Model 1 Ensemble Classiication Model (Bagging)

Noel C. Sieras

2022-12-16

Bagging **Bagging** is also known as *bootstrap aggregating* prediction models, is a general method for fitting multiple versions of a prediction model and then combining (or ensembling) them into an aggregated prediction and is designed to improve the stability and accuracy of regression and classification algorithms.
Packages used for these enseble classification model

```
# Helper packages
library(dplyr)       # for data wrangling
```

```
##
## 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(ggplot2)     # for awesome plotting
```

```
## Warning: package 'ggplot2' was built under R version 4.2.2
```

```
library(doParallel)  # for parallel backend to foreach
```

```
## Loading required package: foreach
```

```
## Loading required package: iterators
```

```
## Loading required package: parallel
```

```
library(foreach)     # for parallel processing with for loops
library(rsample)
```

```
## Warning: package 'rsample' was built under R version 4.2.2
```

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
```

```
## v tibble  3.1.8     v purrr   0.3.5
## v tidyr   1.2.1     v stringr 1.5.0
## v readr   2.1.3     v forcats 0.5.2
```

```
## Warning: package 'stringr' was built under R version 4.2.2
```

```
## -- Conflicts ---------------------------------------- tidyverse_conflicts() --
## x purrr::accumulate() masks foreach::accumulate()
## x dplyr::filter()     masks stats::filter()
## x dplyr::lag()        masks stats::lag()
## x purrr::when()       masks foreach::when()
```

```
# Modeling packages
library(caret)       # for general model fitting
```

```
## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(rpart)       # for fitting decision trees
library(ipred)       # for fitting bagged decision trees
library(ROCR)
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.
##
## Attaching package: 'pROC'
##
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

Use `set.seed()` for reproducibility

```
# for reproducibility
set.seed(12345)
```

Data splitting of normalized data into 80% training data and 20% testing data
The normalized data `datRD` is split into 80% for training data and 20% for testing data.

```
#80% training data - 20% testing data
library(readr)
library(rsample)
library(caret)
setwd("D:/FilesWorkOn&Saved/PhDStat/@MSUIIT/SY20222023/01FirstSemester/STT225_StatisticalComputing1/04F
datRD <- read_csv("normalRad.csv", show_col_types = FALSE)
rdsplit <- initial_split(datRD, prop = 0.8, strata = "Failure.binary")
rdsplit
```

```
## <Training/Testing/Total>
## <157/40/197>
```

```
rdtrain <- training(rdsplit)
rdtest  <- testing(rdsplit)
```

Training a Model

In `bagging()` function, we use `nbagg()` to control how many iterations to include in the bagged model and the `coob = TRUE` to indicate the use of the **Out Of Bag (oob)** error rate. The **oob** is used to estimate the prediction error. The size of the trees can be controlled by `control` arguments, it is an options that control details of the rpart algorithm. The chunks below uses `nbagg = 50`.

```
# train bagged model
bagging_1 <- bagging(
  formula = Failure.binary ~ .,
  data = rdtrain,
  nbagg = 100,
  coob = TRUE,
  control = rpart.control(minsplit = 2, cp = 0)
)

bagging_1
```

```
##
## Bagging regression trees with 100 bootstrap replications
##
## Call: bagging.data.frame(formula = Failure.binary ~ ., data = rdtrain,
##     nbagg = 100, coob = TRUE, control = rpart.control(minsplit = 2,
##         cp = 0))
##
## Out-of-bag estimate of root mean squared error:  0.2789
```

Based on the results, the oob of RMSE is 0.2777.
Bagging and Cross-validation (cv) We can also apply bagging within caret and use 10-fold CV to see how well our ensemble will generalize.

```
#train using caret
bagging_2 <- train(
  Failure.binary ~ .,
  data = rdtrain,
  method = "treebag",
  trControl = trainControl(method = "cv", number = 10),
  nbagg = 100,
  control = rpart.control(minsplit = 2, cp = 0)
)

bagging_2
```

```
## Bagged CART
##
## 157 samples
## 430 predictors
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 141, 142, 141, 141, 142, 141, ...
## Resampling results:
##
##   RMSE       Rsquared   MAE
##   0.2721693  0.6400335  0.1553417
```

