

Wine-Quality-Prediction.R

tomaz

2024-04-18

```
## Wine Quality Prediction using Random Forest, XGBoost, and Decision Trees
# load the necessary libraries
library(caTools)
library(mice)
```

```
## Warning: package 'mice' was built under R version 4.3.3
```

```
##
## Attaching package: 'mice'
```

```
## The following object is masked from 'package:stats':
##
## filter
```

```
## The following objects are masked from 'package:base':
##
## cbind, rbind
```

```
library(ggplot2)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
## filter, lag
```

```
## The following objects are masked from 'package:base':
##
## intersect, setdiff, setequal, union
```

```
library(corrgram)
library(corrplot)
```

```
## corrplot 0.92 loaded
```

```
library(randomForest)
```

```
## randomForest 4.7-1.1
```

```
## 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
```

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.3.3
```

```
## Zorunlu paket yükleniyor: lattice
```

```
##  
## Attaching package: 'lattice'
```

```
## The following object is masked from 'package:corrgram':  
##  
## panel.fill
```

```
library(vip)
```

```
## Warning: package 'vip' was built under R version 4.3.3
```

```
##  
## Attaching package: 'vip'
```

```
## The following object is masked from 'package:utils':  
##  
## vi
```

```
library(class)  
library(rpart)  
library(rpart.plot)  
library(xgboost)
```

```
## Warning: package 'xgboost' was built under R version 4.3.3
```

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

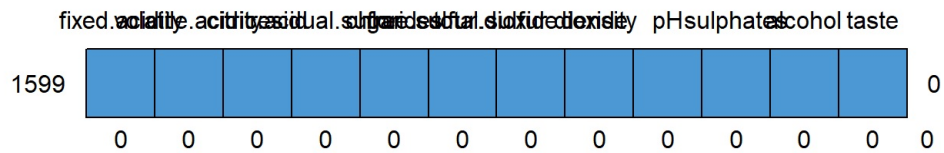
```
## The following object is masked from 'package:dplyr':  
##  
## slice
```

```
# load the data set  
df <- read.csv("wine.csv")  
  
# str of the df & correcttting the classes  
str(df)
```

```
## 'data.frame': 1599 obs. of 13 variables:  
## $ fixed.acidity : num 7.4 7.8 7.8 11.2 7.4 7.4 7.9 7.3 7.8 7.5 ...  
## $ volatile.acidity : num 0.7 0.88 0.76 0.28 0.7 0.66 0.6 0.65 0.58 0.5 ...  
## $ citric.acid : num 0 0 0.04 0.56 0 0 0.06 0 0.02 0.36 ...  
## $ residual.sugar : num 1.9 2.6 2.3 1.9 1.9 1.8 1.6 1.2 2 6.1 ...  
## $ chlorides : num 0.076 0.098 0.092 0.075 0.076 0.075 0.069 0.065 0.073 0.071 ...  
## $ free.sulfur.dioxide : num 11 25 15 17 11 13 15 15 9 17 ...  
## $ total.sulfur.dioxide : num 34 67 54 60 34 40 59 21 18 102 ...  
## $ density : num 0.998 0.997 0.997 0.998 0.998 ...  
## $ pH : num 3.51 3.2 3.26 3.16 3.51 3.51 3.3 3.39 3.36 3.35 ...  
## $ sulphates : num 0.56 0.68 0.65 0.58 0.56 0.56 0.46 0.47 0.57 0.8 ...  
## $ alcohol : num 9.4 9.8 9.8 9.8 9.4 9.4 9.4 10 9.5 10.5 ...  
## $ quality : int 5 5 5 6 5 5 5 7 7 5 ...  
## $ taste : chr "normal" "normal" "normal" "normal" ...
```

```
df$taste <- as.factor(df$taste)  
  
