hw3

Jiajian Huang

4/6/2019

Q8 CHP 9

a.

```
library(caret)
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```
## The following object is masked from 'package:mosaic':
##
##     dotPlot
```

```
## The following object is masked from 'package:survival':
##
##     cluster
```

```
set.seed(2019)
train = sample(dim(OJ)[1], 800)
OJ.train = OJ[train, ]
OJ.test = OJ[-train, ]
```

b.

```
library(e1071)
```

```
##
## Attaching package: 'e1071'
```

```
## The following object is masked from 'package:Hmisc':
##
##     impute
```

```
svm.linear <- svm(Purchase ~ ., data = OJ.train, kernel = "linear", cost = 0.01)
summary(svm.linear)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ.train, kernel = "linear",
##     cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##       gamma:  0.05555556
##
## Number of Support Vectors:  441
##
##  ( 222 219 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

From the output of R's summary function we can see that 441 observations are used as support vector. Moreover, the support vectors are almost equally split among the classes.

c.

```
train.pred <- predict(svm.linear, OJ.train)
table(OJ.train$Purchase, train.pred)
```

```
##     train.pred
##       CH  MM
##   CH 447  55
##   MM  80 218
```

```
(80 + 55)/ ( 447+80+55+218)
```

```
## [1] 0.16875
```

The training error rate is 0.1688.

```
test.pred <- predict(svm.linear, OJ.test)
table(OJ.test$Purchase, test.pred)
```

```
##     test.pred
##       CH  MM
##   CH 135  16
##   MM  26  93
```

```
(26+16)/(135+16+26+93)
```

```
## [1] 0.1555556
```

The testing error rate is 0.1556.

d.

```
set.seed(2019)
tune.out = tune(svm, Purchase ~ ., data = OJ.train, kernel = "linear", ranges = list(
cost = 10^seq(-2,
    1, by = 0.25)))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##    10
##
## - best performance: 0.175
##
## - Detailed performance results:
##           cost   error dispersion
## 1   0.01000000 0.17875 0.04084609
## 2   0.01778279 0.17875 0.04126894
## 3   0.03162278 0.17875 0.04291869
## 4   0.05623413 0.18250 0.04133199
## 5   0.10000000 0.18125 0.04419417
## 6   0.17782794 0.18125 0.04419417
## 7   0.31622777 0.18000 0.04495368
## 8   0.56234133 0.17875 0.04041881
## 9   1.00000000 0.17875 0.04041881
## 10  1.77827941 0.18000 0.03827895
## 11  3.16227766 0.18000 0.03827895
## 12  5.62341325 0.17625 0.03606033
## 13 10.00000000 0.17500 0.03435921
```

Tuning shows that optimal cost is 3.162278.

e.

```
svm.linear <- svm(Purchase ~ ., kernel = "linear", data = OJ.train, cost = tune.out$b
est.parameter$cost)
train.pred <- predict(svm.linear, OJ.train)
table(OJ.train$Purchase, train.pred)
```

```
##     train.pred
##       CH  MM
##   CH 441  61
##   MM  75 223
```

```
(75+61)/(441 + 61+ 75+223)
```

```
## [1] 0.17
```

The training error rate is now 17%

```
test.pred <- predict(svm.linear, OJ.test)
table(OJ.test$Purchase, test.pred)
```

```
##     test.pred
##       CH  MM
##   CH 135  16
##   MM  25  94
```

```
(25+16)/(135+25+16+94)
```

```
## [1] 0.1518519
```

The test error rate is 15.18%.

f.

```
svm.radial <- svm(Purchase ~ ., kernel = "radial", data = OJ.train)
summary(svm.radial)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ.train, kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##       gamma:  0.05555556
##
## Number of Support Vectors:  368
##
##  ( 185 183 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
train.pred <- predict(svm.radial, OJ.train)
table(OJ.train$Purchase, train.pred)
```

```
##     train.pred
##       CH  MM
##   CH 465  37
##   MM  80 218
```

```
(31+18)/(133+18+31+88)
```

```
## [1] 0.1814815
```

```
test.pred <- predict(svm.radial, OJ.test)
table(OJ.test$Purchase, test.pred)
```

```
##     test.pred
##       CH  MM
##   CH 133  18
##   MM  31  88
```

```
(80+37)/(465+37+80+218)
```

```
## [1] 0.14625
```

The classifier has a training error of 14.63% and a test error of 18.15%.

