```r
# library(reticulate)
library(caret)
library(tidyverse)
library(ggpubr)
library(doParallel)
library(ranger)
library(pROC)
library(gbm)
library(pdp)
library(lime)
library(cutpointr)
```

```r
# read data
df = read.csv("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")))
```

```r
# 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, ]
```

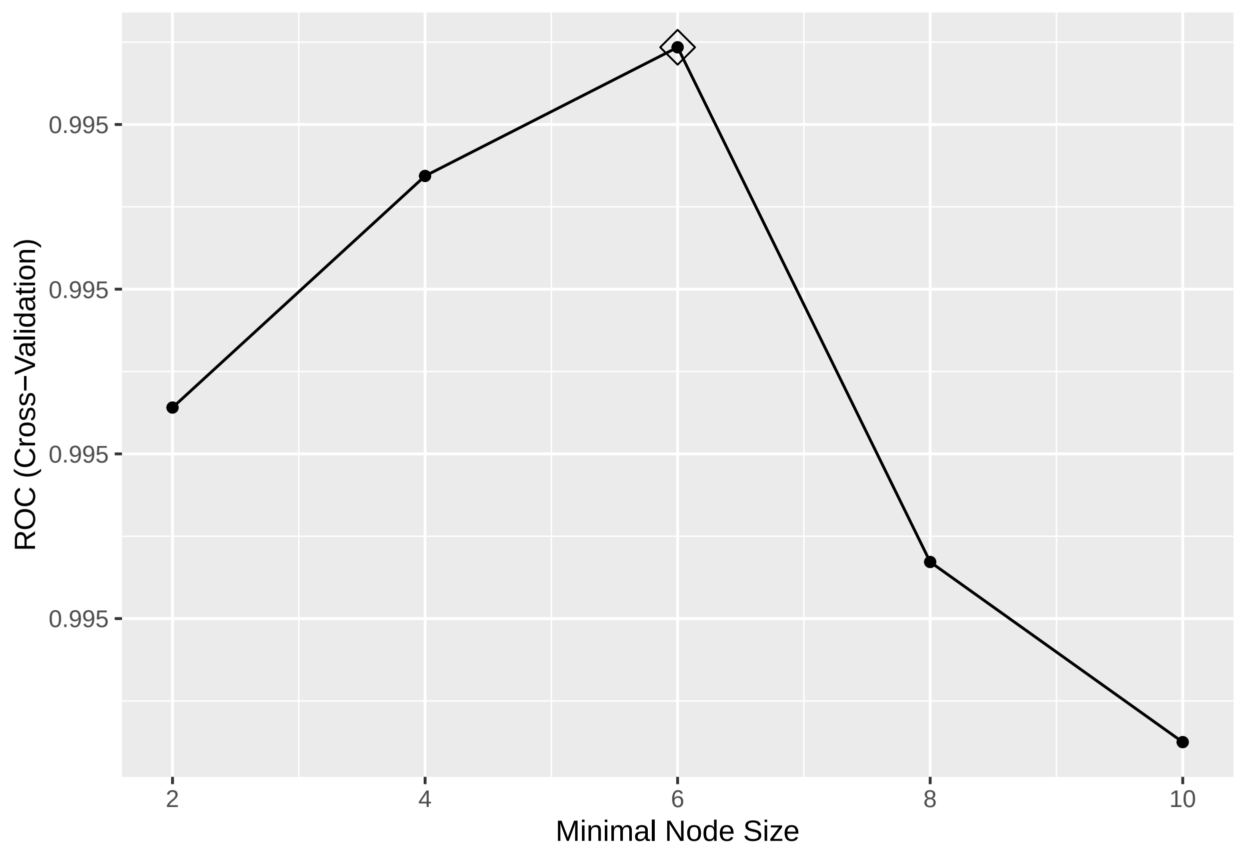
```r
