Classification for phone prices

XIAO MA, HAO ZHENG, YONGZI YU

05/09/2022

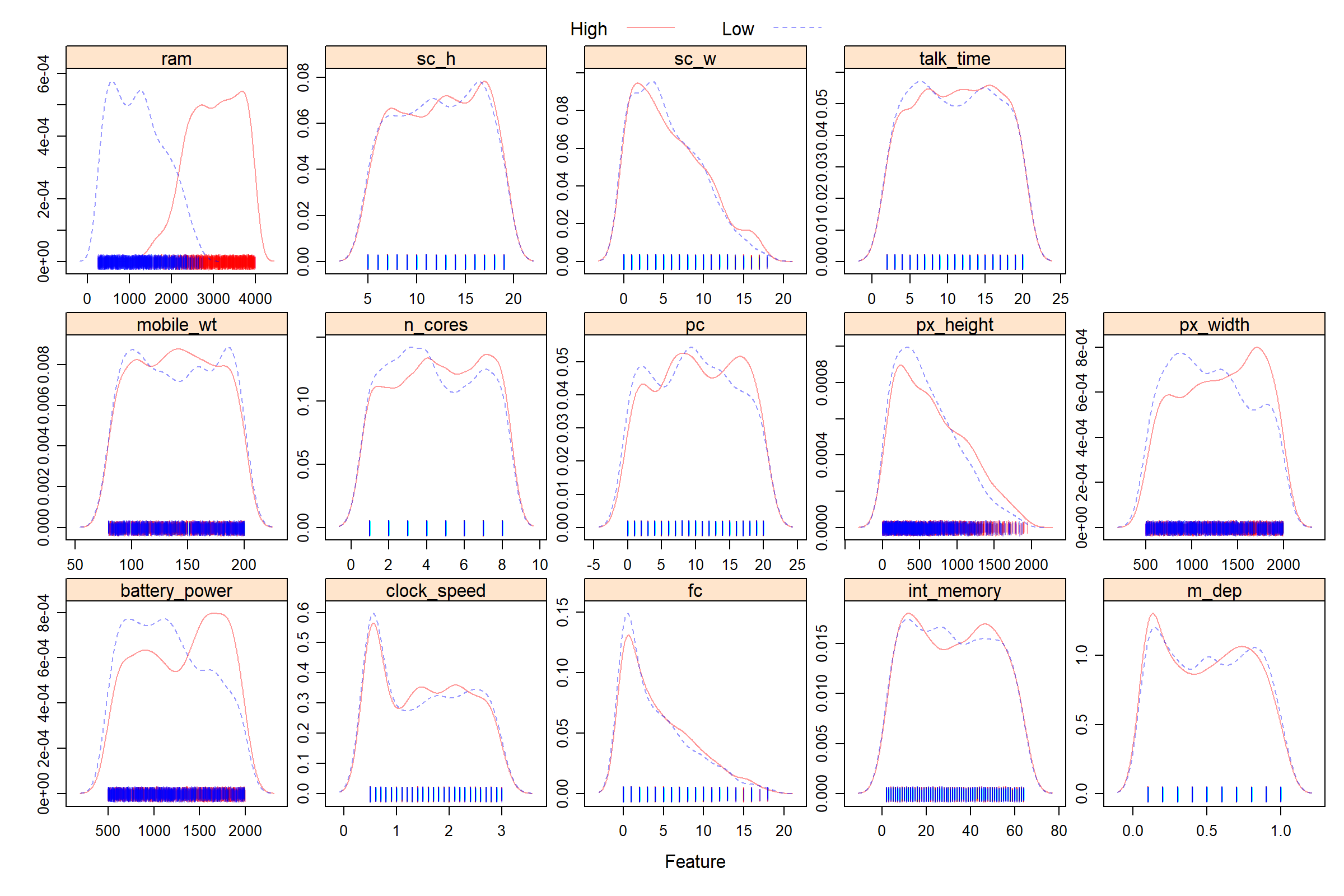
library(ggplot2)  
library(ISLR)  
library(dplyr)  
library(tidyverse)  
library(caret)  
library(glmnet)  
library(e1071)  
library(plotmo)  
library(splines)  
library(mgcv)  
library(pdp)  
library(earth)  
library(RNHANES)  
library(leaps)  
library(mlbench)  
library(pROC)  
library(vip)  
library(AppliedPredictiveModeling)  
library(gridExtra)  
library(mvtnorm)  
library(MASS)  
library(class)  
library(klaR)  
# library(reticulate)  
library(ggpubr)  
library(doParallel)  
library(ranger)  
library(gbm)  
library(pdp)  
library(lime)  
library(cutpointr)

# read data  
df = read.csv("data/train.csv")  
  
# covert outcome to binary  
df$price\_range = as.factor(ifelse(df$price\_range >= 2, "High", "Low"))  
  
# convert data format  
df = df %>%   
 mutate\_at(vars("blue", "dual\_sim", "four\_g", "three\_g", "touch\_screen", "wifi"),   
 ~factor(., levels = c(0, 1), labels = c("No", "Yes")))

# split into training set  
set.seed(1)  
train\_index = createDataPartition(df$price\_range,p=0.8,list = F)  
train\_df = df[train\_index,]  
test\_df = df[-train\_index,]  
  
ctrl <- trainControl(method = "cv", classProbs = TRUE,   
 summaryFunction = twoClassSummary)

# user parallel to accelarate   
cl <- makePSOCKcluster(4)  
registerDoParallel(cl)

#graphical summary: Feature plot  
x = train\_df[,-c(2,4,6,18:21)]  
y = train\_df$price\_range  
  
theme1 = transparentTheme(trans = .4)  
trellis.par.set(theme1)  
featurePlot(x,  
 y,  
 scales = list(x=list(relation="free"),   
 y=list(relation="free")),  
 plot = "density", pch = "|",   
 auto.key = list(columns = 2))



## model fitting

### logistic model

set.seed(1)  
#glm by caret  
glm.fit = train(x = train\_df[, c(1:20)],  
 y = train\_df$price\_range,  
 method = "glm",  
 metric = "ROC",  
 trControl = ctrl)  
summary(glm.fit)

##   
## Call:  
## NULL  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -2.71 0.00 0.00 0.00 1.50   
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) 671.8927 292.2955 2.30 0.022 \*  
## battery\_power -0.1226 0.0536 -2.29 0.022 \*  
## blueYes 4.5104 3.5237 1.28 0.201   
## clock\_speed 0.2048 1.2516 0.16 0.870   
## dual\_simYes 1.4306 1.8158 0.79 0.431   
## fc 0.9872 0.6089 1.62 0.105   
## four\_gYes 3.9353 2.7734 1.42 0.156   
## int\_memory -0.3622 0.1837 -1.97 0.049 \*  
## m\_dep 4.0292 3.6350 1.11 0.268   
## mobile\_wt 0.2175 0.1003 2.17 0.030 \*  
## n\_cores -1.0305 0.5090 -2.02 0.043 \*  
## pc -0.6660 0.3529 -1.89 0.059 .  
## px\_height -0.0617 0.0267 -2.31 0.021 \*  
## px\_width -0.0727 0.0320 -2.27 0.023 \*  
## ram -0.1870 0.0811 -2.31 0.021 \*  
## sc\_h -0.2549 0.2874 -0.89 0.375   
## sc\_w -0.6986 0.4201 -1.66 0.096 .  
## talk\_time -0.4483 0.2730 -1.64 0.101   
## three\_gYes -3.4908 2.4142 -1.45 0.148   
## touch\_screenYes 3.1611 2.3303 1.36 0.175   
## wifiYes 2.3363 2.1383 1.09 0.275   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 2218.071 on 1599 degrees of freedom  
## Residual deviance: 25.177 on 1579 degrees of freedom  
## AIC: 67.18  
##   
## Number of Fisher Scoring iterations: 17

#confusion matrix for test data  
test.pred.prob = predict(glm.fit, newdata = df[-train\_index,],  
 type = "prob")  
test.pred = factor(colnames(test.pred.prob)[max.col(test.pred.prob)])  
confusionMatrix(test.pred,  
 df$price\_range[-train\_index],  
 "High")

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction High Low  
## High 196 2  
## Low 4 198  
##   
## Accuracy : 0.985   
## 95% CI : (0.968, 0.994)  
## No Information Rate : 0.5   
## P-Value [Acc > NIR] : <2e-16   
##   
## Kappa : 0.97   
##   
## Mcnemar's Test P-Value : 0.683   
##   
## Sensitivity : 0.980   
## Specificity : 0.990   
## Pos Pred Value : 0.990   
## Neg Pred Value : 0.980   
## Prevalence : 0.500   
## Detection Rate : 0.490   
## Detection Prevalence : 0.495   
## Balanced Accuracy : 0.985   
##   
## 'Positive' Class : High   
##

# Accuracy of test data classification: 0.985```  
  
  
### Penalized logistic regression

set.seed(1)  
glmnGrid <- expand.grid(.alpha = seq(0, 1, length = 6),  
.lambda = exp(seq(-8, -2, length = 20)))  
set.seed(1)  
#-c(2,4,6,18:21)  
glmn.fit <- train(x = train\_df[, -c(2,4,6,18:22)],  
 y = train\_df$price\_range,  
 method = "glmnet",  
 tuneGrid = glmnGrid,  
 metric = "ROC",  
 trControl = ctrl)  
summary(glmn.fit)

## Length Class Mode   
## a0 100 -none- numeric   
## beta 1400 dgCMatrix S4   
## df 100 -none- numeric   
## dim 2 -none- numeric   
## lambda 100 -none- numeric   
## dev.ratio 100 -none- numeric   
## nulldev 1 -none- numeric   
## npasses 1 -none- numeric   
## jerr 1 -none- numeric   
## offset 1 -none- logical   
## classnames 2 -none- character  
## call 5 -none- call   
## nobs 1 -none- numeric   
## lambdaOpt 1 -none- numeric   
## xNames 14 -none- character  
## problemType 1 -none- character  
## tuneValue 2 data.frame list   
## obsLevels 2 -none- character  
## param 0 -none- list