```
set.seed(2019)
tune.out <- tune(svm, Purchase ~ ., data = OJ.train, kernel = "radial", ranges = list
(cost = 10^seq(-2,
    1, by = 0.25)))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 0.17125
##
## - Detailed performance results:
##           cost   error dispersion
## 1   0.01000000 0.37250 0.06368324
## 2   0.01778279 0.37250 0.06368324
## 3   0.03162278 0.36375 0.07008180
## 4   0.05623413 0.21750 0.04338138
## 5   0.10000000 0.17875 0.05560588
## 6   0.17782794 0.17125 0.04715886
## 7   0.31622777 0.17375 0.04059026
## 8   0.56234133 0.17375 0.04143687
## 9   1.00000000 0.17125 0.04450733
## 10  1.77827941 0.17250 0.04401704
## 11  3.16227766 0.17750 0.04158325
## 12  5.62341325 0.17375 0.04185375
## 13 10.00000000 0.18125 0.04497299
```

Tuning shows that optimal cost is 0.17.

g.

```
svm.poly <- svm(Purchase ~ ., kernel = "polynomial", data = OJ.train, degree = 2)
summary(svm.poly)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ.train, kernel = "polynomial",
##     degree = 2)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  1
##      degree:  2
##       gamma:  0.05555556
##      coef.0:  0
##
## Number of Support Vectors:  435
##
##  ( 222 213 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
train.pred <- predict(svm.poly, OJ.train)
table(OJ.train$Purchase, train.pred)
```

```
##     train.pred
##       CH  MM
##   CH 470  32
##   MM 109 189
```

```
(109+32)/(470+32+109+189)
```

```
## [1] 0.17625
```

```
test.pred <- predict(svm.poly, OJ.test)
table(OJ.test$Purchase, test.pred)
```

```
##     test.pred
##       CH  MM
##   CH 134  17
##   MM  41  78
```

```
(41+17)/(134+17+41+78)
```

```
## [1] 0.2148148
```

With polynomial kernel degree=2, the classifier has a training error of 17.63% and a test error of 21.48%.

```
set.seed(2019)
tune.out <- tune(svm, Purchase ~ ., data = OJ.train, kernel = "polynomial", degree =
2, ranges = list(cost = 10^seq(-2,
    1, by = 0.25)))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##    10
##
## - best performance: 0.18125
##
## - Detailed performance results:
##           cost   error dispersion
## 1   0.01000000 0.37125 0.06402311
## 2   0.01778279 0.35125 0.05905800
## 3   0.03162278 0.34250 0.06671873
## 4   0.05623413 0.31000 0.06609127
## 5   0.10000000 0.29000 0.06687468
## 6   0.17782794 0.25000 0.06614378
## 7   0.31622777 0.20875 0.03821086
## 8   0.56234133 0.19625 0.04332131
## 9   1.00000000 0.19500 0.03343734
## 10  1.77827941 0.18500 0.04594683
## 11  3.16227766 0.18375 0.03488573
## 12  5.62341325 0.18625 0.04016027
## 13 10.00000000 0.18125 0.04379958
```

Tuning shows that optimal cost is 0.17.

```
svm.poly <- svm(Purchase ~ ., kernel = "polynomial", degree = 2, data = OJ.train, cos
t = tune.out$best.parameter$cost)
summary(svm.poly)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ.train, kernel = "polynomial",
##     degree = 2, cost = tune.out$best.parameter$cost)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  10
##      degree:  2
##       gamma:  0.05555556
##      coef.0:  0
##
## Number of Support Vectors:  336
##
##  ( 173 163 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
train.pred <- predict(svm.poly, OJ.train)
table(OJ.train$Purchase, train.pred)
```

```
##     train.pred
##       CH  MM
##   CH 466  36
##   MM  79 219
```

```
(79+36)/(466+36+79+219)
```

```
## [1] 0.14375
```

```
test.pred <- predict(svm.poly, OJ.test)
table(OJ.test$Purchase, test.pred)
```