# removing the quality column due to its strong correlation with taste  
df <- df[, -12]  
  
# check the missing values  
md.pattern(df) # completely observed
```

```
## /\ /\
## { '---' }
## { 0 0 }
## ==> V <== No need for mice. This data set is completely observed.
## \ \ / /
## \-----'
```

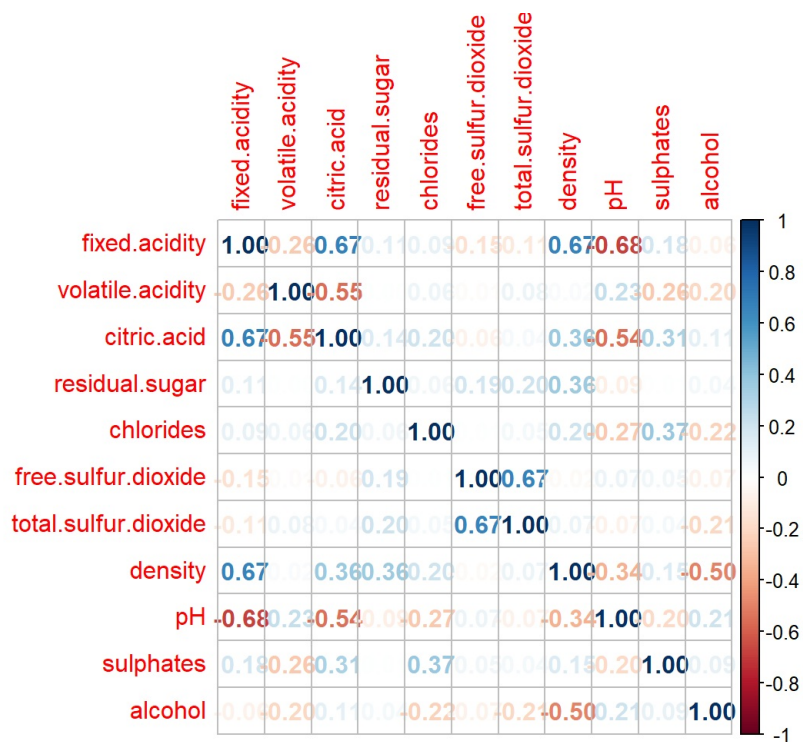


```
##      fixed.acidity volatile.acidity citric.acid residual.sugar chlorides
## 1599          1              1          1          1          1
##           0              0          0          0          0
##      free.sulfur.dioxide total.sulfur.dioxide density pH sulphates alcohol
## 1599          1              1          1  1          1          1
##           0              0          0  0          0          0
##      taste
## 1599          1  0
##           0  0
```

```
# check the taste col
table(df$taste) # upon examining the confusion matrices in my dataset, it appears that sensitivity and specificity values may be unavailable (NA) due to the dataset's lack of a balanced distribution.
```

```
##
##      bad    good normal
##      63     217   1319
```

```
# correlation graph
corr.matrix <- cor(df[, -12])
corrplot(corr.matrix, method = "number")
```



```
# test & train split
split <- sample.split(df$taste, SplitRatio = 0.8)
train <- subset(df, split == TRUE)
test <- subset(df, split == FALSE)
```

```
# dim of the test set
dim(test)
```

```
## [1] 320 12
```

```
# dim of the train set
dim(train)
```

```
## [1] 1279 12
```

```
## RANDOM FOREST
# creating parameter grid for random forest model
param_grid <- expand.grid(mtry = c(1, 2, 3, 4)) # values to try for mtry

# setting cross-validation parameters with the control function
control <- trainControl(method = "cv", number = 5) # 5 fold cross-validation

# select the best parameters for random forest
rf.model <- train(taste~.,
                  data = train,
                  method = "rf",
                  trControl = control,
                  tuneGrid = param_grid)

# best tune
rf.model$bestTune
```

```
## mtry
## 2 2
```

```
# building a rf with best parameters
rf.model.param <- randomForest(taste~., train, mtry = rf.model$bestTune$mtry, ntree = 30)
rf.model.param
```

```
##
## Call:
##  randomForest(formula = taste ~ ., data = train, mtry = rf.model$bestTune$mtry,      ntree = 30)
##              Type of random forest: classification
##              Number of trees: 30
## No. of variables tried at each split: 2
##
##              OOB estimate of  error rate: 14.54%
## Confusion matrix:
##              bad good normal class.error
## bad          3    1    46  0.94000000
## good         0   83    91  0.52298851
## normal       8   40   100  0.04549763
```

```
# predictions
rf.preds <- predict(rf.model.param, test)

