Model 1: Bagging

Junaira I. Ibrahim

2022-12-15

Importing 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
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)
library(tidyverse)
```

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

```
## v tibble  3.1.8     v purrr   0.3.4
## v tidyr   1.2.1     v stringr 1.4.1
## v readr   2.1.3     v forcats 0.5.2
## -- 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()
```

```
library(bestNormalize)
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
```

```
library(vip)
```

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

Importing the dataset The dataset used in this model is imported from `radiomics data`. It has 197 observations and 431 variables.

```
datard <- read.csv("D:/MS_STATISTICS/STT225 Statistical Computing/FINAL PROJECT/radiomics_completedata.
dim(datard)
```

```
## [1] 197 431
```

Checking for null and missing values

```
is.na(datard)
colSums(is.na(datard)) # no NA thus, there is no missing values
```

Checking for normality

```
md1 = datard%>%select_if(is.numeric)
datamd1 = lapply(md1[,-1], shapiro.test)
r = lapply(datamd1, function(x)x$p.value) #Extracting p-value only
s=unlist(r)    #to convert a list to vector
sum(s[s>0.05])
```

```
## [1] 0.1350113
```

```
r$Entropy_cooc.W.ADC
```

```
## [1] 0.1350113
```

Based on the results, there is only one variable who is normally distributed (i.e. Entropy_cooc.W.ADC). All the rest are not normally distributed. Hence, we will try to normalize the data using `orderNorm()` function.

Normalizing the data

```
datard_norm = datard[,c(3,5:length(names(datard)))]
datard_norm = apply(datard_norm,2,orderNorm)
datard_norm = lapply(datard_norm, function(x) x$x.t)   #to transformed original data
datard_norm = datard_norm%>%as.data.frame()
```

Check the new data for normality

```
datalr2 = lapply(datard_norm, shapiro.test)
r2 = lapply(datalr2, function(x) x$p.value)
s2 = unlist(r2)
sum(s2>0.05)
```

```
## [1] 428
```

Based on the results, the rest of the variables is now normally distributed.

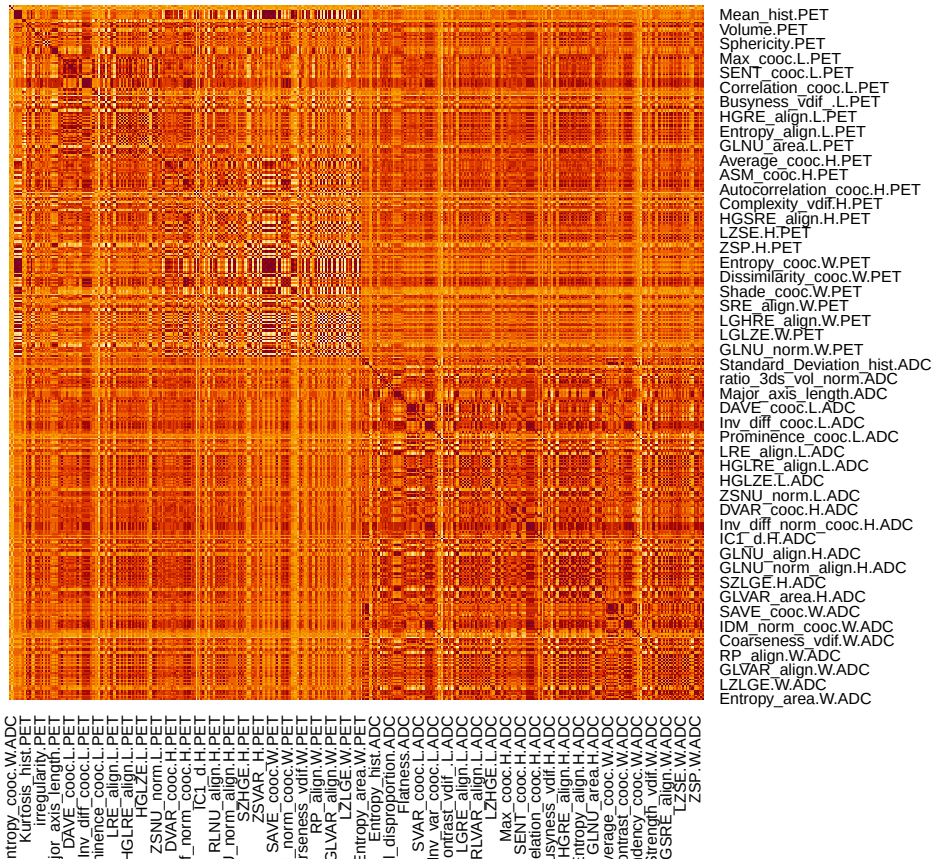
Substituting the normalized values into the original data, we have