#confusion matrix for the test data  
# test.pred.prob = predict(glmn.fit, newdata = df[-train\_index,],  
# type = "prob")  
# test.pred = factor(colnames(test.pred.prob)[max.col(test.pred.prob)])  
# confusionMatrix(test.pred,  
# df$price\_range[-train\_index],  
# "High")  
  
#confusion matrix  
glmn.pred <- predict(glmn.fit, newdata = df[-train\_index,], type = "prob")[,1]  
test.pred <- rep("Low", length(glmn.pred))  
test.pred[glmn.pred > 0.5] <- "High"  
confusionMatrix(data = as.factor(test.pred),  
 reference = df$price\_range[-train\_index],  
 positive = "High")

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction High Low  
## High 197 1  
## Low 3 199  
##   
## Accuracy : 0.99   
## 95% CI : (0.975, 0.997)  
## No Information Rate : 0.5   
## P-Value [Acc > NIR] : <2e-16   
##   
## Kappa : 0.98   
##   
## Mcnemar's Test P-Value : 0.617   
##   
## Sensitivity : 0.985   
## Specificity : 0.995   
## Pos Pred Value : 0.995   
## Neg Pred Value : 0.985   
## Prevalence : 0.500   
## Detection Rate : 0.492   
## Detection Prevalence : 0.495   
## Balanced Accuracy : 0.990   
##   
## 'Positive' Class : High   
##

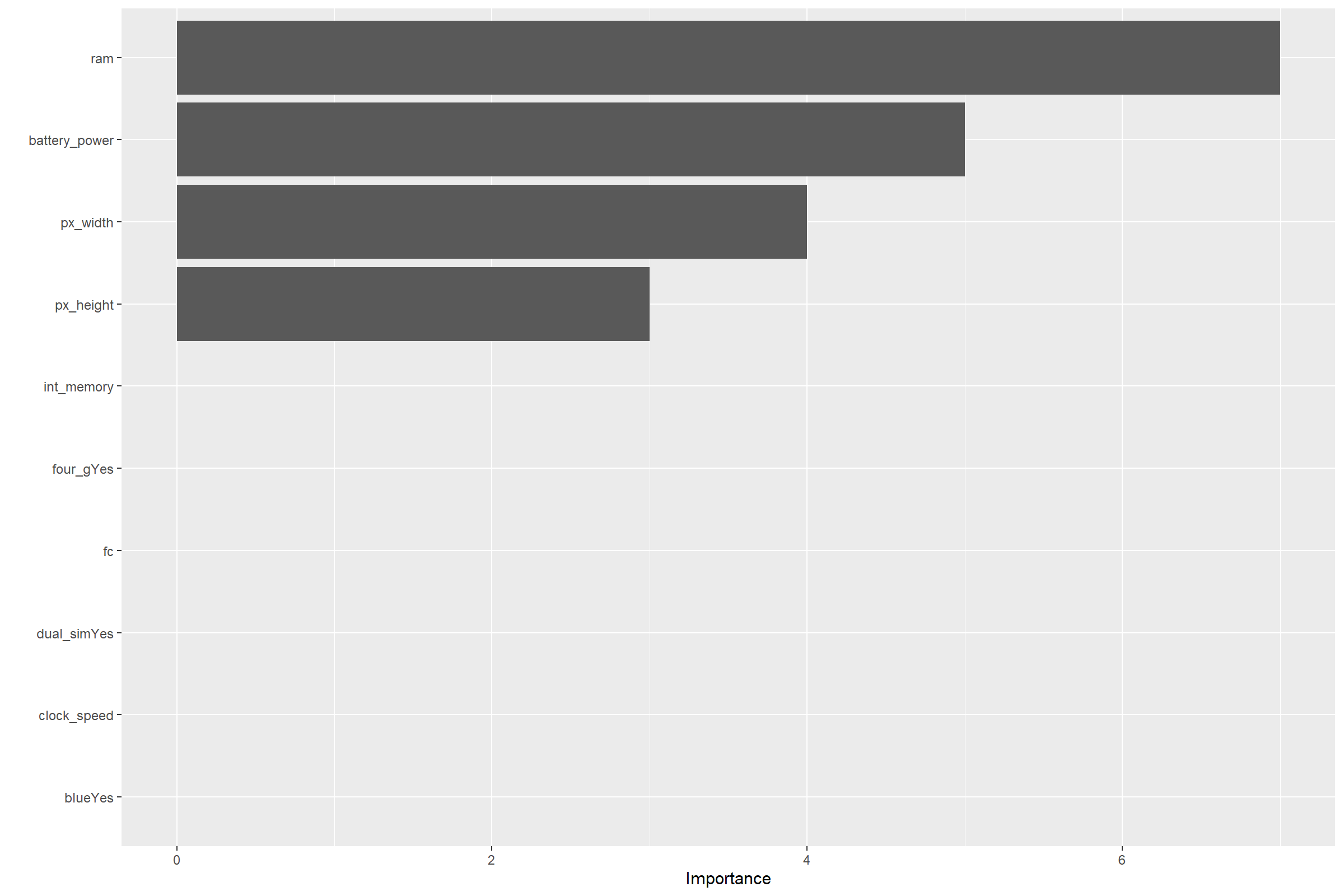
# Accuracy of test data classification: 0.99

### MARS

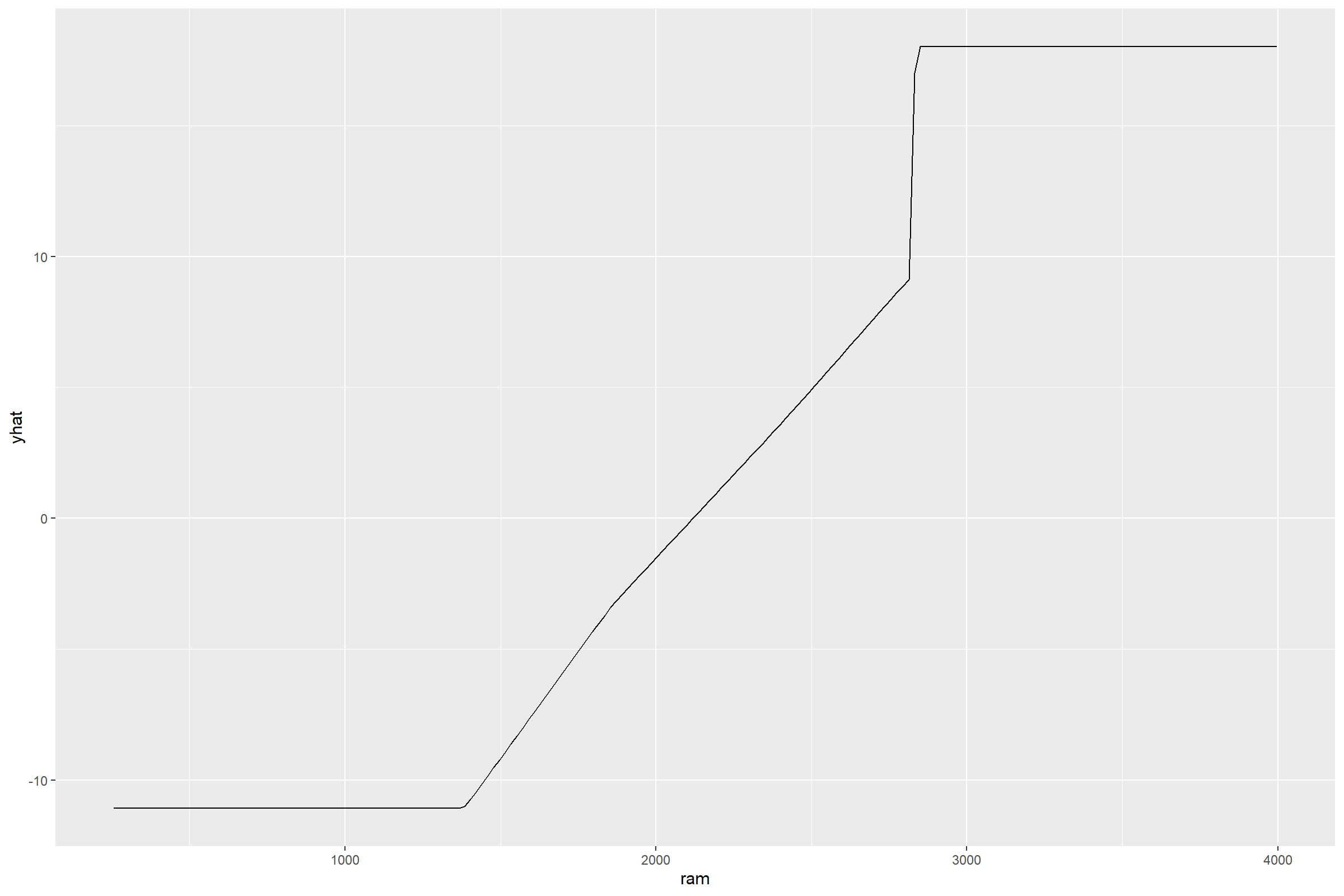
#[,-c(2,4,6,18:21)]  
set.seed(1)  
mars.fit = train(x = train\_df[,c(1:20)],  
 y = train\_df$price\_range,  
 method = "earth",  
 tuneGrid = expand.grid(degree = 1:3,nprune = 2:25),  
 metric = "ROC",   
 trControl = ctrl)  
coef(mars.fit$finalModel)