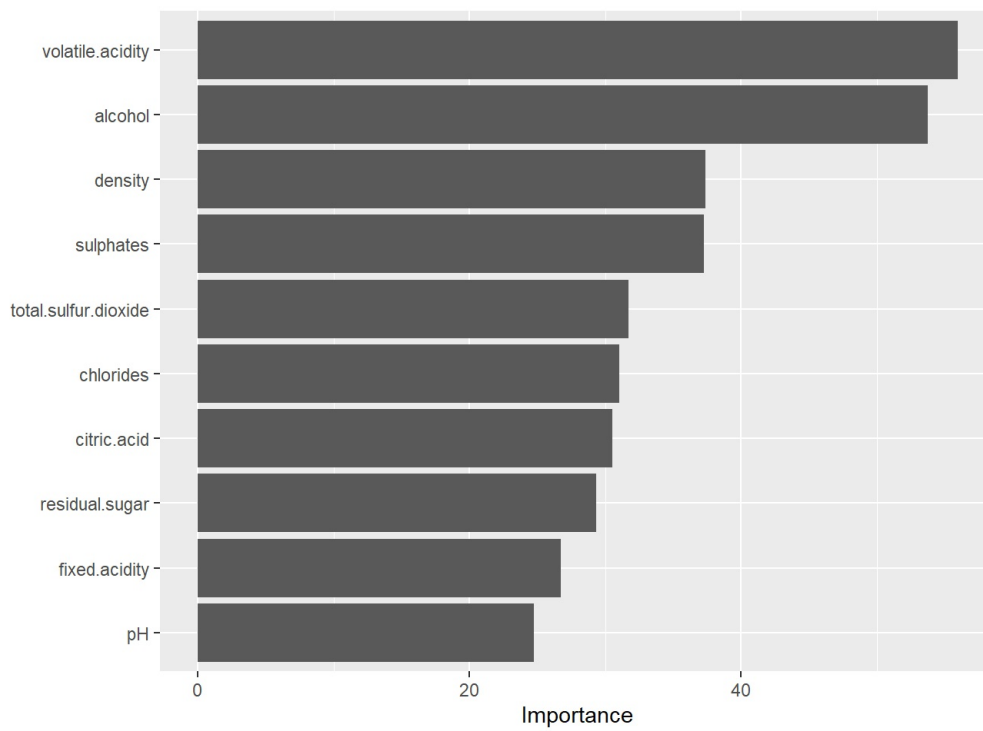
# confusion matrix of the RF model
cm.rf <- confusionMatrix(test$taste, rf.preds)
cm.rf
```

```
## Confusion Matrix and Statistics
##
##              Reference
## Prediction bad good normal
##      bad      0    0    13
##      good      0   21   22
##      normal    0   13  251
##
## Overall Statistics
##
##              Accuracy : 0.85
##              95% CI : (0.8061, 0.8873)
##      No Information Rate : 0.8938
##      P-Value [Acc > NIR] : 0.994
##
##              Kappa : 0.3961
##
##      McNemar's Test P-Value : NA
##
## Statistics by Class:
##
##              Class: bad Class: good Class: normal
## Sensitivity              NA    0.61765    0.8776
## Specificity              0.95937    0.92308    0.6176
## Pos Pred Value              NA    0.48837    0.9508
## Neg Pred Value              NA    0.95307    0.3750
## Prevalence                0.00000    0.10625    0.8938
## Detection Rate              0.00000    0.06563    0.7844
## Detection Prevalence        0.04063    0.13437    0.8250
## Balanced Accuracy              NA    0.77036    0.7476
```

```
cm.rf$byClass
```

```
##              Sensitivity Specificity Pos Pred Value Neg Pred Value Precision
## Class: bad      NA    0.9593750      NA      NA    NA 0.0000000
## Class: good    0.6176471    0.9230769    0.4883721    0.9530686 0.4883721
## Class: normal  0.8776224    0.6176471    0.9507576    0.3750000 0.9507576
##              Recall      F1 Prevalence Detection Rate
## Class: bad      NA      NA    0.00000    0.000000
## Class: good    0.6176471 0.5454545    0.10625    0.065625
## Class: normal  0.8776224 0.9127273    0.89375    0.784375
##              Detection Prevalence Balanced Accuracy
## Class: bad      0.040625      NA
## Class: good      0.134375    0.7703620
## Class: normal    0.825000    0.7476347
```

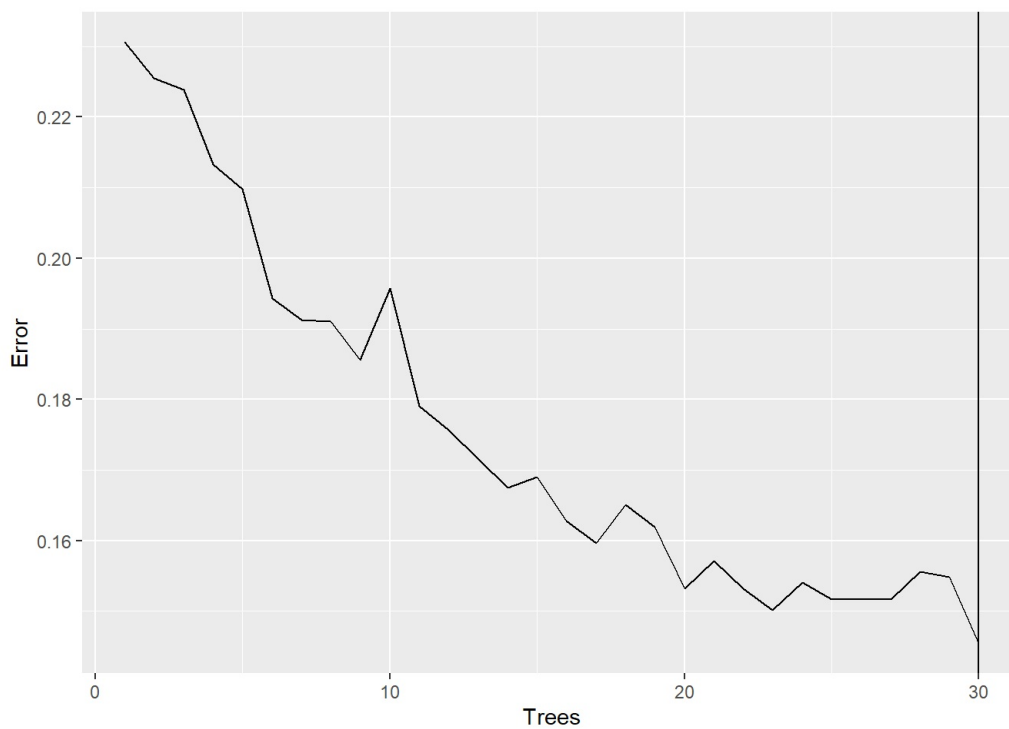
```
# variable importance
vip(rf.model.param)
```