The result shows that the RMSE value is 0.2784 which is almost similar to the OOB estimate with 0.2777.
Parellelizing the bagging algorithm The following chunks illustrates parallelizing the bagging algorithm (with b = 100 decision trees) on the radiomics data using eight clusters.

```
# Create a parallel socket cluster
cl <- makeCluster(8)

registerDoParallel(cl) # register the parallel backend

# Fit trees in parallel and compute predictions on the test set
predictions <- foreach(
  icount(100),
  .packages = "rpart",
  .combine = cbind
) %dopar% {
  # bootstrap copy of training data
  index <- sample(nrow(rdtrain), replace = TRUE)
  boot <- rdtrain[index, ]

# fit tree to bootstrap copy
  bagged_tree <- rpart(
    Failure.binary ~ .,
    control = rpart.control(minsplit = 2, cp = 0),
    data = boot
  )

  predict(bagged_tree, newdata = rdtest)
}

predictions[1:5, 1:7]
```

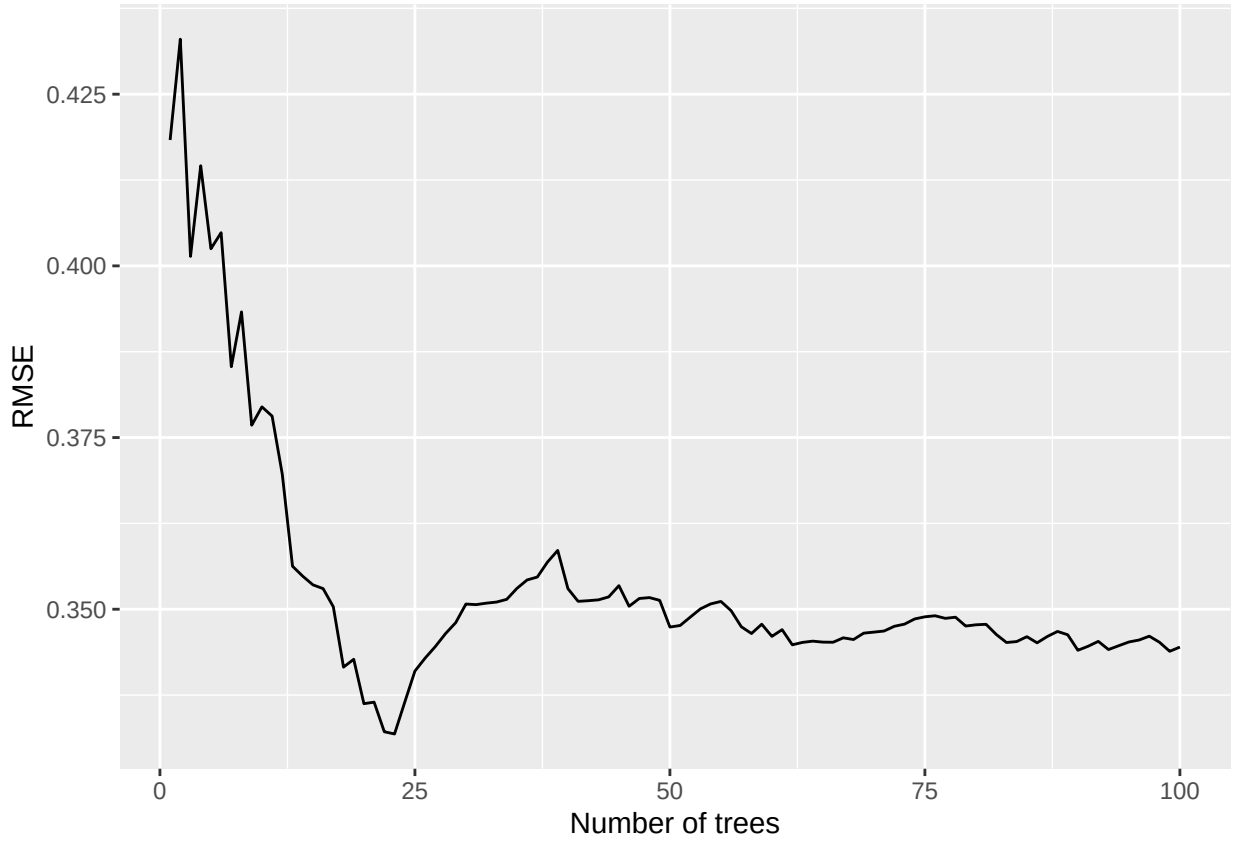
```
##   result.1 result.2 result.3 result.4 result.5 result.6 result.7
## 1        1        1        1        1        1        0        1
## 2        0        0        0        0        0        0        0
## 3        0        1        0        1        0        1        0
## 4        1        1        1        1        1        1        1
## 5        0        0        0        0        0        0        0
```