# user parallel to accelarate
cl <- makePSOCKcluster(4)
registerDoParallel(cl)
```

Bagging

```r
ctrl <- trainControl(method = "cv", classProbs = TRUE,
                     summaryFunction = twoClassSummary)

bagging.grid <- expand.grid(mtry = 20,
                       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)
```

```r
ggplot(bagging.fit, highlight = TRUE)
```

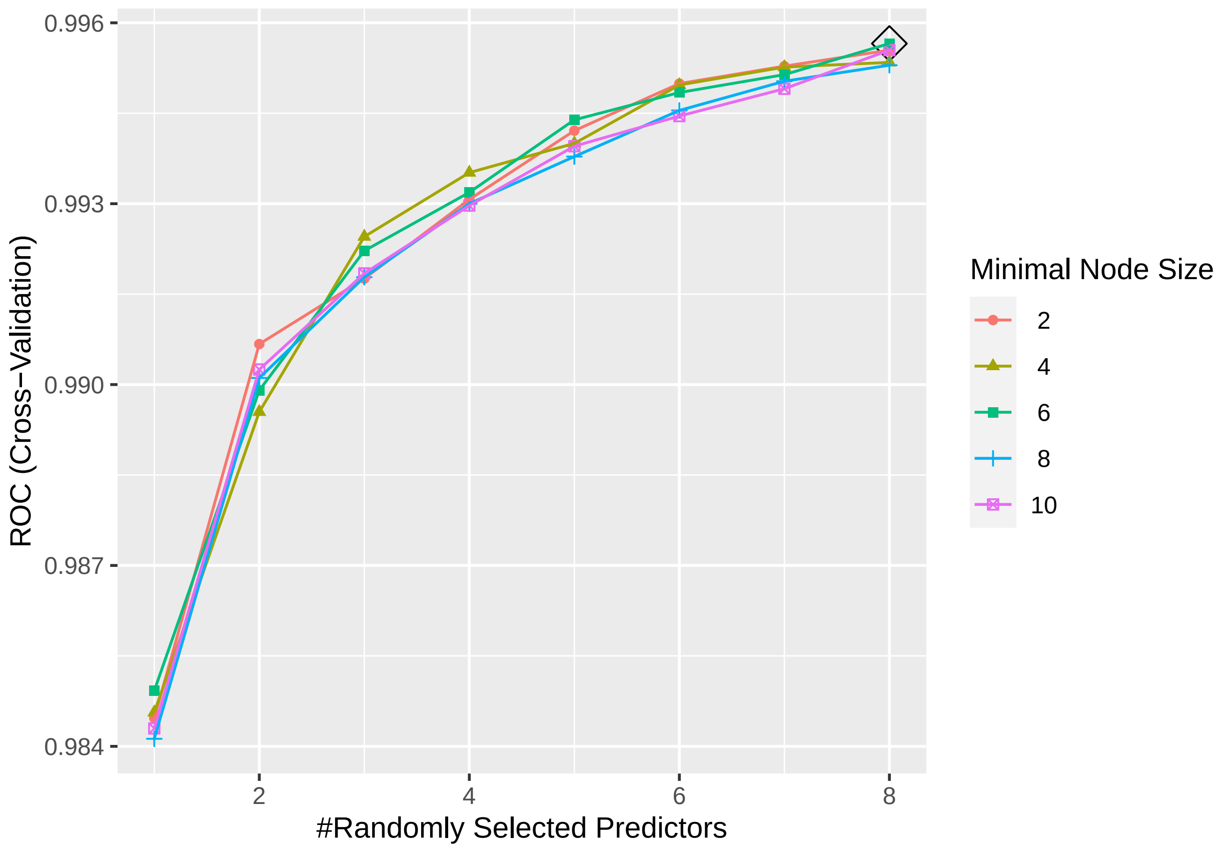


