
##     {
##         df <- confusionMatrix(class_table, mode = "everything")
##         dt[1, 1] <- df$overall[1]
##         dt[1, 2] <- df$byClass[11]
##         dt[1, 3] <- ((2 * (df)$byClass[1] *
(df)$byClass[3])/(df$byClass[1] +
##             (df)$byClass[3]) + (2 * (df)$byClass[2] *
(df)$byClass[4])/(df$byClass[2] +
##             (df)$byClass[4]))/2
##         dt[1, 4] <- (2 * (df)$byClass[1] * (df)$byClass[3])/(df$byClass[1]
+
##             (df)$byClass[3])
##         dt[1, 5] <- (2 * (df)$byClass[2] * (df)$byClass[4])/(df$byClass[2]
+
##             (df)$byClass[4])
##         colnames(dt)[1] <- "Accuracy"
##         colnames(dt)[2] <- "Avg. Balanced Accuracy"
##         colnames(dt)[3] <- "Avg. F1"
##         colnames(dt)[4] <- paste0("F1 ", levels(y)[1])
##         colnames(dt)[5] <- paste0("F1 ", levels(y)[2])
##         result <- dt
##     }
##     result
## }

str(folds)

## List of 5
##  $ n      : num 425
##  $ K      : num 5
##  $ R      : num 1
##  $ subsets: int [1:425, 1] 415 179 14 195 306 118 299 229 244 423 ...
##  $ which  : int [1:425] 1 2 3 4 5 1 2 3 4 5 ...
##  - attr(*, "class")= chr "cvFolds"

table(folds$which)
```

```
##
##  1  2  3  4  5
## 85 85 85 85 85
```

```r
Function_CV2Glmnet(input=ttt, dt=data2, DV=y, ML=cv.glmnet, alpha=0,
foldid=folds$which)
```

```
## Training
##   |
##             True
## Predicted  negative positive Total
##   negative      139       43   182
##   positive       61      182   243
##   Total         200      225   425
##
##  Percent Correct:  0.7553
```

```
##    Accuracy Avg. Balanced Accuracy   Avg. F1 F1 negative F1 positive
## 1 0.7552941              0.7519444 0.7527632   0.7277487   0.7777778
```

```r
Ridge_res <- Function_CV2Glmnet(input=ttt, dt=data2, DV=y, ML=cv.glmnet,
alpha=0, foldid=folds$which)
```

```
## Training
##   |
##             True
## Predicted  negative positive Total
##   negative      139       43   182
##   positive       61      182   243
##   Total         200      225   425
##
##  Percent Correct:  0.7553
```

```r
Lasso_res <- Function_CV2Glmnet(ttt, data2, y, cv.glmnet, alpha=0.5)
```

```
## Training
##   |
##             True
## Predicted  negative positive Total
##   negative      135       56   191
##   positive       65      169   234
##   Total         200      225   425
##
##  Percent Correct:  0.7153
```

```r
Elastic_res <- Function_CV2Glmnet(ttt, data2, y, cv.glmnet, alpha=1)
```

```
## Training
##   |
##             True
## Predicted  negative positive Total
##   negative      127       50   177
```

```
##   positive       73      175   248
##   Total         200      225   425
##
##  Percent Correct:  0.7106
```

```r
#ridge is the best



# to summarize the results

result <- as.data.frame(colMeans(XGBoost_res[ , c(1, 2, 3)]))
str(result)
```

```
## 'data.frame':    3 obs. of  1 variable:
##  $ colMeans(XGBoost_res[, c(1, 2, 3)]): num  0.661 0.664 0.657
```

```r
result <- cbind(result, as.data.frame(colMeans(Ridge_res [ , c(1, 2, 3)])))
result <- cbind(result, as.data.frame(colMeans(Lasso_res [ , c(1, 2, 3)])))
result <- cbind(result, as.data.frame(colMeans(Elastic_res [ , c(1, 2, 3)])))
str(result)
```

```
## 'data.frame':    3 obs. of  4 variables:
##  $ colMeans(XGBoost_res[, c(1, 2, 3)]): num  0.661 0.664 0.657
##  $ colMeans(Ridge_res[, c(1, 2, 3)])  : num  0.755 0.752 0.753
##  $ colMeans(Lasso_res[, c(1, 2, 3)])  : num  0.715 0.713 0.713
##  $ colMeans(Elastic_res[, c(1, 2, 3)]): num  0.711 0.706 0.707
```