```
# optimum number of tree
error.rate <- data.frame(
  Trees = rep(1:nrow(rf.model.param$err.rate)),
  Error = rf.model.param$err.rate[, "OOB"]
)

# min error
minerrorpts <- error.rate$Trees [error.rate$Error == min(error.rate$Error)]

# plotting to determine optimal number of trees
ggplot(error.rate, aes(x = Trees, y = Error)) +
  geom_line() +
  geom_vline(xintercept = minerrorpts [1])
```



```
## KNN
# test & train set without labels
test.knn <- test [1:11] # test set without label to be used in knn classification
train.knn <- train [1:11] # train set without label to be used in knn classification

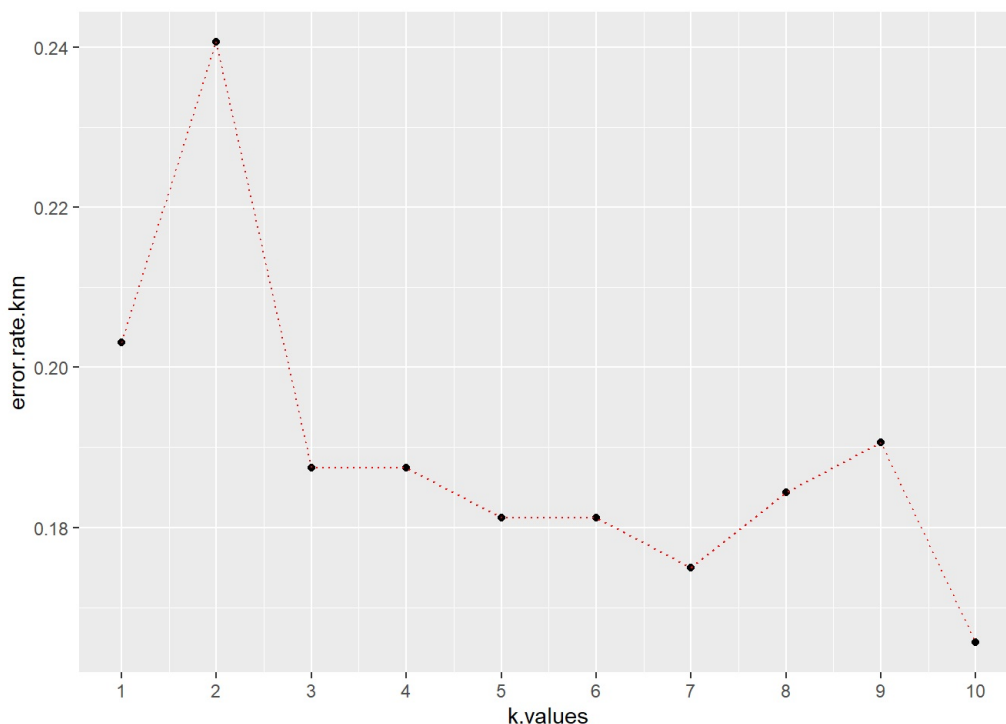
# building a knn model with k = 1
knn.classifier <- knn(train = train.knn, # train set without label used
                      test = test.knn, # test set without label used
                      cl = train$taste, # train set with label used
                      k = 1 )