Random forest

```r
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)
```

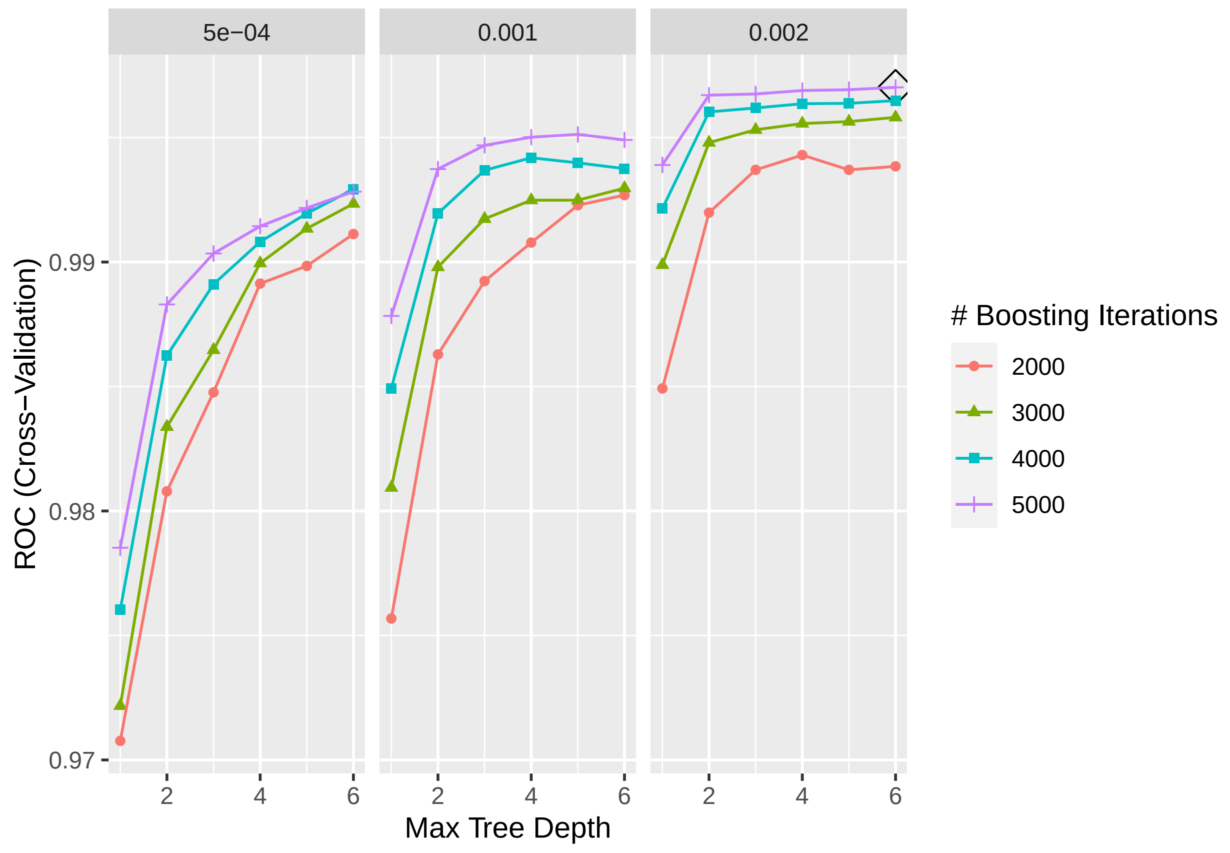
```r
ggplot(rf.fit, highlight = TRUE)
```



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)
```

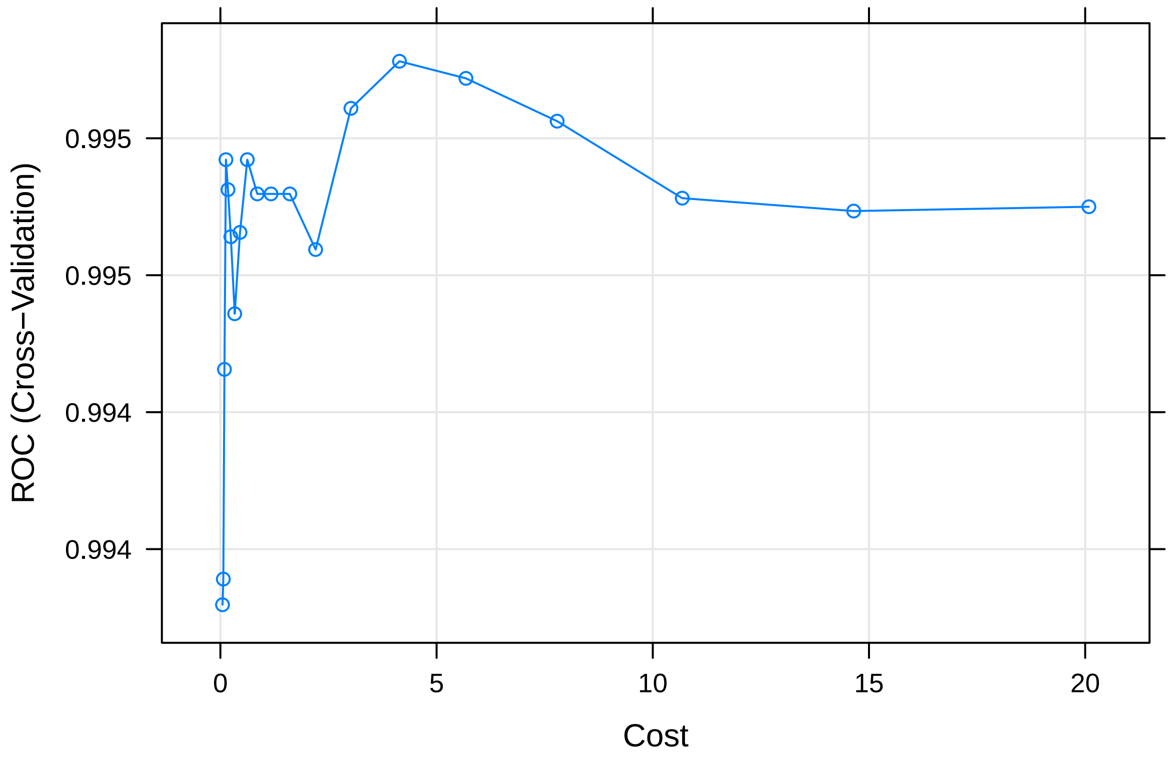
```
ggplot(gbm.fit, highlight = TRUE)
```



SVM

```
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)
```

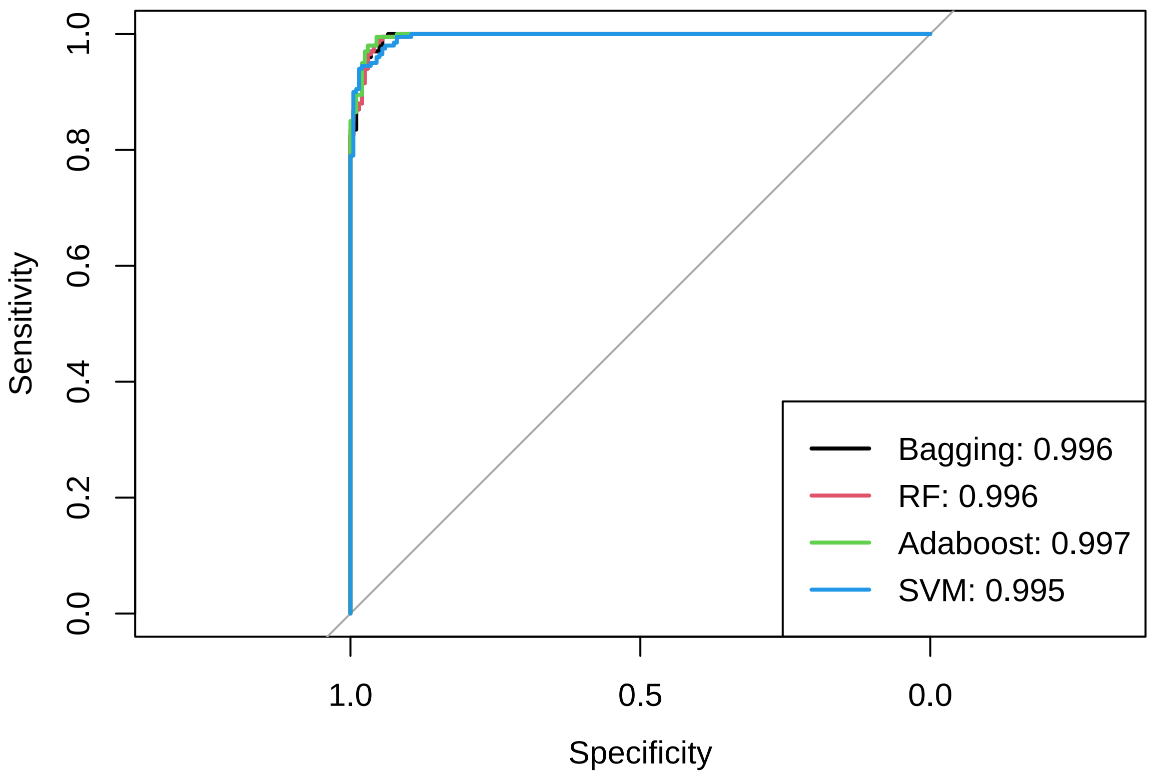