## (Intercept) h(ram-1380) h(1987-battery\_power)   
## -3.21160 -0.03186 0.01624   
## h(1703-px\_width) h(ram-2822) h(1578-px\_height)   
## 0.01105 -2.06084 0.00801   
## h(ram-1862) h(ram-2498)   
## 0.00636 -0.00115

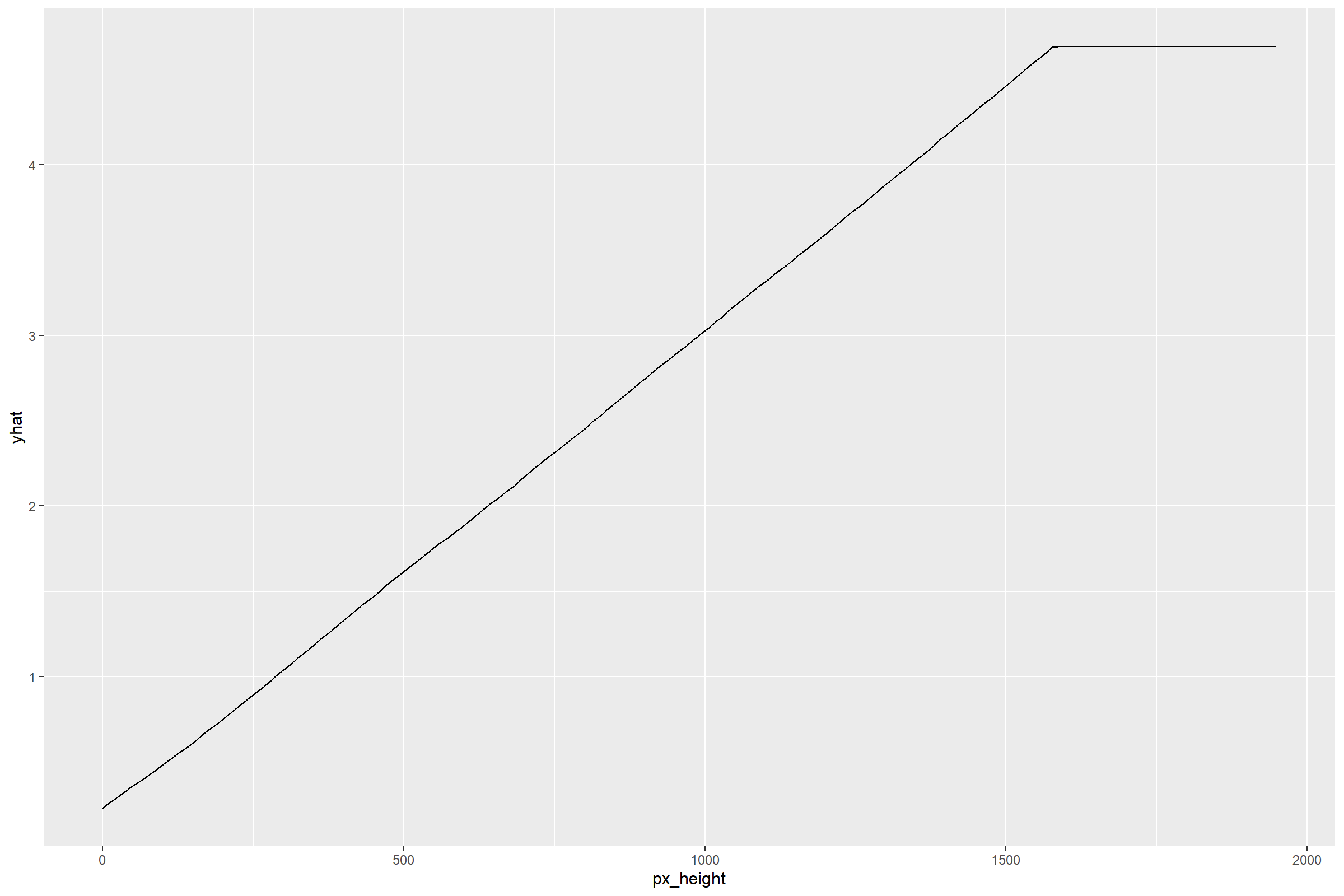
vip(mars.fit$finalModel)



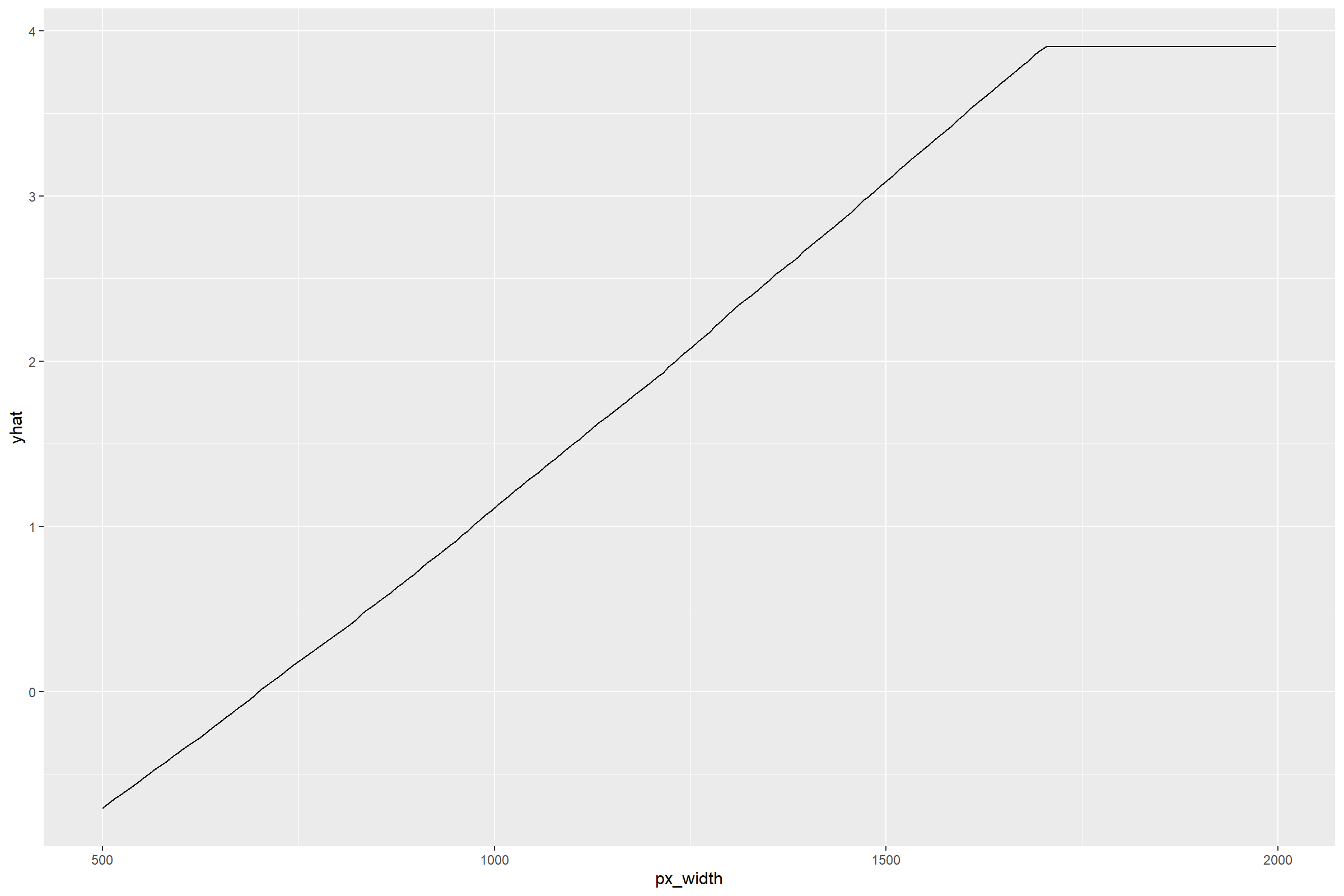
pdp::partial(mars.fit, pred.var = c("ram"), grid.resolution = 200) %>% autoplot()



pdp::partial(mars.fit, pred.var = c("px\_height"), grid.resolution = 200) %>% autoplot()



pdp::partial(mars.fit, pred.var = c("px\_width"), grid.resolution = 200) %>% autoplot()



#confusion matrix  
test.pred.prob = predict(mars.fit, newdata = df[-train\_index,],  
 type = "prob")  
test.pred = factor(colnames(test.pred.prob)[max.col(test.pred.prob)])  
confusionMatrix(test.pred,  
 df$price\_range[-train\_index],  
 "High")

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction High Low  
## High 193 1  
## Low 7 199  
##   
## Accuracy : 0.98   
## 95% CI : (0.961, 0.991)  
## No Information Rate : 0.5   
## P-Value [Acc > NIR] : <2e-16   
##   
## Kappa : 0.96   
##   
## Mcnemar's Test P-Value : 0.0771   
##   
## Sensitivity : 0.965   
## Specificity : 0.995   
## Pos Pred Value : 0.995   
## Neg Pred Value : 0.966   
## Prevalence : 0.500   
## Detection Rate : 0.482   
## Detection Prevalence : 0.485   
## Balanced Accuracy : 0.980   
##   
## 'Positive' Class : High   
##

# Accuracy of test data classification: 0.98

### GAM

set.seed(1)  
gam.fit = train(x = train\_df[, c(1:20)],  
 y = train\_df$price\_range,  
 method = "gam",   
 metric = "ROC",  
 trControl = ctrl)  
gam.fit$finalModel