```
##     test.pred
##       CH  MM
##   CH 131  20
##   MM  33  86
```

```
(33+20)/(131+20+33+86)
```

```
## [1] 0.1962963
```

The classifier has a training error of 14.38% and a test error of 19.63%.

h.

The default gamma approach provides the best results on this data.

Dataset Khan

random forests

```
data("Khan")
```

```
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
```

```
## The following object is masked from 'package:gridExtra':
##
##     combine
```

```
## The following object is masked from 'package:imager':
##
##     grow
```

```
dim(Khan$xtrain)
```

```
## [1]   63 2308
```

```
dim(Khan$xtest)
```

```
## [1]   20 2308
```

```
table(Khan$ytrain)
```

```
##
##  1  2  3  4
##  8 23 12 20
```

```
table(Khan$ytest)
```

```
##
## 1 2 3 4
## 3 6 6 5
```

```
khan_train = data.frame(x = Khan$xtrain, y = as.factor(Khan$ytrain))
khan_test = data.frame(x = Khan$xtest, y = as.factor(Khan$ytest))
```

```
random = train(y ~ ., data = khan_train, method = "rf", trControl = trainControl(meth
od = "cv"))
```

```
confusionMatrix(khan_train$y, predict(random, khan_train))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3  4
##          1  8  0  0  0
##          2  0 23  0  0
##          3  0  0 12  0
##          4  0  0  0 20
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.9431, 1)
##     No Information Rate : 0.3651
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4
## Sensitivity             1.000   1.0000   1.0000   1.0000
## Specificity             1.000   1.0000   1.0000   1.0000
## Pos Pred Value          1.000   1.0000   1.0000   1.0000
## Neg Pred Value          1.000   1.0000   1.0000   1.0000
## Prevalence              0.127   0.3651   0.1905   0.3175
## Detection Rate          0.127   0.3651   0.1905   0.3175
## Detection Prevalence    0.127   0.3651   0.1905   0.3175
## Balanced Accuracy       1.000   1.0000   1.0000   1.0000
```

```
confusionMatrix(khan_test$y, predict(random, khan_test))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction 1 2 3 4
##          1 3 0 0 0
##          2 0 6 0 0
##          3 0 1 5 0
##          4 0 0 0 5
##
## Overall Statistics
##
##                Accuracy : 0.95
##                  95% CI : (0.7513, 0.9987)
##     No Information Rate : 0.35
##     P-Value [Acc > NIR] : 2.903e-08
##
##                   Kappa : 0.932
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4
## Sensitivity              1.00   0.8571   1.0000     1.00
## Specificity              1.00   1.0000   0.9333     1.00
## Pos Pred Value           1.00   1.0000   0.8333     1.00
## Neg Pred Value           1.00   0.9286   1.0000     1.00
## Prevalence               0.15   0.3500   0.2500     0.25
## Detection Rate           0.15   0.3000   0.2500     0.25
## Detection Prevalence     0.15   0.3000   0.3000     0.25
## Balanced Accuracy        1.00   0.9286   0.9667     1.00
```

tree boosting

```
library(xgboost)
```

```
##
## Attaching package: 'xgboost'
```

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

```
library(gbm)
```

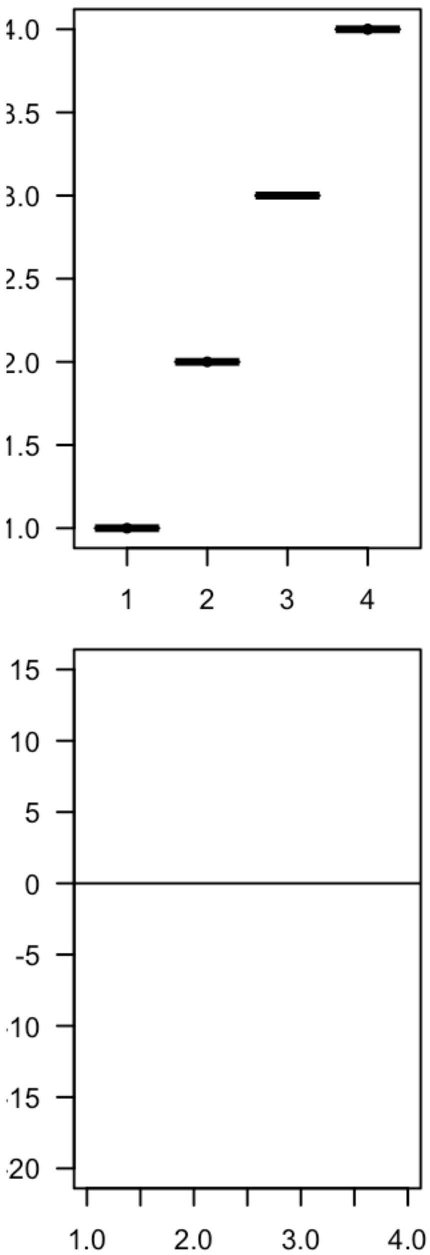
```
## Loaded gbm 2.1.5
```