```
r3 = select(datard, c("Failure.binary",  "Entropy_cooc.W.ADC"))
datard_m = cbind(r3,datard_norm)  #for bagging
```

In this session, we will use `datard_m`.

Getting the correlation of the whole data

```
#correlation
newdatard = select(datard_m, -c("Failure.binary"))
cor.newdatard = cor(newdatard)
corr = round(cor.newdatard,2) # 2 decimals
heatmap(corr,Rowv=NA,Colv=NA,scale="none",revC = T)
```



#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.

```r
# for reproducibility
set.seed(123)
```

The data `datard_m` is split into 80% of training data and 20% testing data.

```r
#80% training data - 20% testing data
rdsplit1 <- initial_split(datard_m, prop = 0.8, strata = "Failure.binary")
rdsplit1
```

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

```r
rdtrain1 <- training(rdsplit1)
rdtest1  <- testing(rdsplit1)
```

In `bagging()` function, we use `nbagg()` to control how many iterations to include in the bagged model and `coob = TRUE` to indicate to use 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 = 100`

```r
# train bagged model
bagging_1 <- bagging(
  formula = Failure.binary ~ .,
  data = rdtrain1,
  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 = rdtrain1,
##     nbagg = 100, coob = TRUE, control = rpart.control(minsplit = 2,
##         cp = 0))
##
## Out-of-bag estimate of root mean squared error:  0.2895
```

Based on the results, the oob of RMSE is 0.2895

We can also apply bagging within caret and use 10-fold CV to see how good our ensemble will generalize.

```r
#train using caret
bagging_2 <- train(
  Failure.binary ~ .,
  data = rdtrain1,
  method = "treebag",
  trControl = trainControl(method = "cv", number = 10),
  nbagg = 100,
  control = rpart.control(minsplit = 2, cp = 0)
)

bagging_2
```

```
## Bagged CART
##
```

```
## 157 samples
## 429 predictors
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 141, 141, 141, 141, 142, 142, ...
## Resampling results:
##
##   RMSE       Rsquared   MAE
##   0.2885362  0.5966712  0.1752458
```

The result shows that the RMSE value is 0.2885 which is almost similar to the OOB estimate with 0.2895

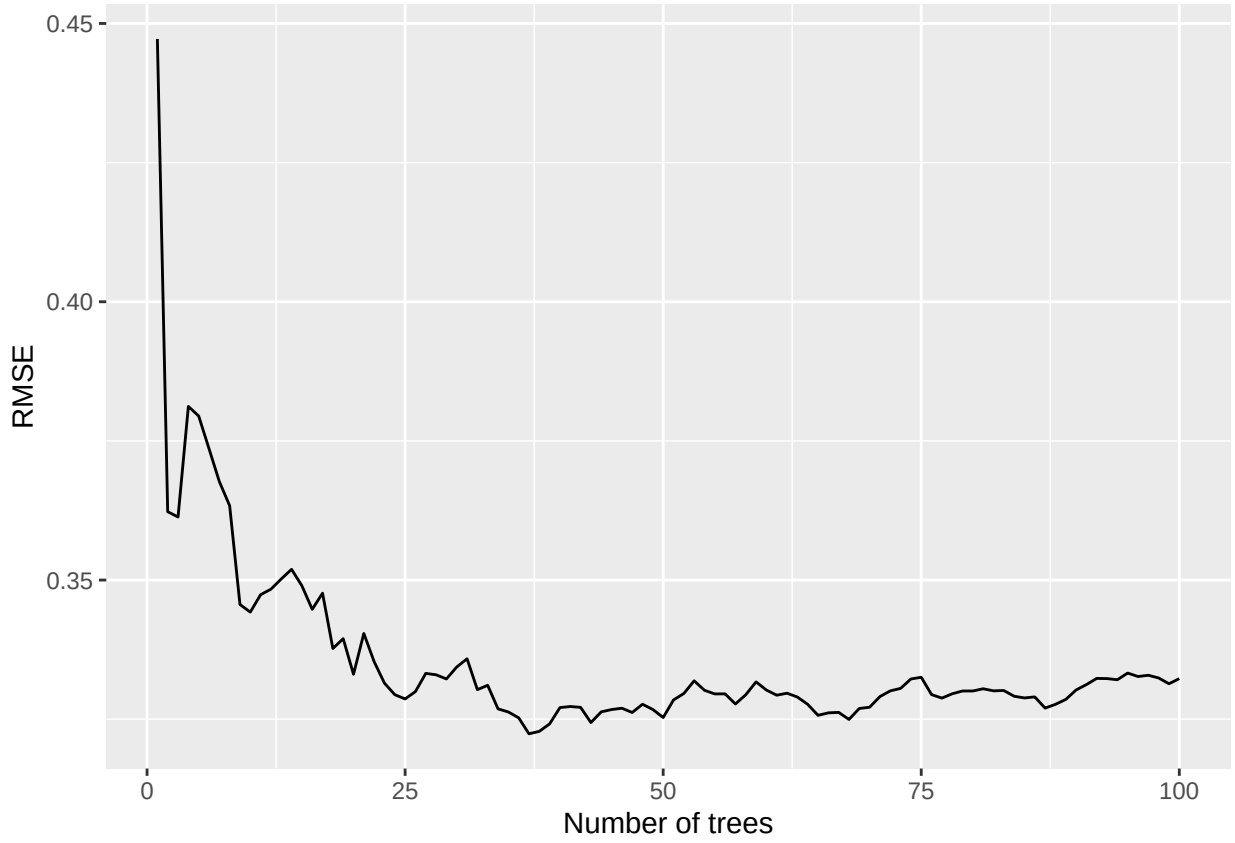
The following chunks illustrates parallelizing the bagging algorithm (with b = 100 decision trees) on the radiomics data using eight clusters.