# choosing the k value
classifier.knn = NULL
error.rate.knn = NULL
for (i in 1:10) {
  classifier.knn <- knn(train = train.knn,
                       test = test.knn,
                       cl = train$taste,
                       k = i)
  error.rate.knn [i] <- mean(test$taste != classifier.knn)
}
print(error.rate.knn)
```

```
## [1] 0.203125 0.240625 0.187500 0.187500 0.181250 0.181250 0.175000 0.184375
## [9] 0.190625 0.165625
```

```
# error rate knn vs values of K
k.values <- 1:10
error.rate.knn.k <- data.frame(error.rate.knn, k.values)

# elbow method to determine optimum k value
ggplot(error.rate.knn.k, aes(x = k.values, y = error.rate.knn)) +
  geom_point() +
  geom_line(lty = "dotted", color = "red") +
  scale_x_continuous(breaks = seq(1, 10, by = 1))
```



```
# confusion matrix of the KNN model
table(knn.classifier, test$taste)
```

```
##
## knn.classifier bad good normal
##      bad      2      0      10
##      good     1     25     26
##      normal  10     18    228
```

```
# error rate of the best KNN model
misclasserror.knn <- mean(test$taste != classifier.knn)
accuracy.knn <- 1 - misclasserror.knn
accuracy.knn
```

```
## [1] 0.834375
```

```
## CLASSIFICATION TREE
# define the parameter grid
param_grid.ct <- expand.grid(
  cp = seq(0.01, 1, by = 0.01))

# control parameters
control.ct <- trainControl(method = "cv", number = 5)

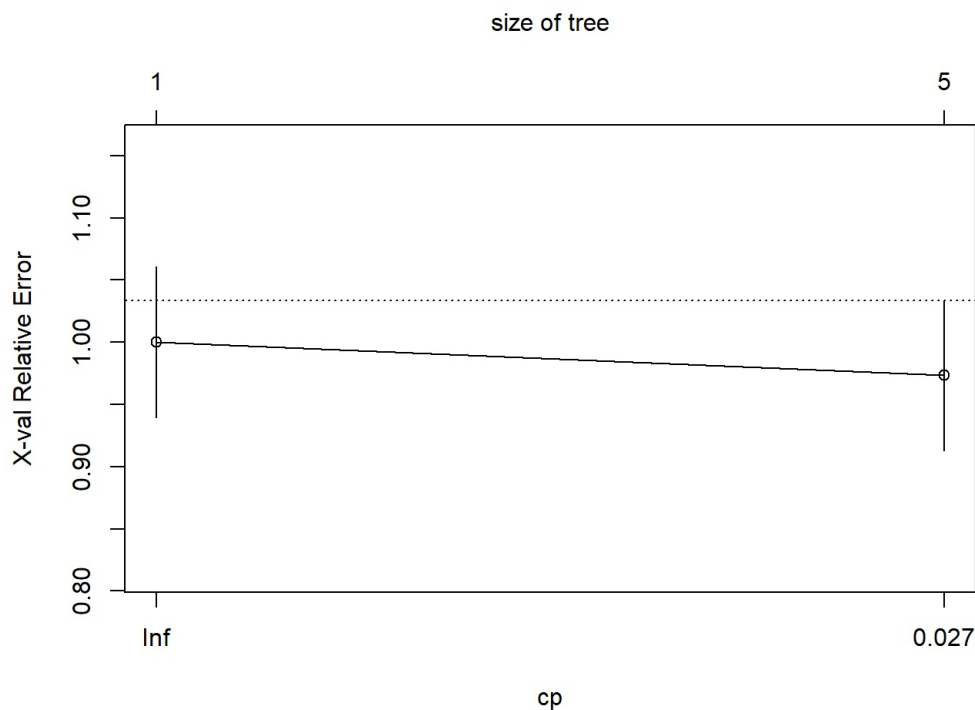
# best parameters via cross validation
ct.model <- train(taste~., data = train,
  method = "rpart",
  trControl = control.ct,
  tuneGrid = param_grid.ct)