```r
resultT <- as.data.frame(t(result))
str(resultT)
```

```
## 'data.frame':    4 obs. of  3 variables:
##  $ Accuracy              : num  0.661 0.755 0.715 0.711
##  $ Avg. Balanced Accuracy: num  0.664 0.752 0.713 0.706
##  $ Avg. F1               : num  0.657 0.753 0.713 0.707
```

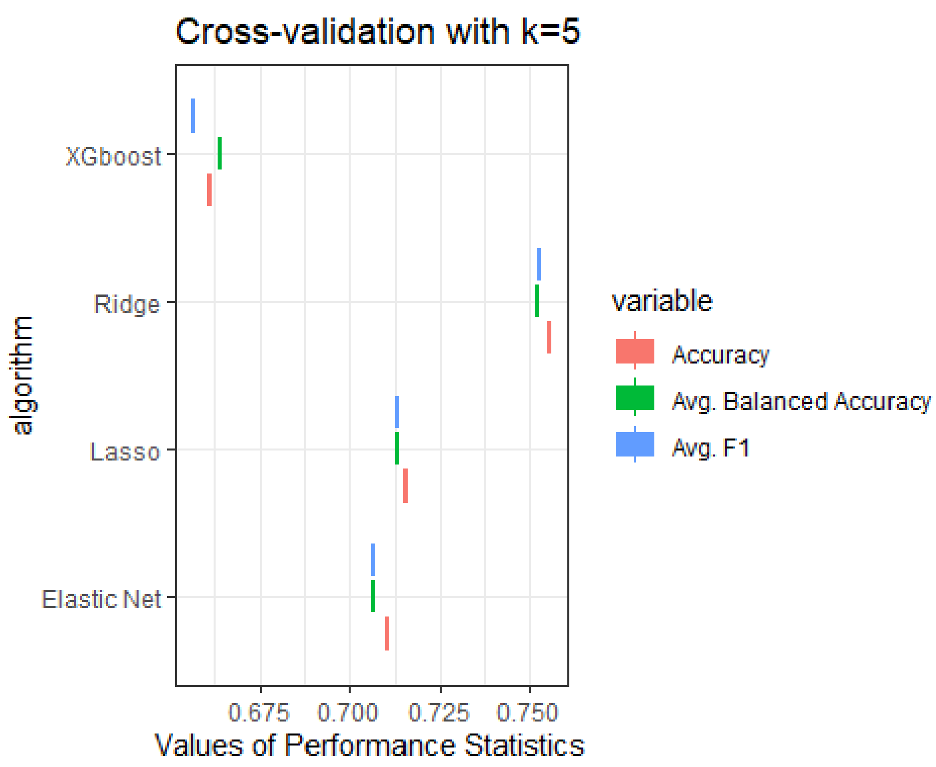
```r
row.names(resultT)[1] = "XGboost"
row.names(resultT)[2] = "Ridge"
row.names(resultT)[3] = "Lasso"
row.names(resultT)[4] = "Elastic Net"
str(resultT)
```

```
## 'data.frame':    4 obs. of  3 variables:
##  $ Accuracy              : num  0.661 0.755 0.715 0.711
##  $ Avg. Balanced Accuracy: num  0.664 0.752 0.713 0.706
##  $ Avg. F1               : num  0.657 0.753 0.713 0.707
```

```r
resultT$algorithm <- row.names(resultT)
str(resultT)
```

```
## 'data.frame':    4 obs. of  4 variables:
##  $ Accuracy              : num  0.661 0.755 0.715 0.711
```

```
##  $ Avg. Balanced Accuracy: num  0.664 0.752 0.713 0.706
##  $ Avg. F1               : num  0.657 0.753 0.713 0.707
##  $ algorithm             : chr  "XGboost" "Ridge" "Lasso" "Elastic Net"

df.long <- melt(resultT)

## Using algorithm as id variables

str(df.long)

## 'data.frame':    12 obs. of  3 variables:
##  $ algorithm: chr  "XGboost" "Ridge" "Lasso" "Elastic Net" ...
##  $ variable : Factor w/ 3 levels "Accuracy","Avg. Balanced Accuracy",..: 1
1 1 1 2 2 2 2 3 3 ...
##  $ value    : num  0.661 0.755 0.715 0.711 0.664 ...