```
# 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(rdtrain1), replace = TRUE)
  boot <- rdtrain1[index, ]

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

  predict(bagged_tree, newdata = rdtest1)
}

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

```
##    result.1 result.2 result.3 result.4 result.5 result.6 result.7
## 1         1        0        1        1        0        0        1
## 3         1        0        0        1        1        1        1
## 4         1        1        1        1        1        1        1
## 8         0        0        1        0        0        0        0
## 24        1        1        1        1        1        1        1
```

```
predictions %>%
  as.data.frame() %>%
  mutate(
    observation = 1:n(),
    actual = rdtest1$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')
```



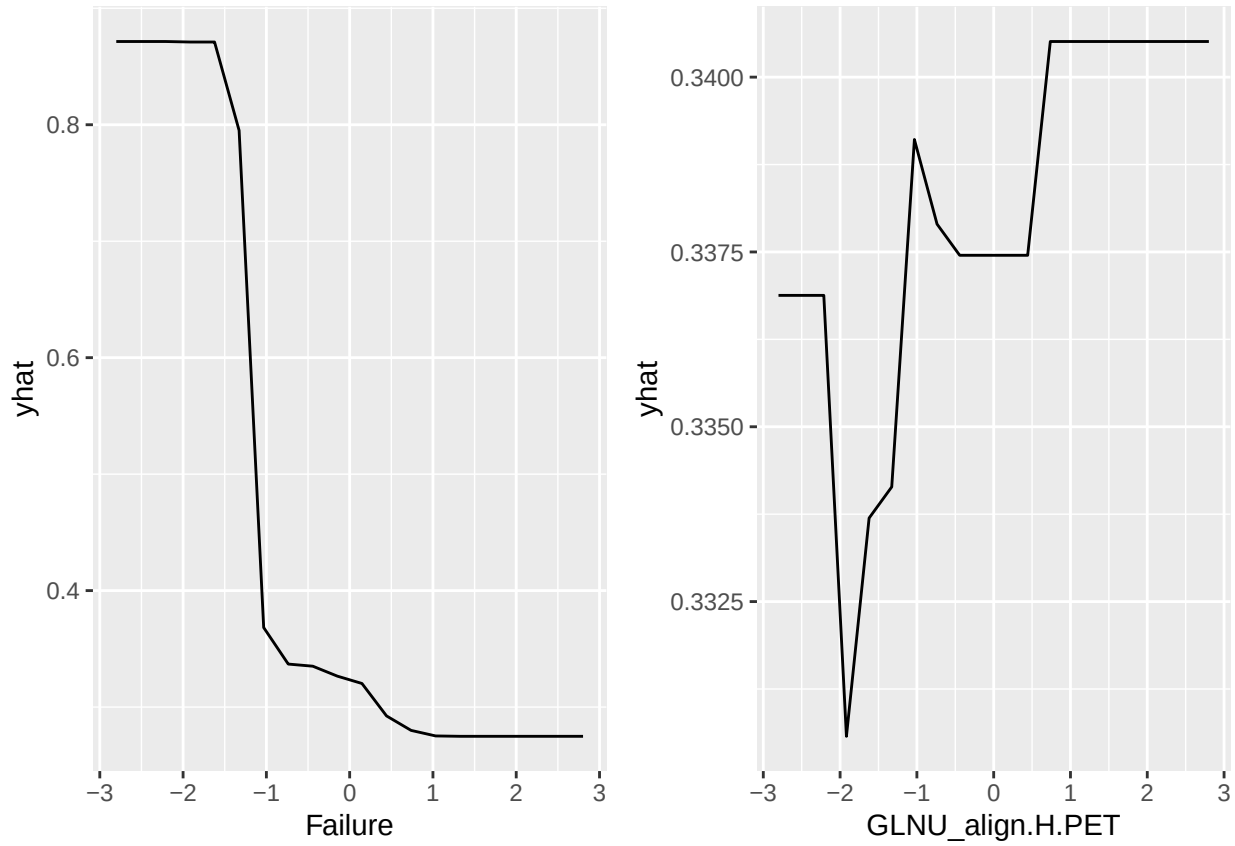

```
# Shutdown parallel cluster
stopCluster(cl)
```

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(datard_m)[3],
  grid.resolution = 20
) %>%
  autoplot()
```

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

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



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, rdtrain1)
summary(pred_train)
```

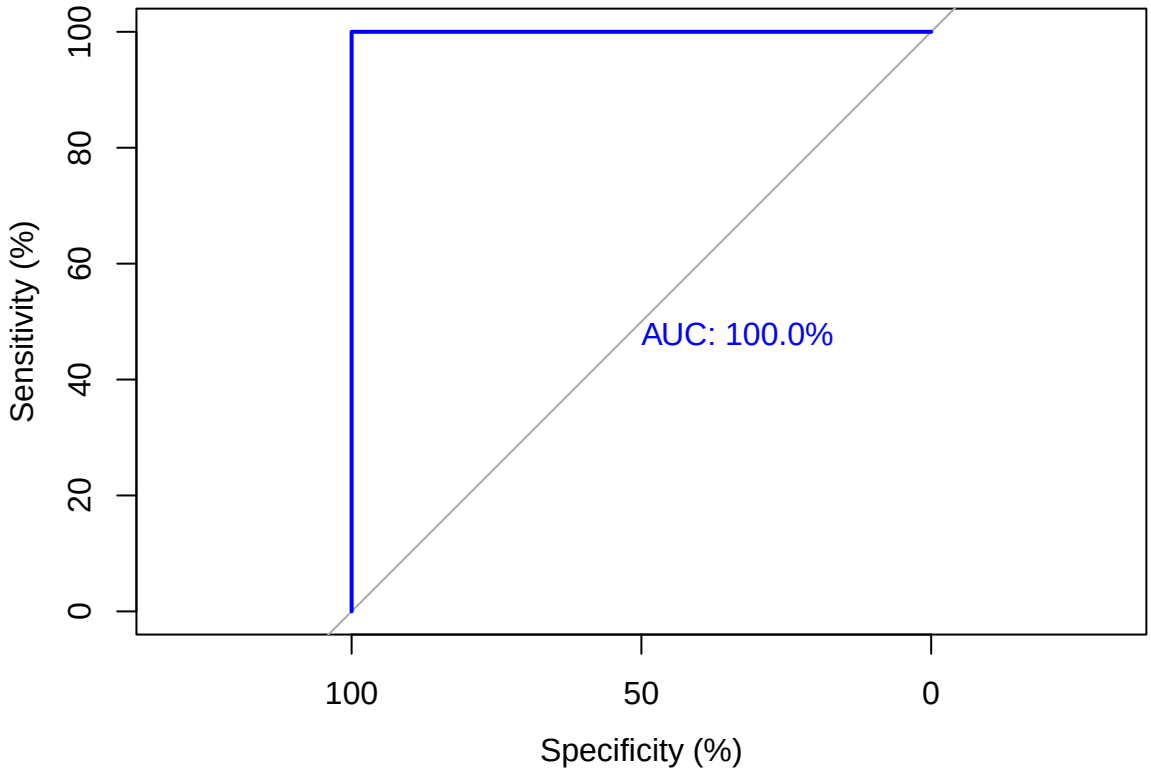
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.0000  0.0500  0.3389  0.8800  1.0000
```

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(rdtrain1$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 = rdtrain1$Failure.binary ~ pred_train, plot = TRUE,     legacy.axes = FALSE, pe
##
## Data: pred_train in 104 controls (rdtrain1$Failure.binary 0) < 53 cases (rdtrain1$Failure.binary 1).
## Area under the curve: 100%
```