```
set.seed(2019)
boost.khan = gbm(y ~., data = khan_train, distribution = "gaussian", n.trees = 5000)
boost.khan
```

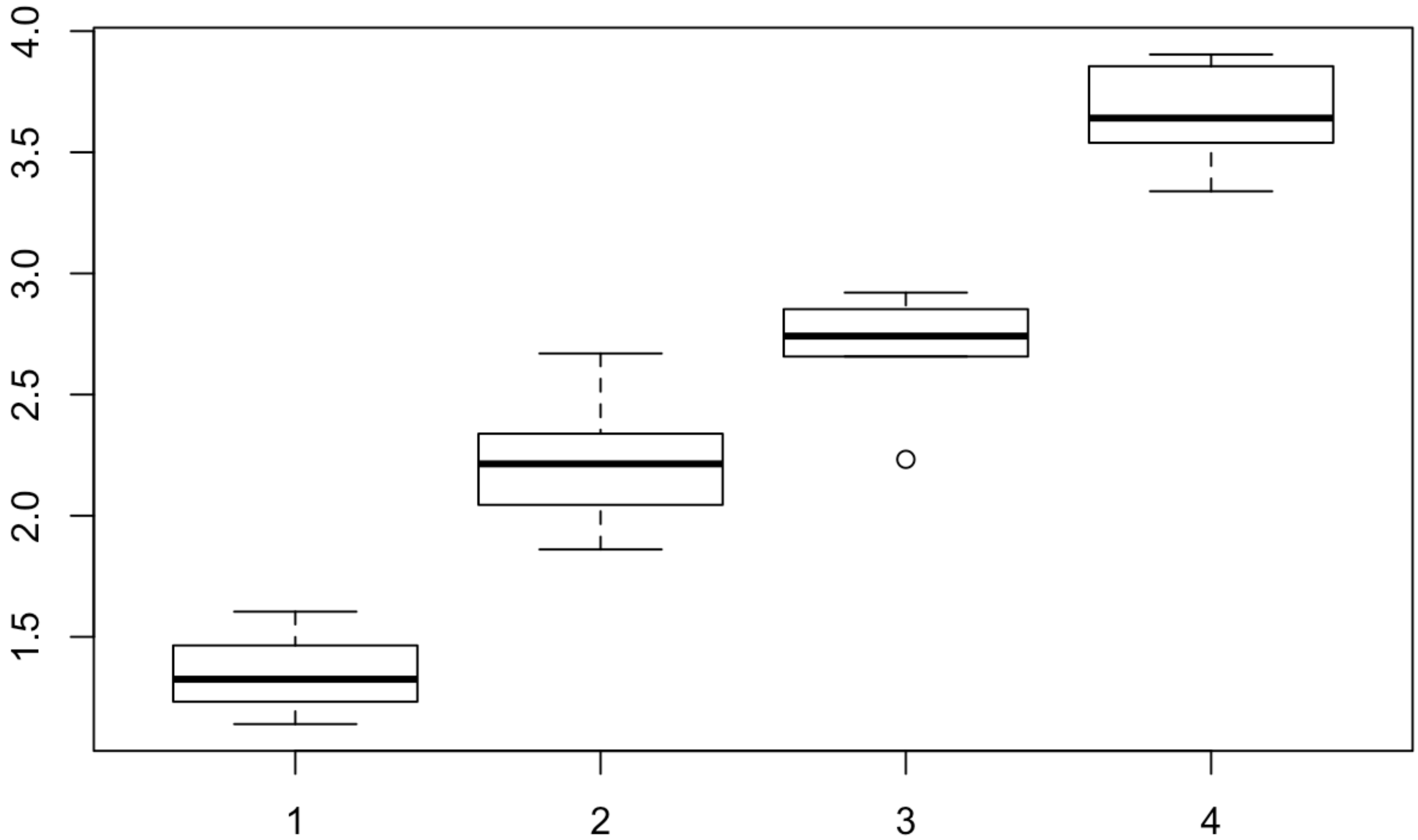
```
## gbm(formula = y ~ ., distribution = "gaussian", data = khan_train,
##     n.trees = 5000)
## A gradient boosted model with gaussian loss function.
## 5000 iterations were performed.
## There were 2308 predictors of which 1845 had non-zero influence.
```