##   
## Family: binomial   
## Link function: logit   
##   
## Formula:  
## .outcome ~ blue + dual\_sim + four\_g + three\_g + touch\_screen +   
## wifi + n\_cores + m\_dep + s(sc\_h) + s(fc) + s(sc\_w) + s(talk\_time) +   
## s(pc) + s(clock\_speed) + s(int\_memory) + s(mobile\_wt) + s(battery\_power) +   
## s(px\_height) + s(px\_width) + s(ram)  
##   
## Estimated degrees of freedom:  
## 1.000 0.000 0.000 2.422 1.000 0.000 0.785   
## 1.030 4.946 4.191 3.278 4.888 total = 32.5   
##   
## UBRE score: -0.959

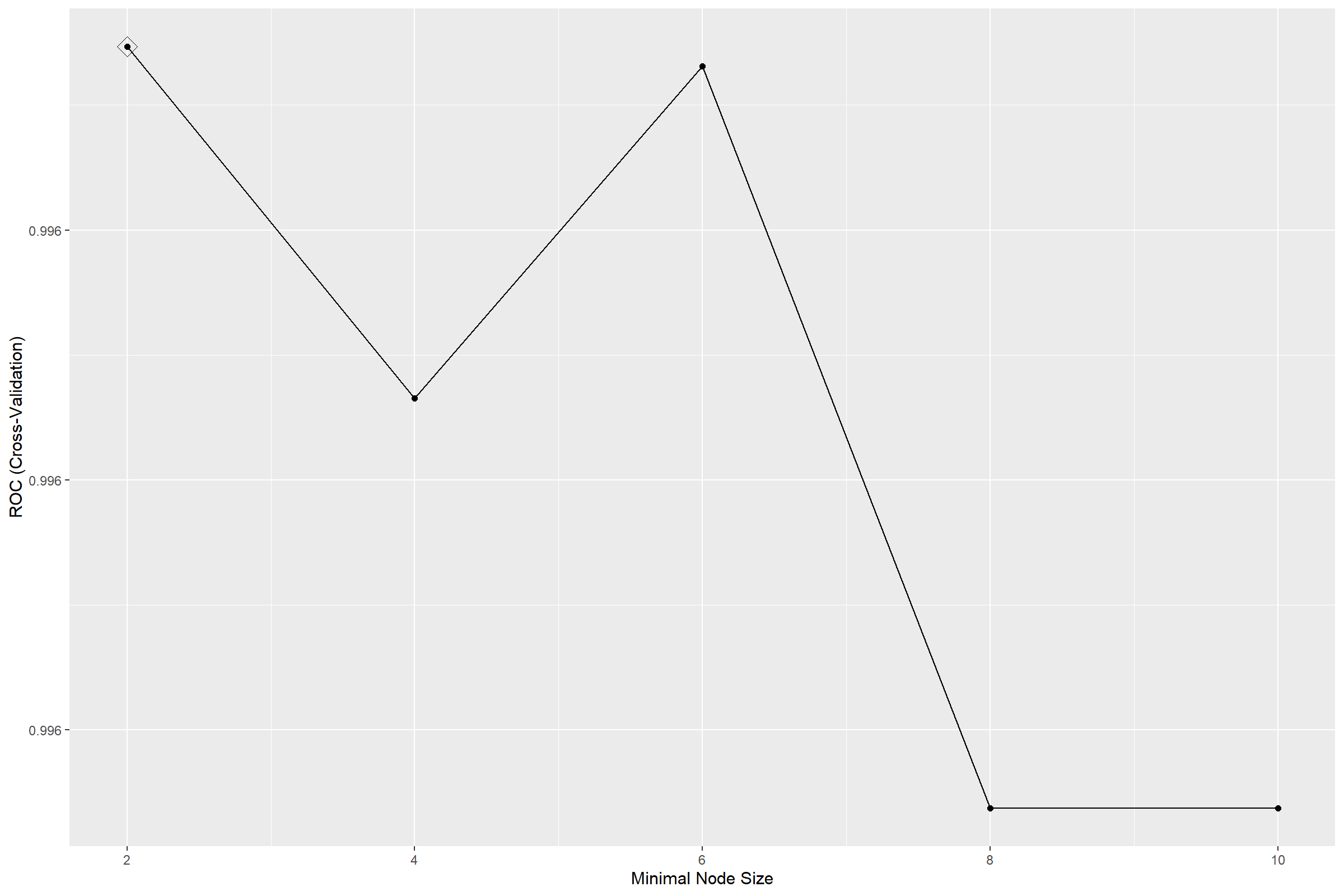
summary(gam.fit)

##   
## Family: binomial   
## Link function: logit   
##   
## Formula:  
## .outcome ~ blue + dual\_sim + four\_g + three\_g + touch\_screen +   
## wifi + n\_cores + m\_dep + s(sc\_h) + s(fc) + s(sc\_w) + s(talk\_time) +   
## s(pc) + s(clock\_speed) + s(int\_memory) + s(mobile\_wt) + s(battery\_power) +   
## s(px\_height) + s(px\_width) + s(ram)  
##   
## Parametric coefficients:  
## Estimate Std. Error z value Pr(>|z|)  
## (Intercept) -31.73 134696.44 0 1  
## blueYes 1.87 21541.45 0 1  
## dual\_simYes 26.92 43365.22 0 1  
## four\_gYes -1.60 19645.99 0 1  
## three\_gYes -18.32 29646.22 0 1  
## touch\_screenYes 16.59 31516.36 0 1  
## wifiYes 19.75 15247.42 0 1  
## n\_cores -7.44 5691.59 0 1  
## m\_dep 28.00 32095.22 0 1  
##   
## Approximate significance of smooth terms:  
## edf Ref.df Chi.sq p-value  
## s(sc\_h) 1.00e+00 9 0 1  
## s(fc) 5.66e-07 9 0 1  
## s(sc\_w) 5.84e-08 9 0 1  
## s(talk\_time) 2.42e+00 8 0 1  
## s(pc) 1.00e+00 9 0 1  
## s(clock\_speed) 2.07e-07 9 0 1  
## s(int\_memory) 7.85e-01 8 0 1  
## s(mobile\_wt) 1.03e+00 8 0 1  
## s(battery\_power) 4.95e+00 8 0 1  
## s(px\_height) 4.19e+00 8 0 1  
## s(px\_width) 3.28e+00 8 0 1  
## s(ram) 4.89e+00 8 0 1  
##   
## R-sq.(adj) = 1 Deviance explained = 100%  
## UBRE = -0.95933 Scale est. = 1 n = 1600

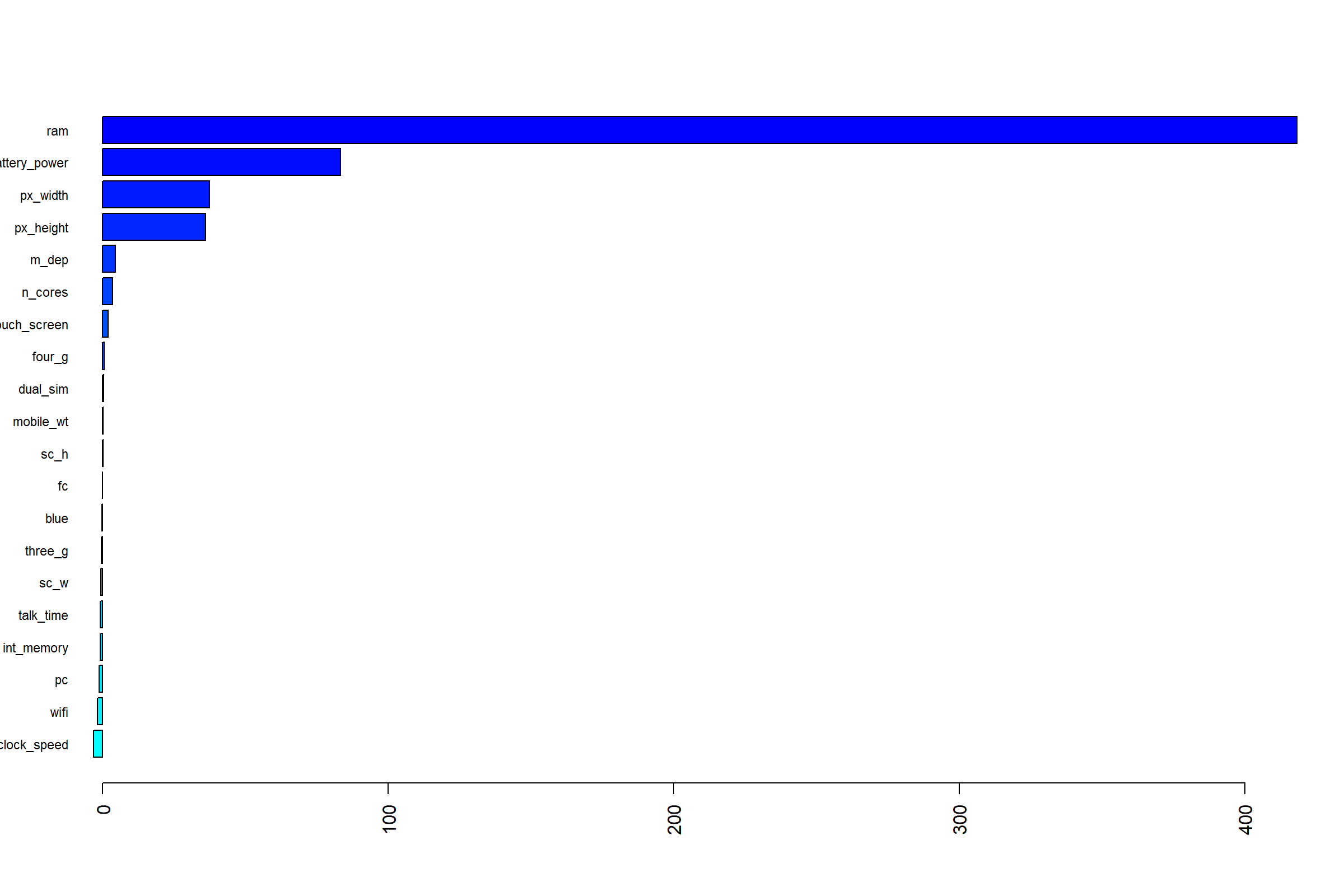
### Bagging

bagging.grid <- expand.grid(mtry = 20, # mtry = number of the predictors  
 splitrule = "gini",  
 min.node.size = seq(from = 2, to = 10, by = 2))  
set.seed(1)   
bagging.fit <- train(price\_range ~ . ,   
 df,   
 subset = train\_index,   
 tuneGrid = bagging.grid,   
 method = "ranger",  
 metric = "ROC",  
 trControl = ctrl)