ggplot(df.long,aes(algorithm,value,fill=variable, color = variable))+
geom_boxplot() + coord_flip() +
theme_bw()  + labs(title = "Cross-validation with k=5") + ylab(label="Values
of Performance Statistics")
```




```
#as already stated, ridge model performs the vest under the default settings

# Grid search

# (let's tune the hyperparameters for the XGBOOST

table(y)
```

```
## y
## negative positive
##      200      225
```

```r
table(y2)
```

```
## y2
##   0   1
## 200 225
```

```r
hyper_gridXG <- expand.grid(
  eta = c(0.3, 1, 2), # hyperparameter values
  max_depth = c(5:7), # hyperparameter values
  nrounds  = c(500, 1000, 1500, 2000) # hyperparameter values
)

nrow(hyper_gridXG ) # 36 possibilities
```

```
## [1] 36
```

```r
hyper_gridXG
```

```
##    eta max_depth nrounds
## 1  0.3         5     500
## 2  1.0         5     500
## 3  2.0         5     500
## 4  0.3         6     500
## 5  1.0         6     500
## 6  2.0         6     500
## 7  0.3         7     500
## 8  1.0         7     500
## 9  2.0         7     500
## 10 0.3         5    1000
## 11 1.0         5    1000
## 12 2.0         5    1000
## 13 0.3         6    1000
## 14 1.0         6    1000
## 15 2.0         6    1000
## 16 0.3         7    1000
## 17 1.0         7    1000
## 18 2.0         7    1000
## 19 0.3         5    1500
## 20 1.0         5    1500
## 21 2.0         5    1500
## 22 0.3         6    1500
## 23 1.0         6    1500
## 24 2.0         6    1500
## 25 0.3         7    1500
## 26 1.0         7    1500
## 27 2.0         7    1500
## 28 0.3         5    2000
```

```
## 29 1.0         5    2000
## 30 2.0         5    2000
## 31 0.3         6    2000
## 32 1.0         6    2000
## 33 2.0         6    2000
## 34 0.3         7    2000
## 35 1.0         7    2000
## 36 2.0         7    2000

dt <- data.frame()

for(j in 1:nrow(hyper_gridXG )){
x <- Function_CV2XG(input=ttt, dt=dt, k=5, DV=y2, ML=xgboost,  eta =
hyper_gridXG $eta [j], nrounds = hyper_gridXG $nrounds [j],
  max_depth = hyper_gridXG $ max_depth[j])
dt[j,1] <- mean(x[ , 1])
dt[j,2] <- mean(x[ , 2])
dt[j,3] <- mean(x[ , 3])
dt[j,4] <- hyper_gridXG $eta [j]
dt[j,5] <- hyper_gridXG $nrounds [j]
dt[j,6] <- hyper_gridXG $ max_depth[j]
colnames(dt)[1] <- "CV_Accuracy"
colnames(dt)[2] <- "CV_Avg. Balanced Accuracy"
colnames(dt)[3] <- "CV_Avg. F1"
colnames(dt)[4] <- "Eta"
colnames(dt)[5] <- "Nrounds"
colnames(dt)[6] <- "MaxDepth"
}

## [1]  train-error:0.455882
## Will train until train_error hasn't improved in 100 rounds.
##


##
##            Actual
## Predictions  0  1
##           0 30 17
##           1 17 21
## [1]  train-error:0.232353
## Will train until train_error hasn't improved in 100 rounds.
##
##
##            Actual
## Predictions  0  1
##           0 28 12
##           1 12 33

colMeans(XGBoost_res2[ , c(1, 2, 3)]) #the best result
```

```
##                Accuracy Avg. Balanced Accuracy                Avg. F1
##               0.6847059               0.6921011              0.6814743
```

```
#let's tune the hyperparameters for the GLMNET

hyper_gridGlmnet <- expand.grid(
  alpha = seq(0, 1, by = 0.1)
)

hyper_gridGlmnet
```

```
##    alpha
## 1    0.0
## 2    0.1
## 3    0.2
## 4    0.3
## 5    0.4
## 6    0.5
## 7    0.6
## 8    0.7
## 9    0.8
## 10   0.9
## 11   1.0
```