# best parameters
ct.model$bestTune
```

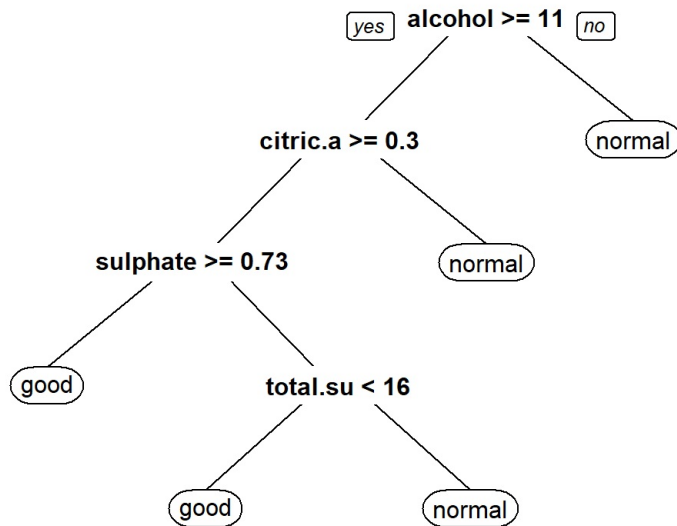
```
##      cp
## 2 0.02
```

```
# building the final model
ct.final.model <- rpart(
  taste~.,
  data = train,
  cp = ct.model$bestTune$cp
)

# Cross-validation Error vs. Complexity Parameter (CP)
plotcp(ct.final.model)
```



```
# classification tree
print(prp(ct.final.model))
```

```

## $obj
## n= 1279
##
## node), split, n, loss, yval, (yprob)
##      * denotes terminal node
##
##  1) root 1279 224 normal (0.03909304 0.13604378 0.82486317)
##    2) alcohol>=11.15 305 122 normal (0.03278689 0.36721311 0.60000000)
##      4) citric.acid>=0.295 179 88 good (0.01117318 0.50837989 0.48044693)
##        8) sulphates>=0.725 74 25 good (0.00000000 0.66216216 0.33783784) *
##          9) sulphates< 0.725 105 44 normal (0.01904762 0.40000000 0.58095238)
##            18) total.sulfur.dioxide< 15.5 36 13 good (0.02777778 0.63888889 0.33333333) *
##              19) total.sulfur.dioxide>=15.5 69 20 normal (0.01449275 0.27536232 0.71014493) *
##        5) citric.acid< 0.295 126 29 normal (0.06349206 0.16666667 0.76984127) *
##      3) alcohol< 11.15 974 102 normal (0.04106776 0.06365503 0.89527721) *
##
## $snipped.nodes
## NULL
##
## $xlim
## [1] -0.2 1.2
##
## $ylim
## [1] -0.1 1.1
##
## $x
## [1] 0.72122580 0.49314628 0.26527204 0.03945073 0.49109336 0.28580125 0.69638546
## [8] 0.72102051 0.94930533
##
## $y
## [1] 0.95182258 0.71730818 0.48279378 0.24827938 0.24827938 0.01376498 0.01376498
## [8] 0.48279378 0.71730818
##
## $branch.x
##      [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## x 0.7212258 0.4931463 0.2652720 0.03945073 0.4910934 0.2858013 0.6963855
##      NA 0.6756099 0.4475714 0.22010778 0.3104363 0.4500349 0.5321518
##      NA 0.7212258 0.4931463 0.26527204 0.2652720 0.4910934 0.4910934
##      [,8]      [,9]
## x 0.7210205 0.9493053
##      0.5387211 0.7668417
##      0.4931463 0.7212258
##
## $branch.y
##      [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## y 0.9576687 0.7231543 0.4886399 0.3011055 0.2541255 0.06659107 0.06659107
##      NA 0.9518226 0.7173082 0.4827938 0.4827938 0.24827938 0.24827938
##      NA 0.9518226 0.7173082 0.4827938 0.4827938 0.24827938 0.24827938
##      [,8]      [,9]
## y 0.5356199 0.7701343
##      0.7173082 0.9518226

```

```
##      0.7173082 0.9518226
##
## $labs
## [1] NA      NA      NA      "good"  NA      "good"  "normal" "normal"
## [9] "normal"
##
## $cex
## [1] 1
##
## $boxes
## $boxes$x1
## [1]      NA      NA      NA -0.01722541      NA 0.22912511
## [7] 0.62240818 0.64704323 0.87532805
##
## $boxes$y1
## [1]      NA      NA      NA 0.23067068      NA -0.00384372
## [7] -0.00384372 0.46518508 0.69969948
##
## $boxes$x2
## [1]      NA      NA      NA 0.09612687      NA 0.34247739 0.77036274
## [8] 0.79499779 1.02328261
##
## $boxes$y2
## [1]      NA      NA      NA 0.30110547      NA 0.06659107 0.06659107
## [8] 0.53561987 0.77013427
##
##
## $split.labs
## [1] ""
##
## $split.cex
## [1] 1 1 1 1 1 1 1 1 1
##
## $split.box
## $split.box$x1
## [1] 0.58281670 0.35294740 0.09941976      NA 0.36282630      NA      NA
## [8]      NA      NA
##
## $split.box$y1
## [1] 0.9342139 0.6996995 0.4651851      NA 0.2306707      NA      NA
## [8]      NA      NA
##
## $split.box$x2
## [1] 0.8596349 0.6333452 0.4311243      NA 0.6193604      NA      NA
## [8]      NA      NA
##
## $split.box$y2
## [1] 1.0046487 0.7701343 0.5356199      NA 0.3011055      NA      NA
## [8]      NA      NA
```