Predictions

To return the prediction for the test data for each of the trees.

```
predictions %>%
  as.data.frame() %>%
  mutate(
    observation = 1:n(),
    actual = rdtest$Failure.binary) %>%
  tidyr::gather(tree, predicted, -c(observation, actual)) %>%
  group_by(observation) %>%
  mutate(tree = stringr::str_extract(tree, '\\d+') %>% as.numeric()) %>%
  ungroup() %>%
  arrange(observation, tree) %>%
  group_by(observation) %>%
  mutate(avg_prediction = cummean(predicted)) %>%
  group_by(tree) %>%
  summarize(RMSE = RMSE(avg_prediction, actual)) %>%
  ggplot(aes(tree, RMSE)) +
  geom_line() +
```

```
  xlab('Number of trees')
```



Shutdown the parallel processing

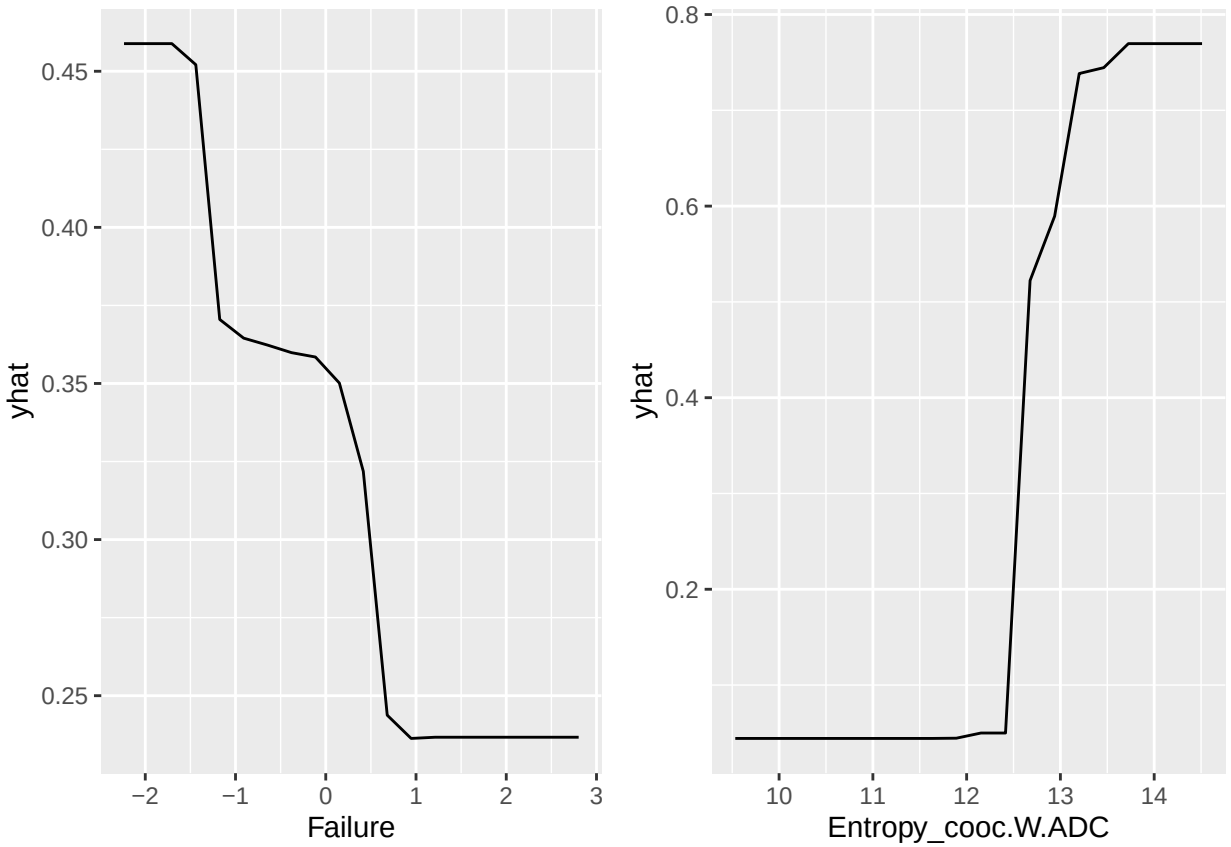
```
# Shutdown parallel cluster
stopCluster(cl)
```