```
dt <- data.frame()

for(j in 1:nrow(hyper_gridGlmnet )){
x <- Function_CV2Glmnet(input=ttt, dt=data2, DV=y, ML=cv.glmnet,
alpha=hyper_gridGlmnet $alpha[j], foldid=folds$which)
dt[j,1] <- mean(x[ , 1])
dt[j,2] <- mean(x[ , 2])
dt[j,3] <- mean(x[ , 3])
dt[j,4] <- hyper_gridGlmnet $alpha[j]
colnames(dt)[1] <- "CV_Accuracy"
colnames(dt)[2] <- "CV_Avg. Balanced Accuracy"
colnames(dt)[3] <- "CV_Avg. F1"
colnames(dt)[4] <- "Alpha"
}
```

```
## Training
##    |
##           True
## Predicted  negative positive Total
##   negative      127       50   177
##   positive       73      175   248
```

```
##   Total         200      225   425
##
##  Percent Correct:  0.7106
```

```r
dt
```

```
##    CV_Accuracy CV_Avg. Balanced Accuracy CV_Avg. F1 Alpha
## 1    0.7552941                 0.7519444  0.7527632   0.0
## 2    0.7482353                 0.7444444  0.7452509   0.1
## 3    0.7364706                 0.7333333  0.7339950   0.2
## 4    0.7317647                 0.7291667  0.7297162   0.3
## 5    0.7223529                 0.7202778  0.7206688   0.4
## 6    0.7152941                 0.7130556  0.7134603   0.5
## 7    0.7082353                 0.7061111  0.7064656   0.6
## 8    0.7082353                 0.7055556  0.7060071   0.7
## 9    0.7058824                 0.7030556  0.7035121   0.8
## 10   0.7082353                 0.7058333  0.7062430   0.9
## 11   0.7105882                 0.7063889  0.7068489   1.0
```

```r
head(arrange(dt, -CV_Accuracy )) # the ridge model (alpha=0) appears to be
the best one
```

```
##   CV_Accuracy CV_Avg. Balanced Accuracy CV_Avg. F1 Alpha
## 1   0.7552941                 0.7519444  0.7527632   0.0
## 2   0.7482353                 0.7444444  0.7452509   0.1
## 3   0.7364706                 0.7333333  0.7339950   0.2
## 4   0.7317647                 0.7291667  0.7297162   0.3
## 5   0.7223529                 0.7202778  0.7206688   0.4
## 6   0.7152941                 0.7130556  0.7134603   0.5
```

```r
# Predicting the test-set

#let's create the DfM for the test-set

x_test <- read.csv("test_review23.csv", stringsAsFactors=FALSE)
str(x_test)
```

```
## 'data.frame':    1080 obs. of  3 variables:
##  $ X        : chr  "cv002_17424.txt" "cv003_12683.txt" "cv004_12641.txt"
"cv005_29357.txt" ...
##  $ text     : chr  "it is movies like these that make a jaded movie viewer
thankful for the invention of the timex indiglo watch . "| __truncated__ " \"
quest for camelot \" is warner bros . ' first feature-length , fully-animated
attempt to steal clout from d"| __truncated__ "synopsis : a mentally unstable
man undergoing psychotherapy saves a boy from a potentially fatal accident
and t"| __truncated__ "capsule : in 2176 on the planet mars police taking
into custody an accused murderer face the title menace . \nt"| __truncated__
...
##  $ Sentiment: logi  NA NA NA NA NA NA ...
```

```
myCorpusTwitterTest <- corpus(x_test)
tok_t <- tokens(myCorpusTwitterTest , remove_punct = TRUE,
remove_numbers=TRUE, remove_symbols = TRUE, split_hyphens = TRUE,
remove_separators = TRUE, remove_URL = TRUE)

## Warning: remove_URL argument is not used.

tok_t <- tokens_remove(tok_t, stopwords("en"))
tok_t <- tokens_remove(tok_t, c("0*"))
tok_t <- tokens_wordstem (tok_t)
Dfm_test <- dfm(tok_t)
Dfm_test<- dfm_trim(Dfm_test, min_docfreq = 2, verbose=TRUE)

## Removing features occurring:

##   - in fewer than 2 documents: 7,878

##   Total features removed: 7,878 (39.4%).

Dfm_test<- dfm_remove(Dfm_test, min_nchar = 2)
Dfm_test <- Dfm_test[ntoken(Dfm_test) != 0,]
str(Dfm_test)