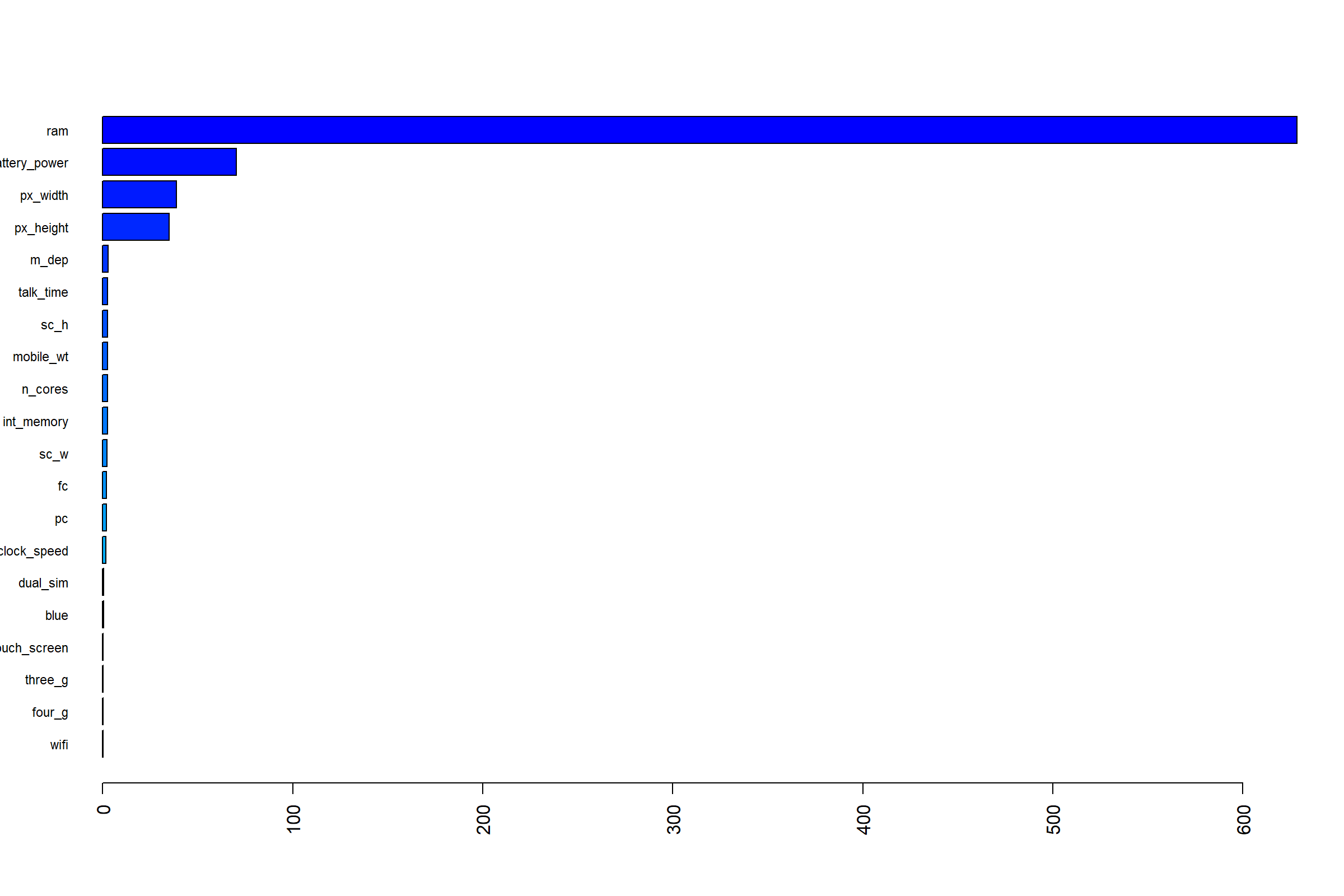
ggplot(bagging.fit, highlight = TRUE)



# Permutation based importance  
set.seed(1)  
bagging.final.per <- ranger(price\_range ~.,  
 df[train\_index, ],  
 mtry = bagging.fit$bestTune[[1]],  
 splitrule = "gini",  
 min.node.size = bagging.fit$bestTune[[3]],  
 importance = "permutation",  
 scale.permutation.importance = TRUE)  
barplot(sort(ranger::importance(bagging.final.per), decreasing = FALSE),  
 las = 2, horiz = TRUE, cex.names = 0.7,  
 col = colorRampPalette(colors = c("cyan", "blue"))(20))



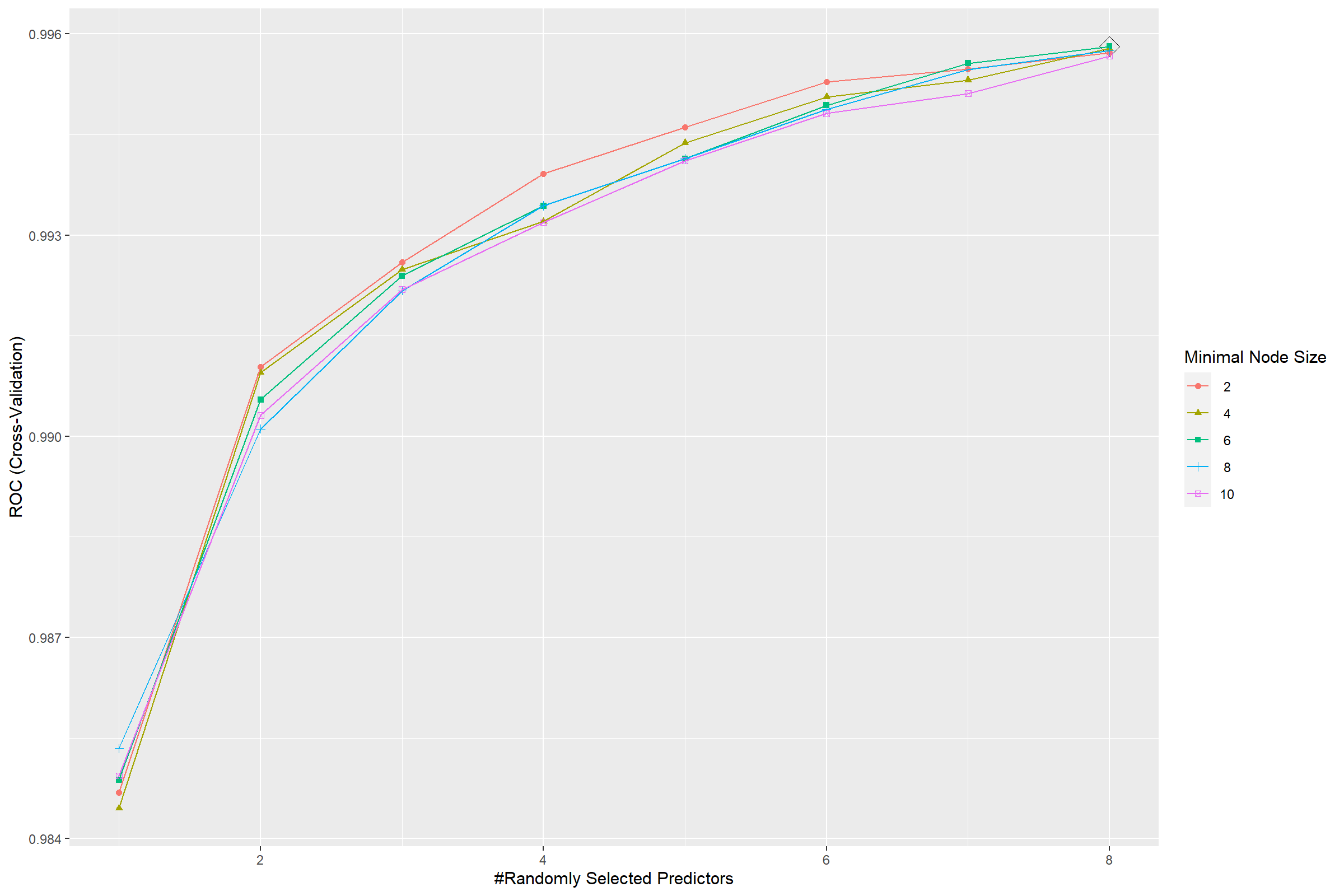
# Impurity based importance  
set.seed(1)  
bagging.final.imp <- ranger(price\_range~ .,  
 df[train\_index, ],  
 mtry = bagging.fit$bestTune[[1]],  
 splitrule = "gini",  
 min.node.size = bagging.fit$bestTune[[3]],  
 importance = "impurity")  
barplot(sort(ranger::importance(bagging.final.imp), decreasing = FALSE),  
 las = 2, horiz = TRUE, cex.names = 0.7,  
 col = colorRampPalette(colors = c("cyan", "blue"))(20))



### Random forest

ctrl <- trainControl(method = "cv", classProbs = TRUE,   
 summaryFunction = twoClassSummary)   
  
rf.grid <- expand.grid(mtry = 1:8,   
 splitrule = "gini",  
 min.node.size = seq(from = 2, to = 10, by = 2))  
set.seed(1)   
rf.fit <- train(price\_range ~ . ,   
 df,   
 subset = train\_index,   
 tuneGrid = rf.grid,   
 method = "ranger",  
 metric = "ROC",  
 trControl = ctrl)