Partial Dependence Plots PDPs or partial dependence plots tell us visually how each feature influences the predicted output, on average. PDPs help us to interpret any "black box" model.

```
# Construct partial dependence plots
p1 <- pdp::partial(
  bagging_2,
  pred.var = names(datRD)[3],
  grid.resolution = 20
) %>%
  autoplot()
```

```
p2 <- pdp::partial(
  bagging_2,
  pred.var = names(datRD)[4],
  grid.resolution = 20
) %>%
  autoplot()
```

```
gridExtra::grid.arrange(p1, p2, nrow = 1)
```



Prediction using training data To predict using training data of `bagging_2` model, we use the `predict()` function

```
# Use the predict function to predict using training data
pred_train <- predict(bagging_2, rdtrain)
summary(pred_train)
```

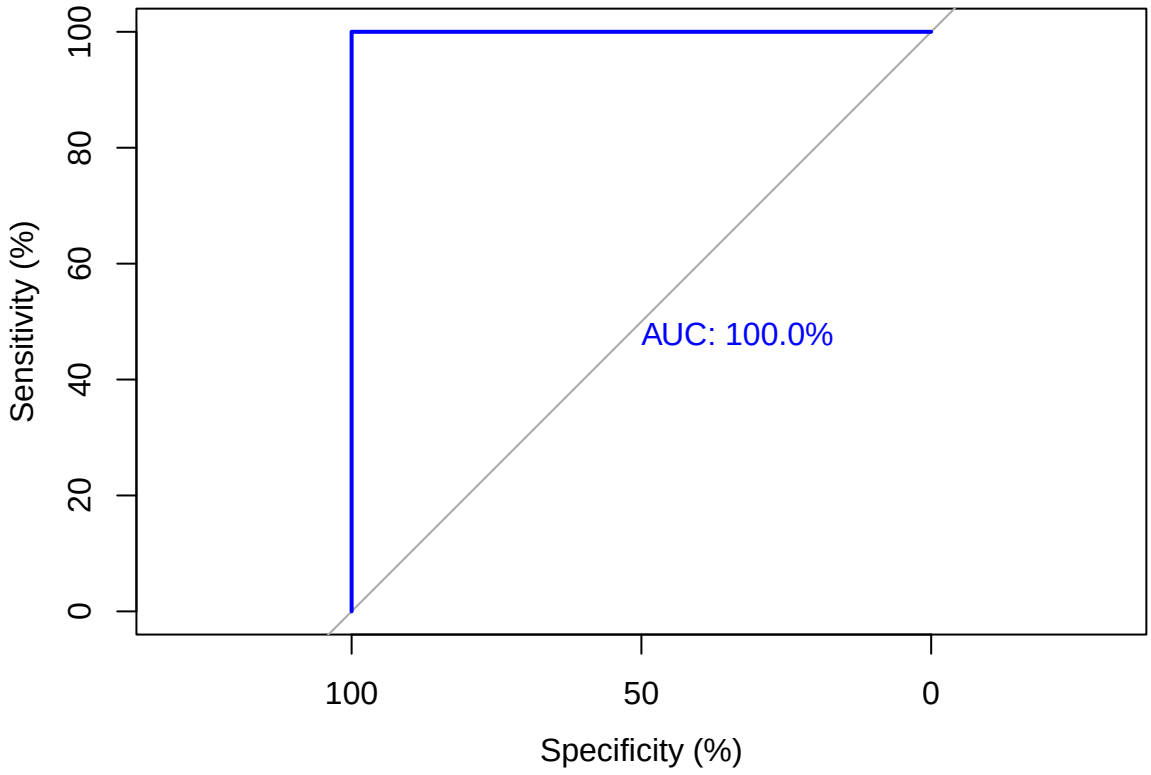
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.0000  0.0300  0.3353  0.8900  1.0000
```

Plotting using training data and printing of AUC values
To plot the training data and print the AUc values, we use the function `roc()`.