```
par(mfcol = c(2, 4), mar = c(2, 2, 1, 1), las = 1)
plot(khan_train$y, predict(boost.khan, n.trees = 5000))
plot(khan_train$y, predict(boost.khan, n.trees = 5000) - khan_train$y, ylim = c(-20,
15));abline(h = 0)
```

```
## Warning in Ops.factor(predict(boost.khan, n.trees = 5000), khan_train$y):
## '-' not meaningful for factors
```

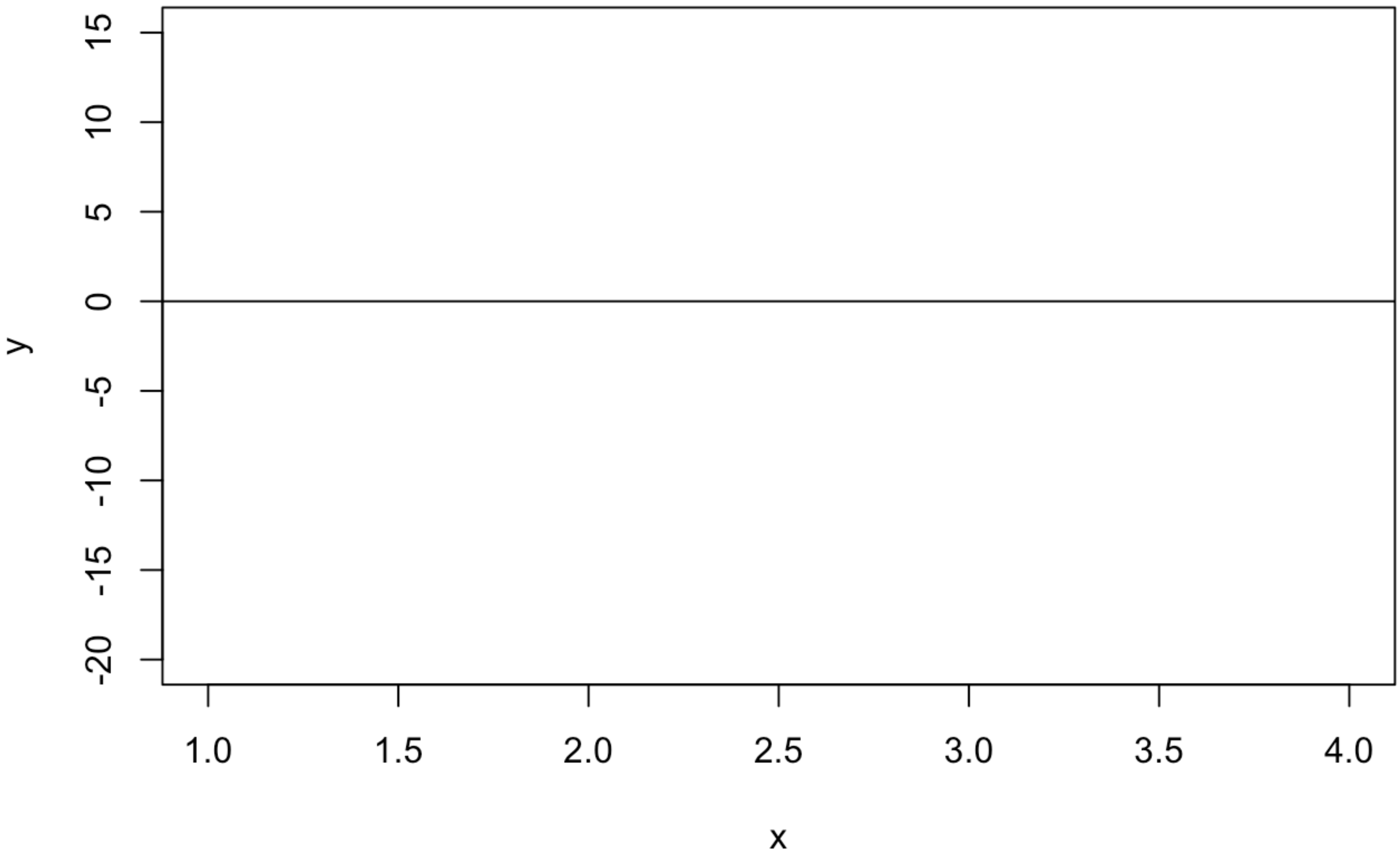