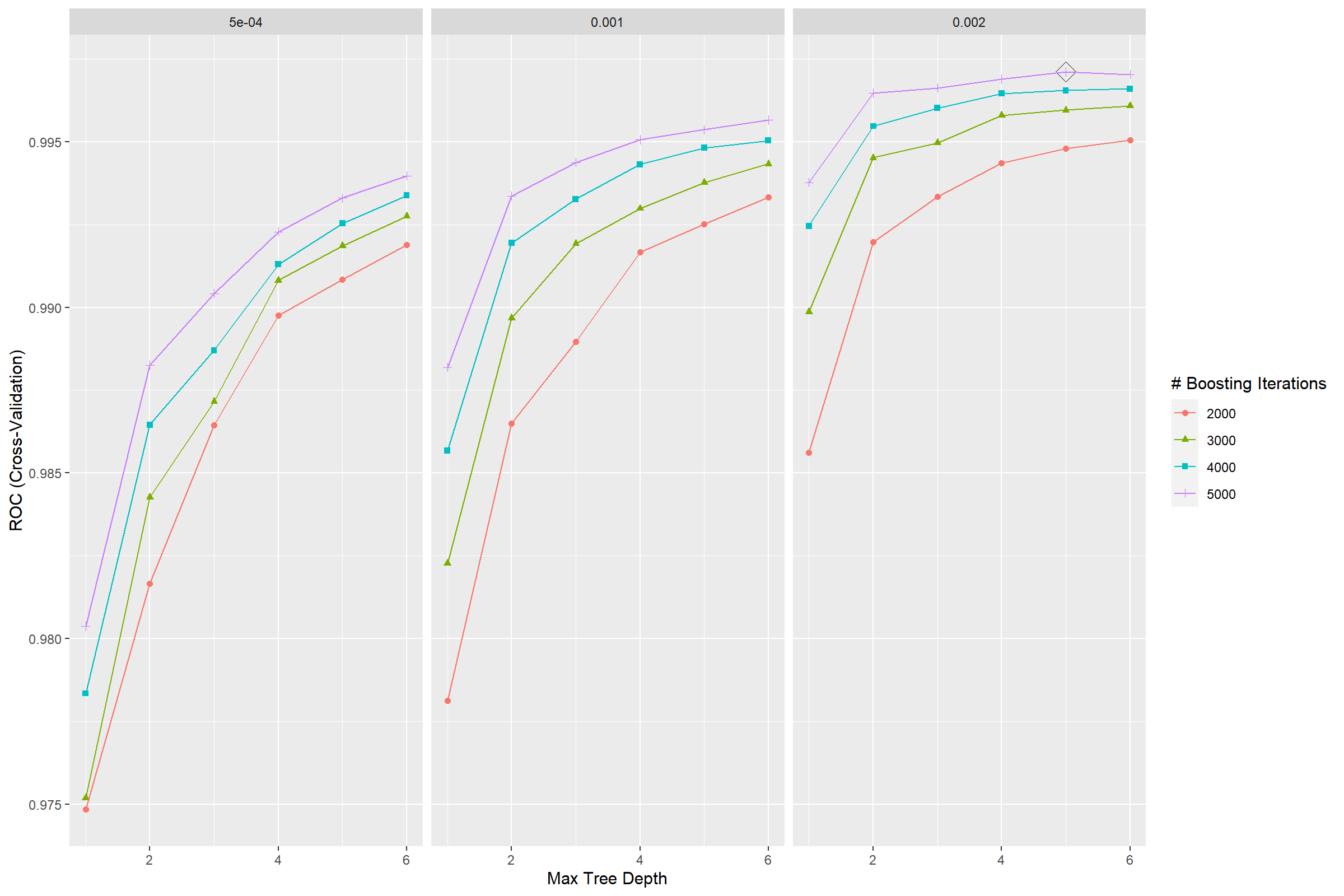
ggplot(rf.fit, highlight = TRUE)



### AdaBoost

gbm.grid <- expand.grid(n.trees = c(2000,3000,4000,5000),   
 interaction.depth = 1:6,   
 shrinkage = c(0.0005,0.001,0.002),   
 n.minobsinnode = 1)  
set.seed(1)   
gbm.fit <- train(price\_range ~ . ,   
 df,   
 subset = train\_index,   
 tuneGrid = gbm.grid,   
 trControl = ctrl,  
 method = "gbm",  
 distribution = "adaboost",  
 metric = "ROC",  
 verbose = FALSE)

ggplot(gbm.fit, highlight = TRUE)

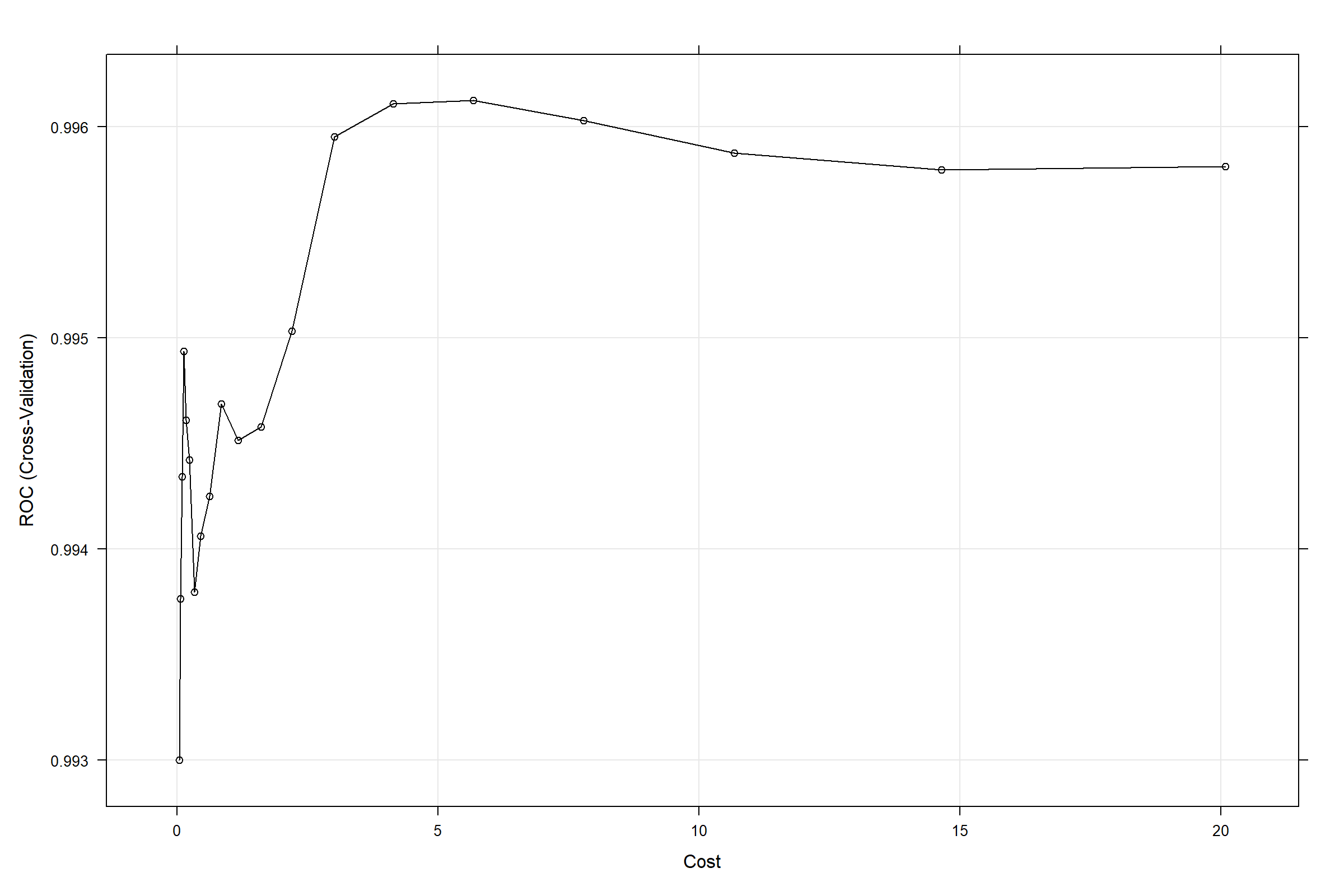


### SVM

# Tune over cost  
set.seed(1)  
svm.fit = train(price\_range ~ . ,   
 df,   
 subset = train\_index,   
 method = "svmRadialCost",  
 tuneGrid = data.frame(C = exp(seq(-3,3,len=20))),  
 trControl = ctrl,  
 metric = "ROC",  
 prob.model = TRUE,  
 verbose = FALSE)  
summary(svm.fit)