```
# Plot the training data performance while print the AUC values
roc(rdtrain$Failure.binary ~ pred_train, plot=TRUE, legacy.axes=FALSE,
    percent=TRUE, col="blue", lwd=2, print.auc=TRUE)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```




```
##
## Call:
## roc.formula(formula = rdtrain$Failure.binary ~ pred_train, plot = TRUE,     legacy.axes = FALSE, per
##
## Data: pred_train in 104 controls (rdtrain$Failure.binary 0) < 53 cases (rdtrain$Failure.binary 1).
## Area under the curve: 100%
```

Prediction using teting data To predict using testing data of 'bagging_2 model, we again use the predict() function.

```
# Use the predict function to predict using testing data
pred_test <- predict(bagging_2, rdtest)
summary(pred_test)
```

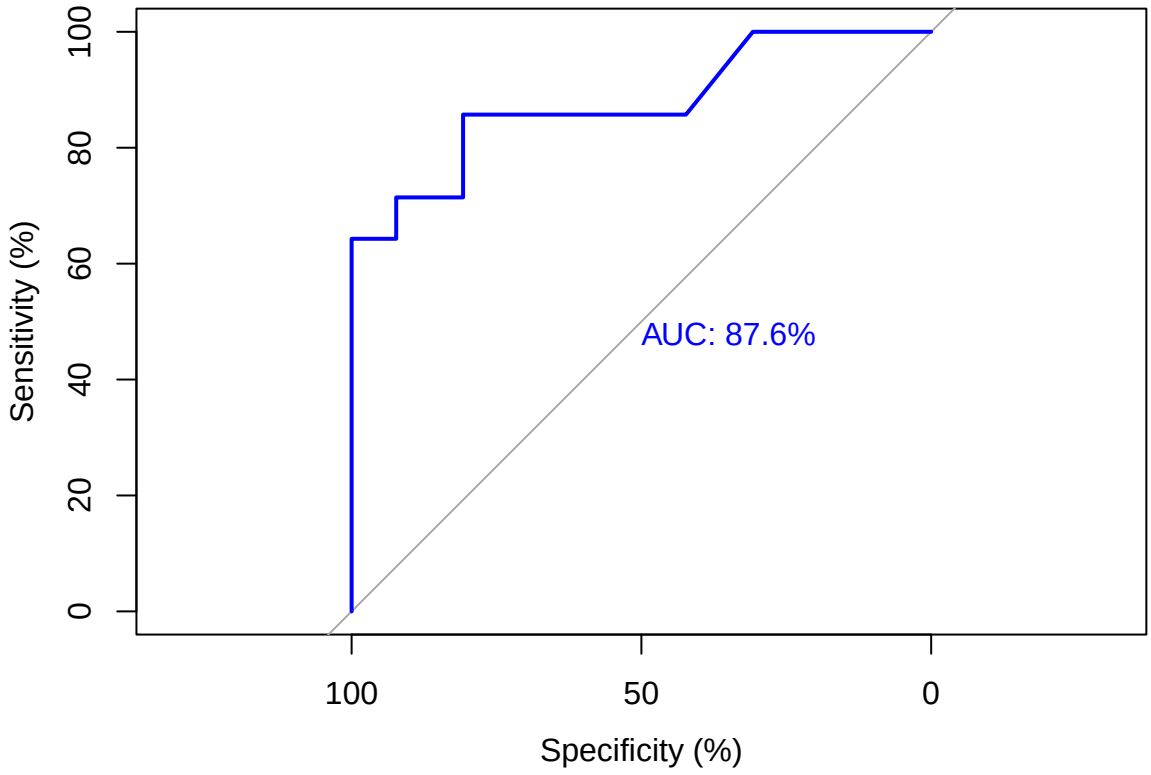
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.0200  0.0500  0.2915  0.6725  0.9900
```

Plotting using testing data and printing of AUC values To plot the testing data and print the AUC values, we use the function roc().