## Formal class 'dfm' [package "quanteda"] with 8 slots
##   ..@ docvars :'data.frame': 1080 obs. of  5 variables:
##   .. ..$ docname_ : chr [1:1080] "text1" "text2" "text3" "text4" ...
##   .. ..$ docid_   : Factor w/ 1080 levels "text1","text2",..: 1 2 3 4 5 6
7 8 9 10 ...
##   .. ..$ segid_   : int [1:1080] 1 1 1 1 1 1 1 1 1 1 ...
##   .. ..$ X        : chr [1:1080] "cv002_17424.txt" "cv003_12683.txt"
"cv004_12641.txt" "cv005_29357.txt" ...
##   .. ..$ Sentiment: logi [1:1080] NA NA NA NA NA NA ...
##   ..@ meta    :List of 3
##   .. ..$ system:List of 5
##   .. .. ..$ package-version:Classes 'package_version', 'numeric_version'
hidden list of 1
##   .. .. .. ..$ : int [1:3] 3 3 1
##   .. .. ..$ r-version      :Classes 'R_system_version', 'package_version',
'numeric_version'  hidden list of 1
##   .. .. .. ..$ : int [1:3] 4 3 1
##   .. .. ..$ system         : Named chr [1:3] "Windows" "x86-64" "Miras"
##   .. .. .. ..- attr(*, "names")= chr [1:3] "sysname" "machine" "user"
##   .. .. ..$ directory      : chr
"C:/Users/Miras/Desktop/u_m/1st/big_data_analytics/Labs/projects"
##   .. .. ..$ created        : Date[1:1], format: "2023-11-27"
##   .. ..$ object:List of 9
##   .. .. ..$ unit        : chr "documents"
##   .. .. ..$ what        : chr "word"
##   .. .. ..$ ngram       : int 1
##   .. .. ..$ skip        : int 0
##   .. .. ..$ concatenator: chr "_"
```

```
##   .. .. ..$ weight_tf   :List of 3
##   .. .. .. ..$ scheme: chr "count"
##   .. .. .. ..$ base  : NULL
##   .. .. .. ..$ k     : NULL
##   .. .. ..$ weight_df   :List of 5
##   .. .. .. ..$ scheme    : chr "unary"
##   .. .. .. ..$ base      : NULL
##   .. .. .. ..$ c         : NULL
##   .. .. .. ..$ smoothing: NULL
##   .. .. .. ..$ threshold: NULL
##   .. .. ..$ smooth      : num 0
##   .. .. ..$ summary     :List of 2
##   .. .. .. ..$ hash: chr(0)
##   .. .. .. ..$ data: NULL
##   .. ..$ user  : list()
##   ..@ i       : int [1:264137] 0 2 3 4 5 6 7 8 9 11 ...
##   ..@ p       : int [1:12100] 0 908 1754 2546 2556 2730 2849 2898 3358
3554 ...
##   ..@ Dim     : int [1:2] 1080 12099
##   ..@ Dimnames:List of 2
##   .. ..$ docs    : chr [1:1080] "text1" "text2" "text3" "text4" ...
##   .. ..$ features: chr [1:12099] "movi" "like" "make" "jade" ...
##   ..@ x       : num [1:264137] 3 3 2 1 7 3 8 4 5 27 ...
##   ..@ factors : list()
```

```
setequal(featnames(Dfm_train), featnames(Dfm_test))
```

```
## [1] FALSE
```

```
test_dfm  <- dfm_match(Dfm_test, features = featnames(Dfm_train))
setequal(featnames(Dfm_train), featnames(test_dfm ))
```