## Length Class Mode   
## 1 ksvm S4

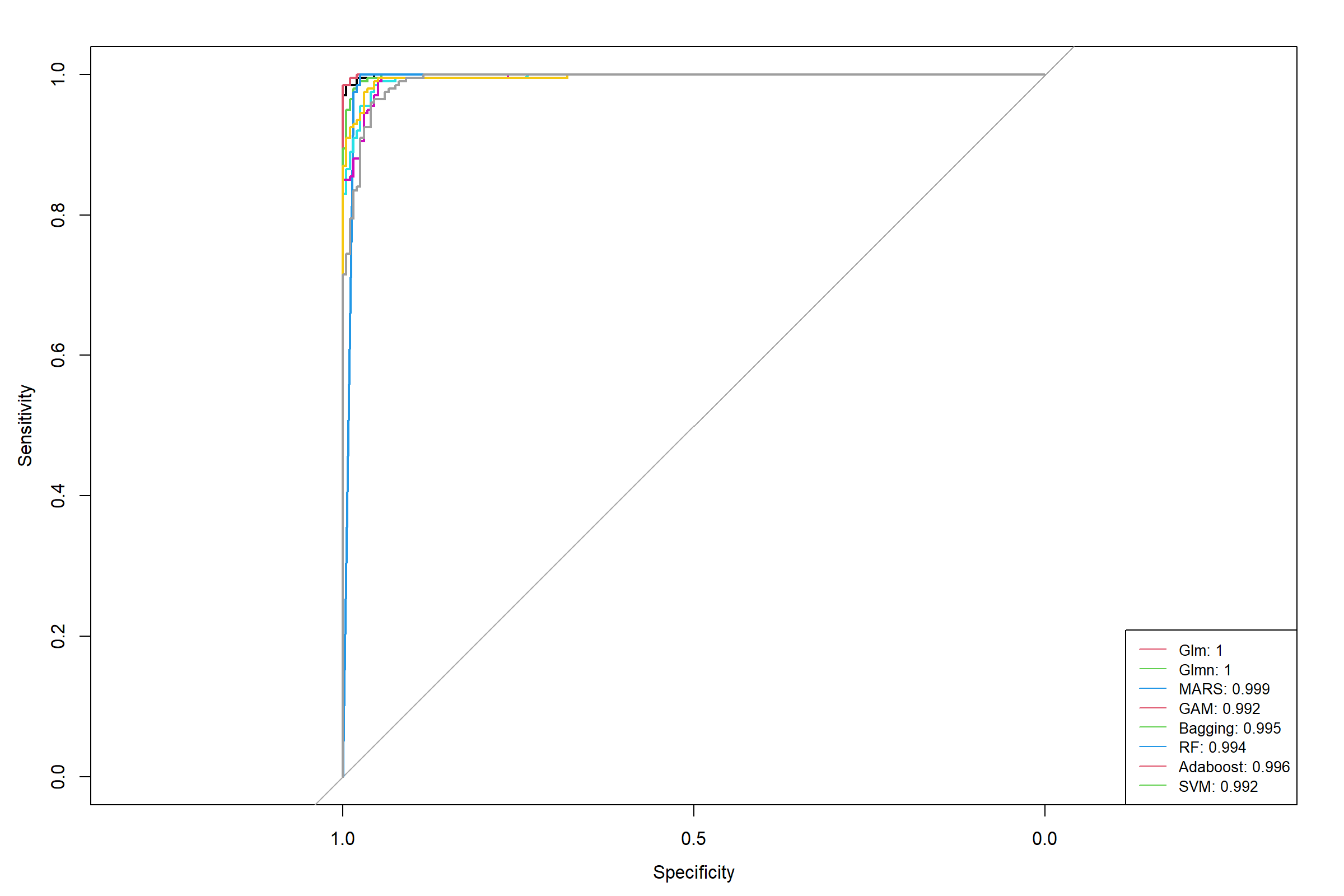
plot(svm.fit)



## ROC camparison

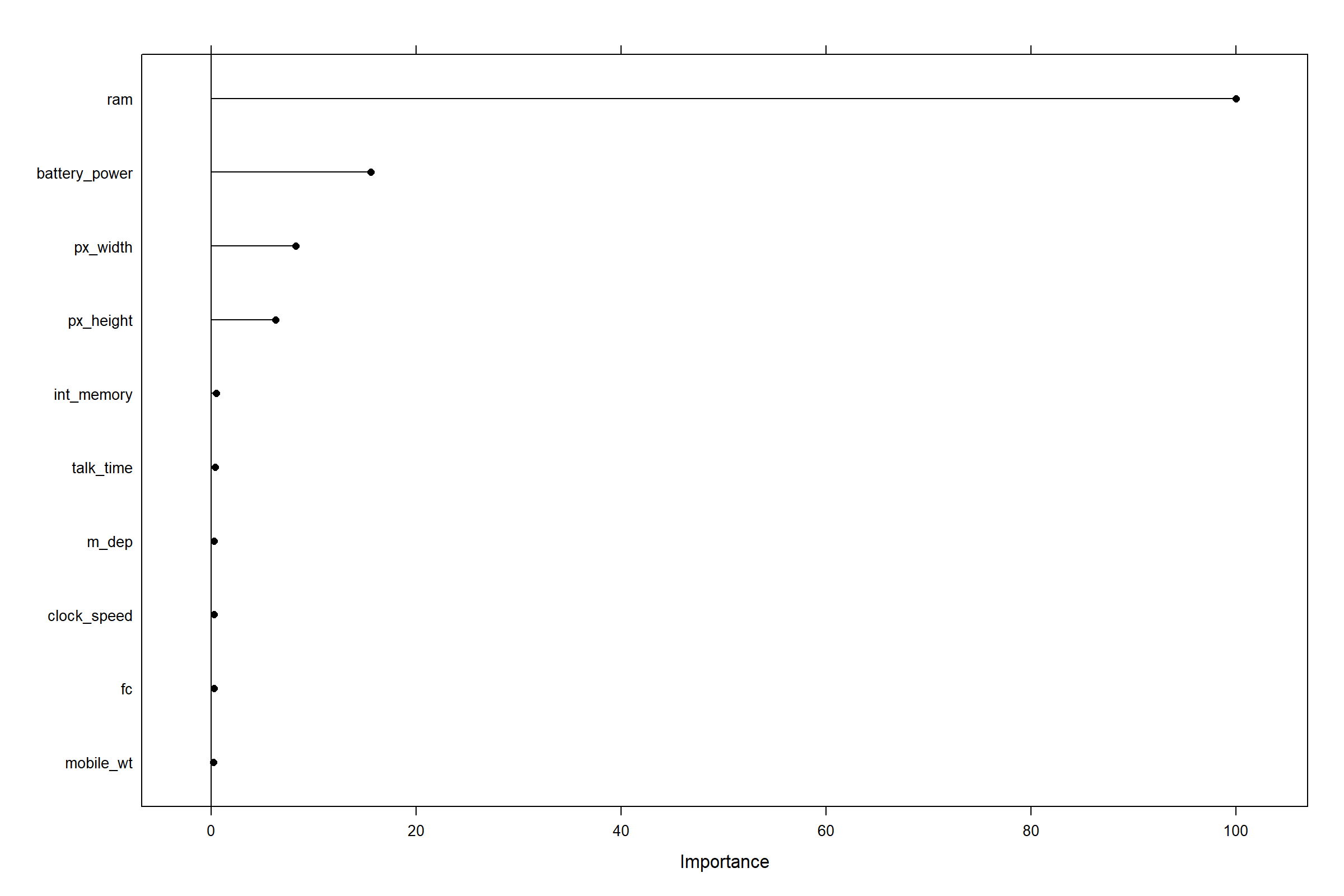
pred.glm = predict(glm.fit, newdata = df[-train\_index,], type = "prob")[,1]  
roc.glm = pROC::roc(df$price\_range[-train\_index], pred.glm)  
pred.glmn <- predict(glmn.fit, newdata = df[-train\_index,], type = "prob")[,1]  
roc.glmn <- pROC::roc(df$price\_range[-train\_index], pred.glmn)  
pred.mars <- predict(mars.fit, newdata = df[-train\_index, ], type = "prob")[,1]  
roc.mars <- pROC::roc(df$price\_range[-train\_index], pred.mars)  
pred.gam = predict(gam.fit, newdata = df[-train\_index,], type = "prob")[,1]  
roc.gam = pROC::roc(df$price\_range[-train\_index], pred.gam)

pred.bagging = predict(bagging.fit, newdata = df[-train\_index, ], type = "prob")[,1]  
roc.bagging = pROC::roc(df$price\_range[-train\_index], pred.bagging)  
pred.rf = predict(rf.fit, newdata = df[-train\_index, ], type = "prob")[,1]  
roc.rf = pROC::roc(df$price\_range[-train\_index], pred.rf)  
pred.gbm = predict(gbm.fit, newdata = df[-train\_index, ], type = "prob")[,1]  
roc.gbm = pROC::roc(df$price\_range[-train\_index], pred.gbm)  
pred.svm = predict(svm.fit, newdata = df[-train\_index, ], type = "prob")[,1]  
roc.svm = pROC::roc(df$price\_range[-train\_index], pred.svm)  
  
plot(roc.glm, col = 1)   
plot(roc.glmn, add = TRUE, col = 2)  
plot(roc.mars, add = TRUE, col = 3)  
plot(roc.gam, add = TRUE, col = 4)  
plot(roc.bagging, add = TRUE, col = 5)   
plot(roc.rf, add = TRUE, col = 6)  
plot(roc.gbm, add = TRUE, col = 7)  
plot(roc.svm, add = TRUE, col = 8)  
auc <- c(roc.glm$auc[1], roc.glmn$auc[1], roc.mars$auc[1], roc.gam$auc[1], roc.bagging$auc[1], roc.rf$auc[1], roc.gbm$auc[1], roc.svm$auc[1])  
modelNames <- c("Glm", "Glmn", "MARS", "GAM", "Bagging", "RF","Adaboost", "SVM")   
legend("bottomright",   
 legend = paste0(modelNames, ": ",   
 round(auc,3)),   
 col = 2:4, lty = 1, cex=0.8, pt.cex = 1)