```
plot(svm.fit)
```



ROC camparison

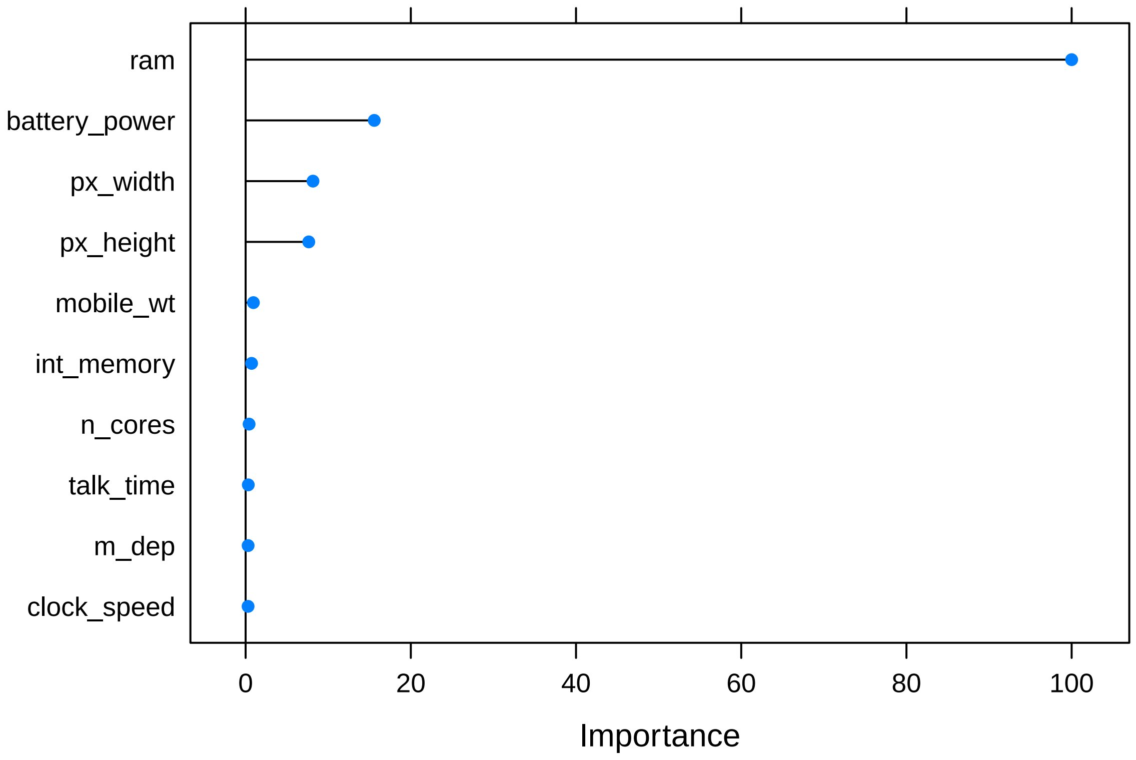
```
pred.bagging = predict(bagging.fit, newdata = df[-train_index, ], type = "prob")[,1]
roc.bagging = pROC::roc(df$price_range[-train_index], pred.bagging)
pred.rf = predict(rf.fit, newdata = df[-train_index, ], type = "prob")[,1]
roc.rf = pROC::roc(df$price_range[-train_index], pred.rf)