```
## [1] TRUE
```

```
test <- as(test_dfm, "dgCMatrix")
```

```
# Let's run Regularized regression - RIDGE model with "alpha=0"
```

```
system.time(predicted_ridge <- predict(ROLS, test, type="class"))
```

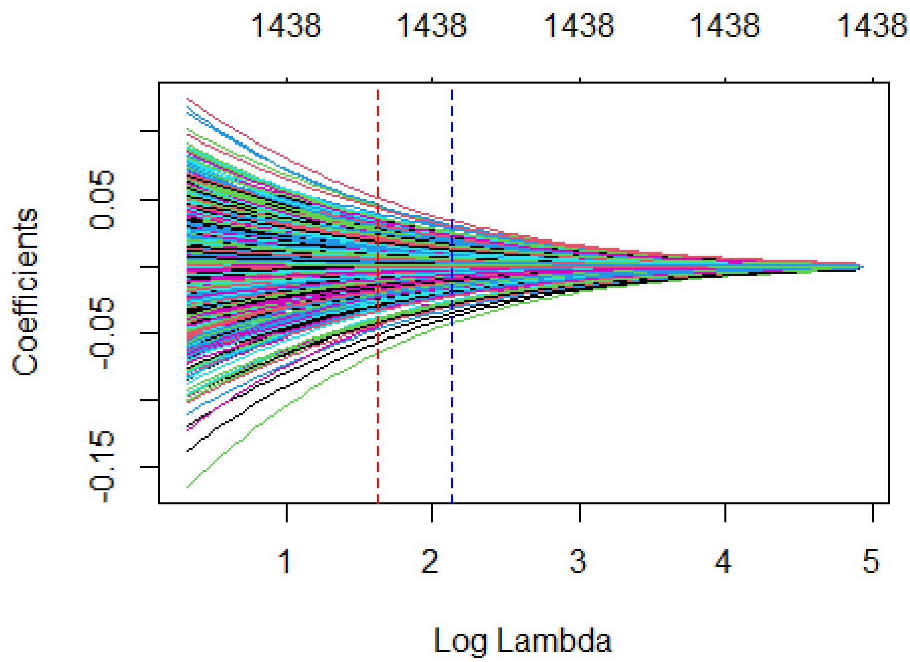
```
##    user  system elapsed 
##    0.03    0.00    0.05
```

```
table(predicted_ridge ) #what's the best lambda though?
```

```
## predicted_ridge
## negative positive 
##    24578    83422
```

```
set.seed(123)
system.time(ridge <-  cv.glmnet(y= Dfm_train@docvars$Sentiment, x=train,
```

```r
    family="binomial", alpha=0, nfolds=10, type.measure="class", trace.it=1
))

## Training
##   |
##    user  system elapsed
##    1.87    0.17    2.13

min(ridge $cvm) # minimum miss-classification error (i.e., 1-accuracy)

## [1] 0.2424242

ridge $lambda.min # lambda for this minimum

## [1] 5.096305

log(ridge $lambda.min )

## [1] 1.628516

ridge $cvm[ridge $lambda == ridge $lambda.1se]  # 1 st.error of minimum miss-
classification error

## [1] 0.2565657

ridge $lambda.1se  # lambda for this error

## [1] 8.501149

log(ridge$lambda.1se)

## [1] 2.140201

lse <- as.numeric(ridge$lambda.1se) # let's save the lambda in this latter
case
lse

## [1] 8.501149

plot(ROLS, xvar = "lambda")
abline(v = log(ridge $lambda.min), col = "red", lty = "dashed")
abline(v = log(ridge $lambda.1se), col = "blue", lty = "dashed")
```




```
# results for lambda.min
system.time(predicted_glment <- predict(ROLS, test, s = ridge $lambda.min,
type="class"))

##    user  system elapsed
##       0       0       0

table(predicted_glment)

## predicted_glment
## negative positive
##      444      636

prop.table(table(predicted_glment))

## predicted_glment
##  negative  positive
## 0.4111111 0.5888889

# same results you get if you predict with the CV for ridge above
system.time(predicted_rr2 <- predict(ridge ,  s= ridge $lambda.min,
test,type="class"))

##    user  system elapsed
##    0.01    0.00    0.02

table(predicted_rr2 )
```

```
## predicted_rr2
## negative positive
##      444      636
```

```r
prop.table(table(predicted_rr2))
```

```
## predicted_rr2
##  negative  positive
## 0.4111111 0.5888889
```

```r
# these are the results when you use lambda.1se
system.time(predicted_glment2 <- predict(ROLS, test, s = lse, type="class"))
```

```
##    user  system elapsed
##    0.02    0.00    0.02
```

```r
table(predicted_glment2)
```

```
## predicted_glment2
## negative positive
##      379      701
```

```r
prop.table(table(predicted_glment2))
```

```
## predicted_glment2
##  negative  positive
## 0.3509259 0.6490741
```