```
plot(khan_test$y, predict(boost.khan, khan_test, n.tree = 500))
```




```
plot(khan_test$y, predict(boost.khan, khan_test, n.trees = 500) - khan_test$y, ylim =
c(-20, 15));abline(h = 0)
```

```
## Warning in Ops.factor(predict(boost.khan, khan_test, n.trees = 500),
## khan_test$y): '-' not meaningful for factors
```




```
set.seed(2019)
boost.han = gbm(y ~., data = khan_train, distribution = "gaussian", n.trees = 5000, i
nteraction.depth = 4)
```

```
yhat.bt1.nn = predict(boost.khan, khan_test, n.trees = seq(500, 5000, 500))
dim(yhat.bt1.nn)
```

```
## [1] 20 10
```

```
a = apply(yhat.bt1.nn, 2, function(x) mean(Khan$ytest - x)^2)
round(a, 5)
```

```
##     500    1000    1500    2000    2500    3000    3500    4000    4500
## 0.00340 0.00353 0.00353 0.00353 0.00353 0.00353 0.00353 0.00353 0.00353
##    5000
## 0.00353
```

```
boost.x = xgboost(data = Khan$xtrain, label = Khan$ytrain, max_depth = 4, eta = 1, nr
ound = 5, objective = "reg:linear")
```

```
## [1]  train-rmse:0.386910
## [2]  train-rmse:0.090095
## [3]  train-rmse:0.032245
## [4]  train-rmse:0.009076
## [5]  train-rmse:0.003345
```