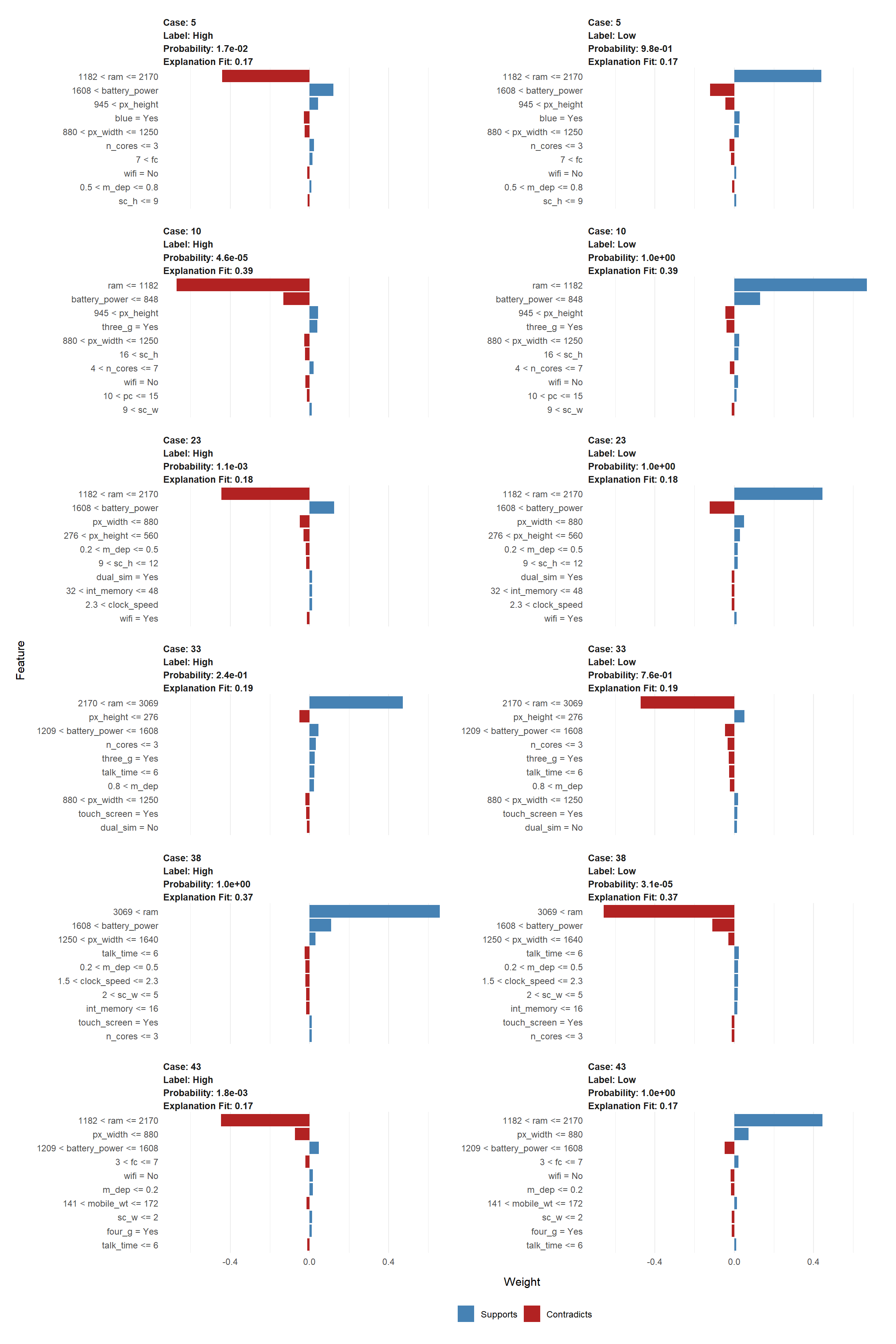
## Global Importance

gbmImp <- varImp(gbm.fit, scale = TRUE)  
plot(gbmImp, top = 10)



## LIME

explainer.rf <- lime(df[train\_index, -21], gbm.fit)  
new\_obs = df[-train\_index, -21][1:6, ]  
explaination.obs = explain(new\_obs,   
 explainer = explainer.rf,  
 n\_features = 10,  
 n\_labels = 2)  
plot\_features(explaination.obs)



## Prediction error

pred.gbm.train = predict(gbm.fit, newdata = df[train\_index, ], type = "prob")[, 1]  
train\_df$pred.gbm = pred.gbm.train  
cp <- cutpointr(train\_df, pred.gbm, price\_range,   
 method = maximize\_metric, metric = sum\_sens\_spec)  
summary(cp)

## Method: maximize\_metric   
## Predictor: pred.gbm   
## Outcome: price\_range   
## Direction: >=   
##   
## AUC n n\_pos n\_neg  
## 1 1600 800 800  
##   
## optimal\_cutpoint sum\_sens\_spec acc sensitivity specificity tp fn fp tn  
## 0.459 2 1 1 1 800 0 0 800  
##   
## Predictor summary:   
## Data Min. 5% 1st Qu. Median Mean 3rd Qu. 95% Max. SD  
## Overall 1.64e-05 3.86e-05 1.48e-04 0.449148 0.4989 0.99988 1.000 1.00 0.4867  
## High 4.59e-01 8.93e-01 9.99e-01 0.999885 0.9819 0.99995 1.000 1.00 0.0628  
## Low 1.64e-05 3.18e-05 5.85e-05 0.000148 0.0159 0.00197 0.109 0.44 0.0535  
## NAs  
## 0  
## 0  
## 0

test\_df$pred.gbm = as.factor(ifelse(pred.gbm > cp$optimal\_cutpoint, "High", "Low"))  
cft = confusionMatrix(test\_df$pred.gbm, test\_df$price\_range)  
print(cft)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction High Low  
## High 192 4  
## Low 8 196  
##   
## Accuracy : 0.97   
## 95% CI : (0.948, 0.984)  
## No Information Rate : 0.5   
## P-Value [Acc > NIR] : <2e-16   
##   
## Kappa : 0.94   
##   
## Mcnemar's Test P-Value : 0.386   
##   
## Sensitivity : 0.960   
## Specificity : 0.980   
## Pos Pred Value : 0.980   
## Neg Pred Value : 0.961   
## Prevalence : 0.500   
## Detection Rate : 0.480   
## Detection Prevalence : 0.490   
## Balanced Accuracy : 0.970   
##   
## 'Positive' Class : High   
##

* Since we have response variables, we consider the problem to be supervised learning. Therefore, the unsupervised learning methods (such as clustering and deep learning) are not suitable for this problem.
* For the models we learned in the second half of this semester, we start with Bagging for classification problem. Bagging is a method which use bootstrap to reduce the variance. Though bagging can improve the predictive accuracy, it may be difficult to interpret since it’s impossible to represent the results as a single tree. In the classification setting, we used Gini index to obtain the variable importance information, predictors with the larger total decrease in the Gini index are considered more important. Here, we use caret to do grid search, then choose the node size with the highest ROC value.
* After analyzing the variable importance in the bagging model, we realize that the variable ram is of great importance, which may cause bagging less valid in decreasing the variance since all of the bagged trees will look quite similar with top split at ram. Therefore, we fit a random forest model, which is an improved method of bagging by decorrelating the trees. It only consider a subset of predictors at each split so as to make the bagged trees less correlated with each other. After grid search, the optimal random forest model is to use 8 random selected predictors in each split, with a minimum node size 6.
* Next, because the outcome of the mobile dataset is binary, we fit a gbm model to do boosting. Boosting is quite similar to bagging except it build trees using information from the previous tree. There are three tuning parameters in boosting: The number of trees B, which are selected using cross-validation; Shrinkage parameter which controls the learning rate; And d, the number of splits in each tree which controls the complexity. From the result, we can see that the optimal situation is reached when B = 5000, = 0.002, d = 5.
* Now consider to fit another classic classification model: support vector machine with radical kernel. There are two tuning parameters for SVM, cost and sigma. Here, we only tune on cost.
* In order to predict the high cost phone, we decided to build a binary classification model. We randomly divided our dataset into two data sets before training the classification algorithms: the training and the test sets. The training and test sets each included 80% and 20% of the total data, respectively.
* The parameters of each algorithm were determined based on the classification performance of the training set as measured by five-fold cross-validation. On the test set, the performance of all algorithms was tested and compared. We evaluated and compared the results of five different algorithms since different classification methods are better suited to different types of data. Bagging, random forest, ada boosting, and radical kernel SVM are among the models under consideration.
* We plotted the ROC curves of all the different algorithms on the test dataset. Over all reasonable sensitivity thresholds and recall thresholds, the ada boosting model is consistently better than all the other models. The feature importance of the ADA boosting model is scaled between 0 and 100. Random access memory (RAM) is the most important predictor since the importance value goes to 100. Battery power is 20% as important as RAM. Pixel height and pixel width are each around 10% as important as RAM. All the other predictors are less than 5% as important as RAM.
* For the first six test cases and label combinations, we utilized LIME to visually represent the explanations for the relationship between mobile phone price level and features. Positively associated features are displayed in blue, while negatively correlated features are displayed in red. For example, case 30, which refers to the row 30 of the test data, has the highest explanation fit 0.40. Label which is high means this case is for predicting the high price mobile phone. ‘RAM smaller than 1209’ feature which is red color implies the phone with this ‘RAM smaller than 1209’ feature has large possibility that it does not belong to the high price phone. For case 18, label which is high is for predicting the high price mobile phone. ‘RAM > 3033’ feature which is blue color implies the phone with this ‘RAM > 3033’ feature has large possibility that it belongs to high price phone. We can also observe that all of the predictors for the phone pricing outcome selected the same features, showing that these are important features both locally and globally. For example, these features include ram, battery power, talking time, pixel height, pixel width, 3g internet, WiFi, Bluetooth.
* We selected to maximize the sum of sensitivity and specificity in order to determine the best cut point for the prediction probability. On the training dataset, the best cut point is 0.459, which yields an accuracy of 1, a sensitivity of 1, and a specificity of 1. On the test dataset, we have an accuracy of 0.97, sensitivity of 0.96, and specificity of 0.98 using the optimal cut point. As a consequence, our model appears to be extremely effective in predicting high prices for phones.