```
# predictions
ct.preds <- predict(ct.final.model, newdata = test, type = "class")

# confusion matrix
confusionMatrix(test$taste, ct.preds)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction bad good normal
##      bad      0      0      13
##      good      0     12     31
##      normal    0     17    247
##
## Overall Statistics
##
##           Accuracy : 0.8094
##           95% CI : (0.762, 0.8509)
##      No Information Rate : 0.9094
##      P-Value [Acc > NIR] : 1
##
##           Kappa : 0.1977
##
##  McNemar's Test P-Value : NA
##
## Statistics by Class:
##
##           Class: bad Class: good Class: normal
## Sensitivity                NA      0.41379      0.8488
## Specificity                0.95937      0.89347      0.4138
## Pos Pred Value              NA      0.27907      0.9356
## Neg Pred Value              NA      0.93863      0.2143
## Prevalence                  0.00000      0.09062      0.9094
## Detection Rate              0.00000      0.03750      0.7719
## Detection Prevalence        0.04063      0.13437      0.8250
## Balanced Accuracy           NA      0.65363      0.6313
```

```
## XGB00ST
# grid search
param_grid_xgb <- expand.grid(
  nrounds = c(50, 100, 200, 500),
  max_depth = c(2, 4, 6),
  eta = seq(0.1, 0.3, by = 0.1),
  gamma = c(1, 3, 5),
  min_child_weight = c(2, 5, 7),
  subsample = 0.5,
  colsample_bytree = 0.5
)

# train control
control_xgb <- trainControl(
  method = "cv",
  number = 5,
  verboseIter = TRUE,
  allowParallel = TRUE
)

# xgb tune
xgb.tune <- train(
  x = train[, -12],
  y = train[, 12],
  trControl = control_xgb,
  tuneGrid = param_grid_xgb,
  verbose = TRUE,
  method = "xgbTree"
)
```

```
## + Fold1: eta=0.1, max_depth=2, gamma=1, colsample_bytree=0.5, min_child_weight=2, subsample=0.5, nrounds=500
## [17:58:18] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead.
## [17:58:18] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead.
## [17:58:18] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead.
## - Fold1: eta=0.1, max_depth=2, gamma=1, colsample_bytree=0.5, min_child_weight=2, subsample=0.5, nrounds=500
## + Fold1: eta=0.1, max_depth=2, gamma=1, colsample_bytree=0.5, min_child_weight=5, subsample=0.5, nrounds=500
## [17:58:19] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead.
## [17:58:19] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead.
## [17:58:19] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead.
## - Fold1: eta=0.1, max_depth=2, gamma=1, colsample_bytree=0.5, min_child_weight=5, subsample=0.5, nrounds=500
## + Fold1: eta=0.1, max_depth=2, gamma=1, colsample_bytree=0.5, min_child_weight=7, subsample=0.5, nrounds=500
## [17:58:22] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead.
## [17:58:22] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead.
## [17:58:22] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead.
## - Fold1: eta=0.1, max_depth=2, gamma=1, colsample_bytree=0.5, min_child_weight=7, subsample=0.5, nrounds=500
## + Fold1: eta=0.1, max_depth=2, gamma=3, colsample_bytree=0.5, min_child_weight=2, subsample=0.5, nrounds=500
```

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