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, rdtest1)
summary(pred_test)
```

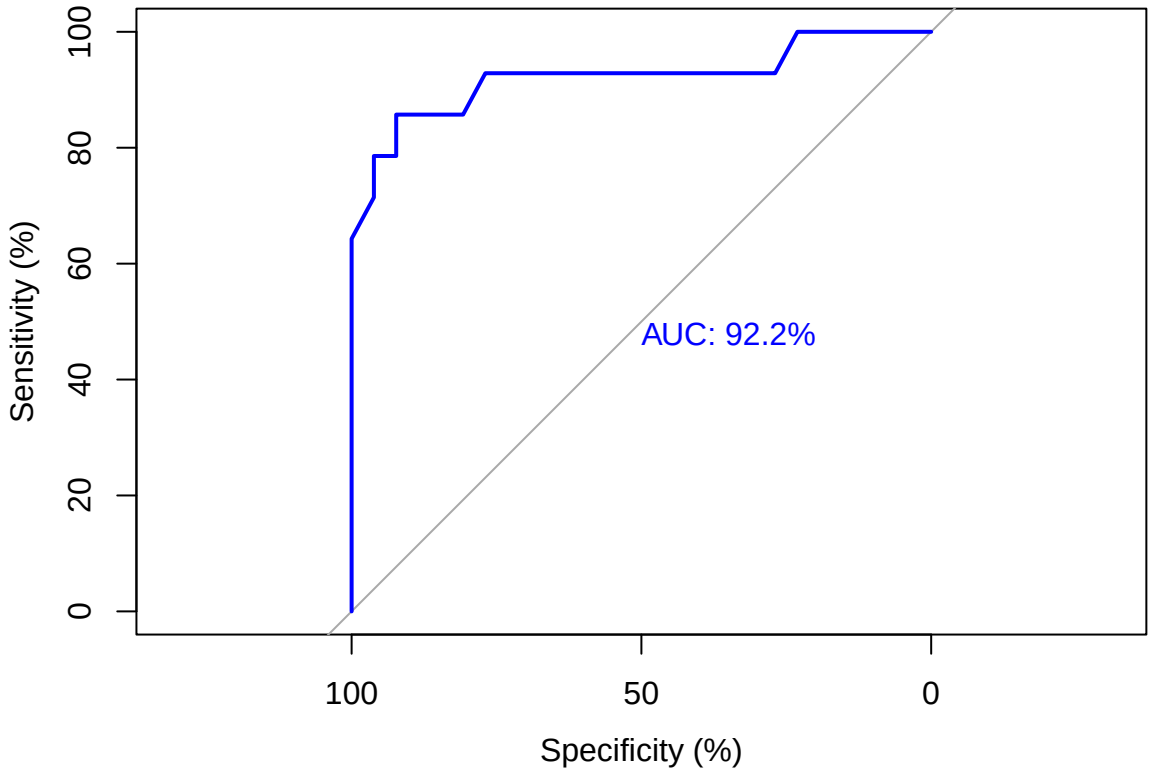
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.0750  0.2400  0.4088  0.8000  1.0000
```