```
all.equal(as.numeric(predict(boost.x, Khan$xtest) > 0.5), Khan$ytest)
```

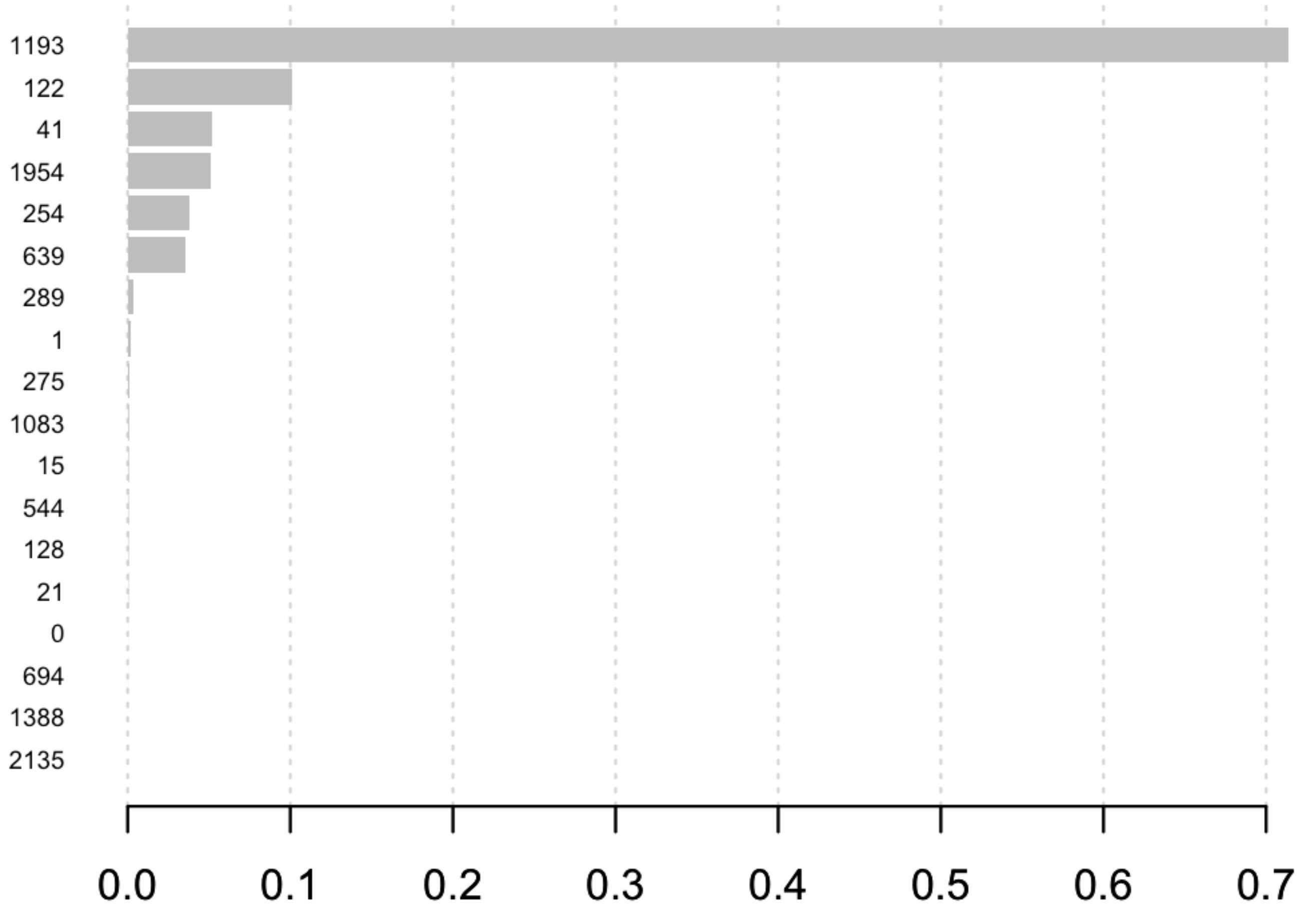
```
## [1] "Mean relative difference: 1.941176"
```