```
## [18:15:11] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead.
## [18:15:11] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead.
## [18:15:11] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead.
## - Fold5: eta=0.3, max_depth=6, gamma=1, colsample_bytree=0.5, min_child_weight=7, subsample=0.5, nrounds=500
## + Fold5: eta=0.3, max_depth=6, gamma=3, colsample_bytree=0.5, min_child_weight=2, subsample=0.5, nrounds=500
## [18:15:15] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead.
## [18:15:15] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead.
## [18:15:15] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead.
## - Fold5: eta=0.3, max_depth=6, gamma=3, colsample_bytree=0.5, min_child_weight=2, subsample=0.5, nrounds=500
## + Fold5: eta=0.3, max_depth=6, gamma=3, colsample_bytree=0.5, min_child_weight=5, subsample=0.5, nrounds=500
## [18:15:19] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead.
## [18:15:19] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead.
## [18:15:19] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead.
## - Fold5: eta=0.3, max_depth=6, gamma=3, colsample_bytree=0.5, min_child_weight=5, subsample=0.5, nrounds=500
## + Fold5: eta=0.3, max_depth=6, gamma=3, colsample_bytree=0.5, min_child_weight=7, subsample=0.5, nrounds=500
## [18:15:22] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead.
## [18:15:22] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead.
## [18:15:22] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead.
## - Fold5: eta=0.3, max_depth=6, gamma=3, colsample_bytree=0.5, min_child_weight=7, subsample=0.5, nrounds=500
## + Fold5: eta=0.3, max_depth=6, gamma=5, colsample_bytree=0.5, min_child_weight=2, subsample=0.5, nrounds=500
## [18:15:26] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead.
## [18:15:26] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead.
## [18:15:26] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead.
## - Fold5: eta=0.3, max_depth=6, gamma=5, colsample_bytree=0.5, min_child_weight=2, subsample=0.5, nrounds=500
## + Fold5: eta=0.3, max_depth=6, gamma=5, colsample_bytree=0.5, min_child_weight=5, subsample=0.5, nrounds=500
## [18:15:30] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead.
## [18:15:30] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead.
## [18:15:30] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead.
## - Fold5: eta=0.3, max_depth=6, gamma=5, colsample_bytree=0.5, min_child_weight=5, subsample=0.5, nrounds=500
## + Fold5: eta=0.3, max_depth=6, gamma=5, colsample_bytree=0.5, min_child_weight=7, subsample=0.5, nrounds=500
## [18:15:33] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead.
## [18:15:33] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead.
## [18:15:33] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range` instead.
## - Fold5: eta=0.3, max_depth=6, gamma=5, colsample_bytree=0.5, min_child_weight=7, subsample=0.5, nrounds=500
## Aggregating results
## Selecting tuning parameters
## Fitting nrounds = 500, max_depth = 4, eta = 0.3, gamma = 1, colsample_bytree = 0.5, min_child_weight = 2, subsample = 0.5 on full training set
```

```
# best tune
xgb.tune$bestTune
```

```
##      nrounds max_depth eta gamma colsample_bytree min_child_weight subsample
## 256      500      4 0.3    1          0.5              2          0.5
```

```
# writing out the best model
control.xgb <- trainControl(method = "none",
                           verboseIter = TRUE,
                           allowParallel = TRUE
)

# final grid with the best values
final.grid <- expand.grid(nrounds = xgb.tune$bestTune$nrounds,
                        max_depth = xgb.tune$bestTune$max_depth,
                        gamma = xgb.tune$bestTune$gamma,
                        colsample_bytree = xgb.tune$bestTune$colsample_bytree,
                        min_child_weight = xgb.tune$bestTune$min_child_weight,
                        subsample = xgb.tune$bestTune$subsample,
                        eta = xgb.tune$bestTune$eta)

# building the model with the best parameters
xgb.model <- train(
  x = train[, -12],
  y = train[, 12],
  trControl = control.xgb,
  tuneGrid = final.grid,
  method = "xgbTree",
  verbose = TRUE
)
```

```
## Fitting nrounds = 500, max_depth = 4, gamma = 1, colsample_bytree = 0.5, min_child_weight = 2, subsample = 0.5, eta = 0.3 on full training set
```

```
# predictions
xgb.preds <- predict(xgb.model, test)

# confusion matrix
confusionMatrix(xgb.preds, test$taste)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction bad good normal
##      bad      0      0      1
##      good      0     22      7
##      normal  13     21    256
##
## Overall Statistics
##
##           Accuracy : 0.8688
##           95% CI : (0.8268, 0.9037)
##      No Information Rate : 0.825
##      P-Value [Acc > NIR] : 0.02067
##
##           Kappa : 0.4532
##
##  McNemar's Test P-Value : NA
##
## Statistics by Class:
##
##           Class: bad Class: good Class: normal
## Sensitivity          0.000000      0.51163      0.9697
## Specificity          0.996743      0.97473      0.3929
## Pos Pred Value       0.000000      0.75862      0.8828
## Neg Pred Value       0.959248      0.92784      0.7333
## Prevalence           0.040625      0.13437      0.8250
## Detection Rate       0.000000      0.06875      0.8000
## Detection Prevalence 0.003125      0.09062      0.9062
## Balanced Accuracy     0.498371      0.74318      0.6813
```