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(rdtest1$Failure.binary ~ pred_test, plot=TRUE, legacy.axes=FALSE,
    percent=TRUE, col="blue", lwd=2, print.auc=TRUE)
```

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

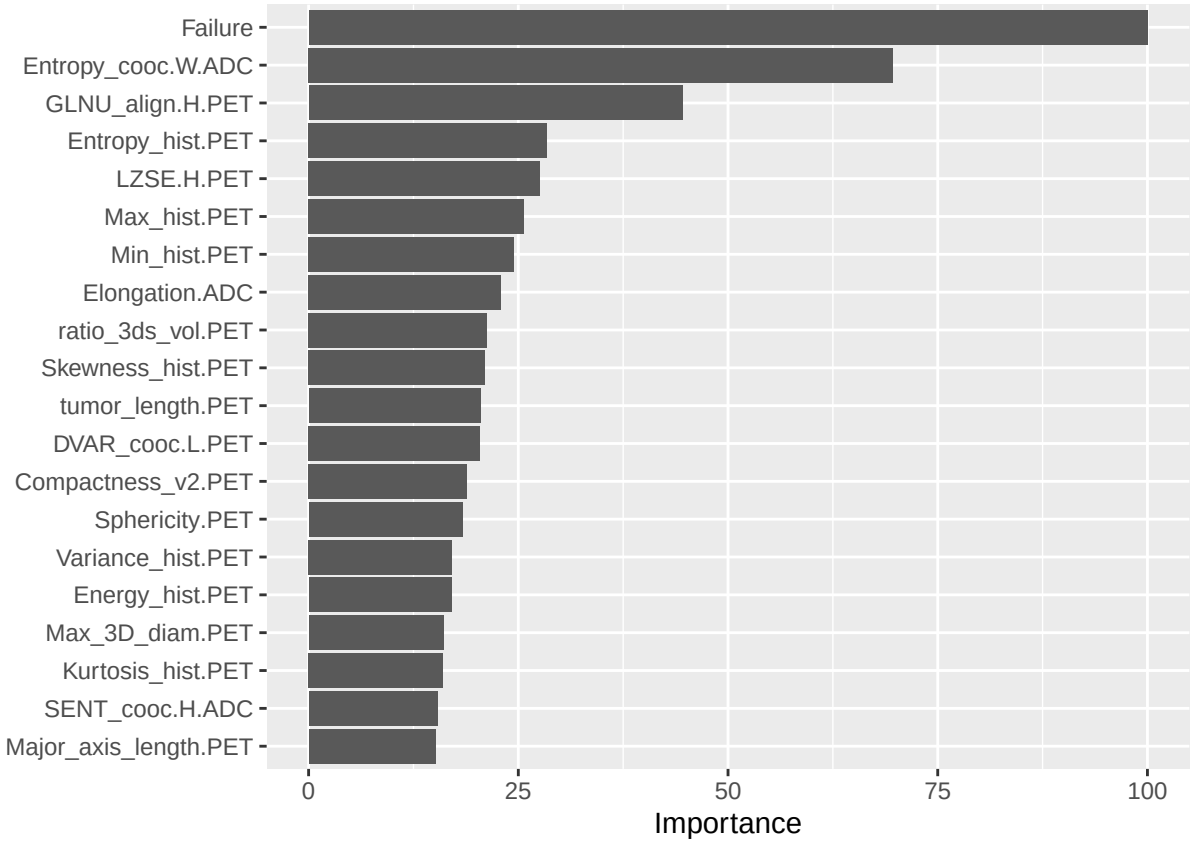
```
## Setting direction: controls < cases
```




```
##
## Call:
## roc.formula(formula = rdtest1$Failure.binary ~ pred_test, plot = TRUE,     legacy.axes = FALSE, perc
##
## Data: pred_test in 26 controls (rdtest1$Failure.binary 0) < 14 cases (rdtest1$Failure.binary 1).
## Area under the curve: 92.17%
```

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

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


Failure
Entropy_cooc.W.ADC
GLNU_align.H.PET
Entropy_hist.PET
LZSE.H.PET
Max_hist.PET
Min_hist.PET
Elongation.ADC
ratio_3ds_vol.PET
Skewness_hist.PET
tumor_length.PET
DVAR_cooc.L.PET
Compactness_v2.PET
Sphericity.PET
Variance_hist.PET
Energy_hist.PET
Max_3D_diam.PET
Kurtosis_hist.PET
SENT_cooc.H.ADC
Major_axis_length.PET
0
25
50
75
100
Importance