```
# Plot the testing data performance while print the AUC values
roc(rdtest$Failure.binary ~ pred_test, plot=TRUE, legacy.axes=FALSE,
    percent=TRUE, col="blue", lwd=2, print.auc=TRUE)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```



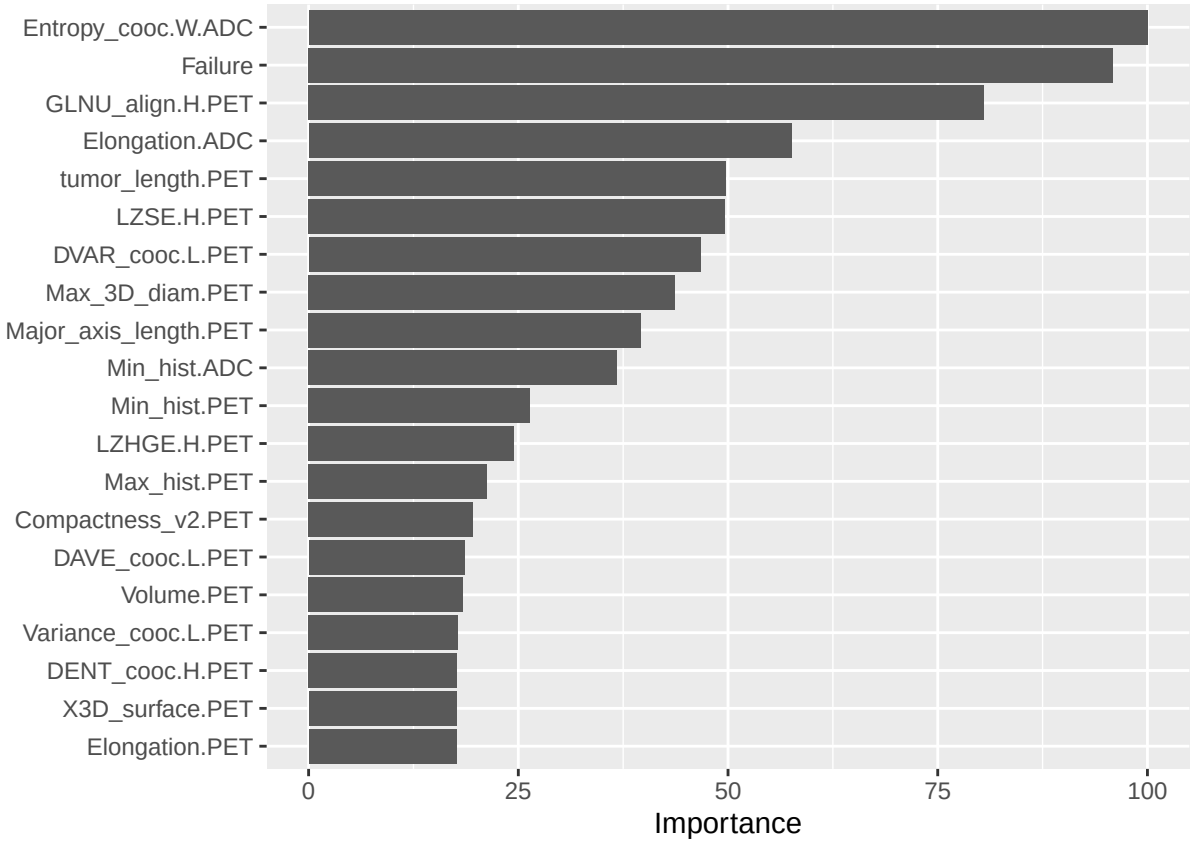

```
##
## Call:
## roc.formula(formula = rdtest$Failure.binary ~ pred_test, plot = TRUE,     legacy.axes = FALSE, percen
##
## Data: pred_test in 26 controls (rdtest$Failure.binary 0) < 14 cases (rdtest$Failure.binary 1).
## Area under the curve: 87.64%
```

Plotting of the Important Variable we use `vip()` to construct a variable importance plot (VIP) of the top 20 features in the `bagging_2` model.

```
library(vip)
```

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

```
vip(bagging_2, num_features = 20)
```


Entropy_cooc.W.ADC
Failure
GLNU_align.H.PET
Elongation.ADC
tumor_length.PET
LZSE.H.PET
DVAR_cooc.L.PET
Max_3D_diam.PET
Major_axis_length.PET
Min_hist.ADC
Min_hist.PET
LZHGE.H.PET
Max_hist.PET
Compactness_v2.PET
DAVE_cooc.L.PET
Volume.PET
Variance_cooc.L.PET
DENT_cooc.H.PET
X3D_surface.PET
Elongation.PET
0
25
50
75
100
Importance