```
boost.y = xgboost(data = Khan$xtrain, label = Khan$ytrain, max_depth = 1, eta = 0.1,
nround = 50, objective = "reg:linear")
```

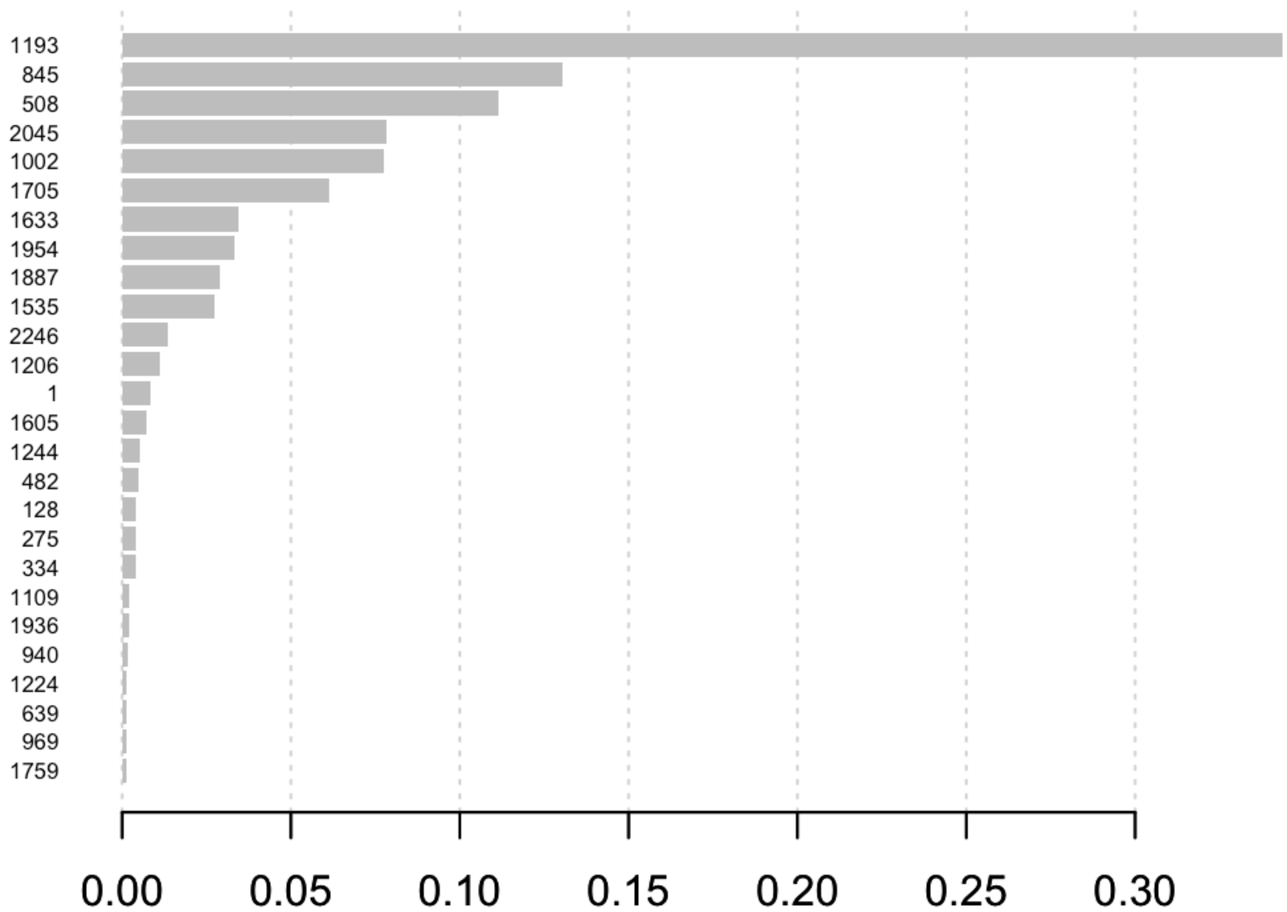
```
## [1]  train-rmse:2.213275
## [2]  train-rmse:2.013805
## [3]  train-rmse:1.835033
## [4]  train-rmse:1.674052
## [5]  train-rmse:1.529896
## [6]  train-rmse:1.399689
## [7]  train-rmse:1.281420
## [8]  train-rmse:1.174630
## [9]  train-rmse:1.078368
## [10] train-rmse:0.991666
## [11] train-rmse:0.912312
## [12] train-rmse:0.841279
## [13] train-rmse:0.776592
## [14] train-rmse:0.718592
## [15] train-rmse:0.665687
## [16] train-rmse:0.617682
## [17] train-rmse:0.574163
## [18] train-rmse:0.534266
## [19] train-rmse:0.498723
## [20] train-rmse:0.466249
## [21] train-rmse:0.437040
## [22] train-rmse:0.410511
## [23] train-rmse:0.386137
## [24] train-rmse:0.363850
## [25] train-rmse:0.343359
## [26] train-rmse:0.324529
## [27] train-rmse:0.307804
## [28] train-rmse:0.292361
## [29] train-rmse:0.278201
## [30] train-rmse:0.265033
```

```
## [31] train-rmse:0.253135
## [32] train-rmse:0.242135
## [33] train-rmse:0.231865
## [34] train-rmse:0.222444
## [35] train-rmse:0.213813
## [36] train-rmse:0.205722
## [37] train-rmse:0.198237
## [38] train-rmse:0.191119
## [39] train-rmse:0.184456
## [40] train-rmse:0.178209
## [41] train-rmse:0.172499
## [42] train-rmse:0.167112
## [43] train-rmse:0.161845
## [44] train-rmse:0.157029
## [45] train-rmse:0.152446
## [46] train-rmse:0.148043
## [47] train-rmse:0.143891
## [48] train-rmse:0.140015
## [49] train-rmse:0.136257
## [50] train-rmse:0.132729
```

```
importance_matrix = xgb.importance(model = boost.x)
xgb.plot.importance(importance_matrix = importance_matrix)
```




```
xgb.plot.importance(importance_matrix = xgb.importance(model =boost.y))
```



1193
845
508
2045
1002
1705
1633
1954
1887
1535
2246
1206
1
1605
1244
482
128
275
334
1109
1936
940
1224
639
969
1759
0.00
0.05
0.10
0.15
0.20
0.25
0.30