pred.gbm = 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.bagging, col = 1)
plot(roc.rf, add = TRUE, col = 2)
plot(roc.gbm, add = TRUE, col = 3)
plot(roc.svm, add = TRUE, col = 4)
auc <- c(roc.bagging$auc[1], roc.rf$auc[1], roc.gbm$auc[1], roc.svm$auc[1])
modelNames <- c("Bagging", "RF","Adaboost", "SVM")
legend("bottomright",
       legend = paste0(modelNames, ": ",
                       round(auc,3)),
       col = 1:4, lwd = 2)
```



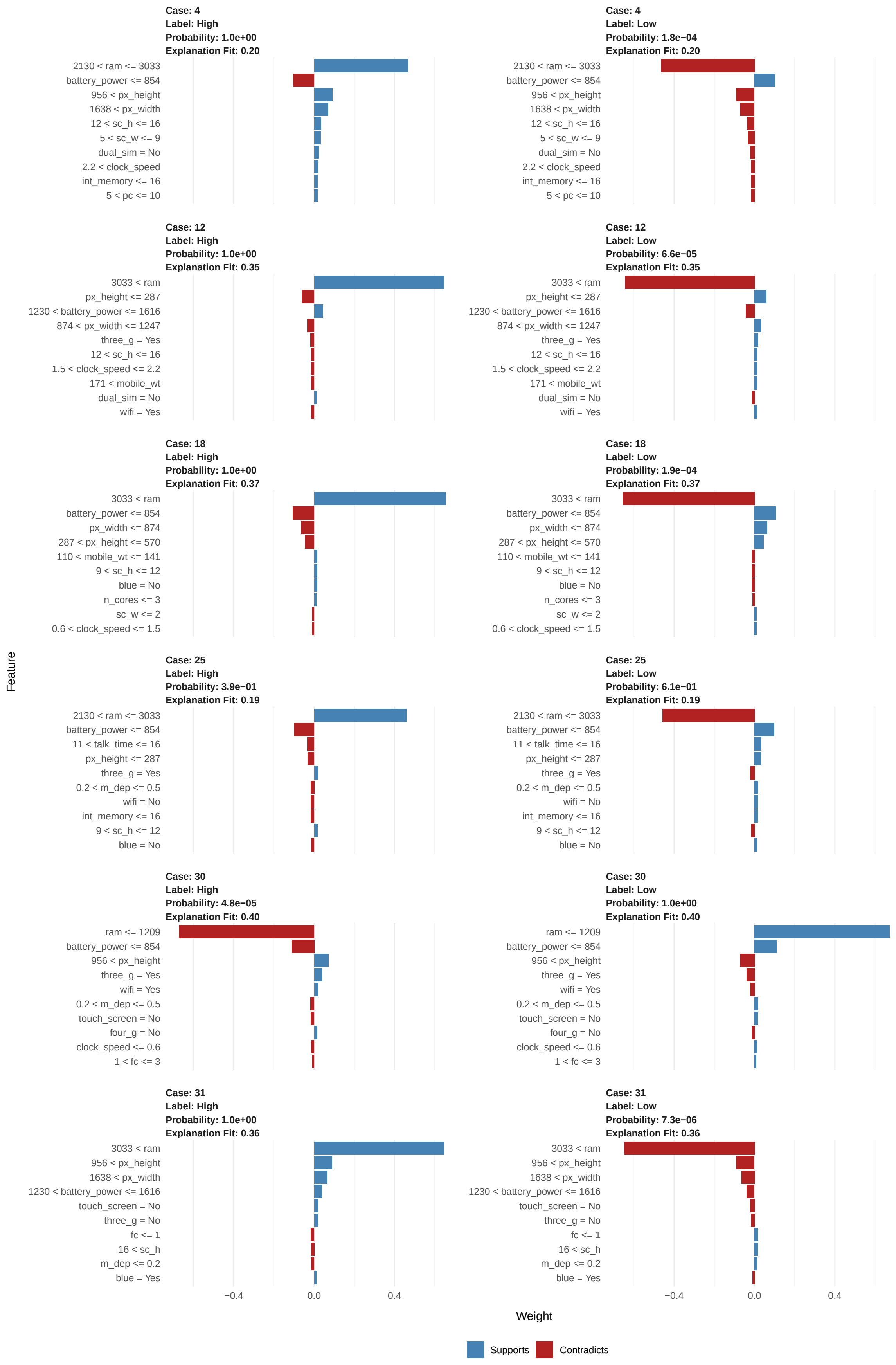
Global Importance

```
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)
```



Case: 4
Label: High
Probability: 1.0e+00
Explanation Fit: 0.20
2130 < ram <= 3033
battery_power <= 854
956 < px_height
1638 < px_width
12 < sc_h <= 16
5 < sc_w <= 9
dual_sim = No
2.2 < clock_speed
int_memory <= 16
5 < pc <= 10
Case: 4
Label: Low
Probability: 1.8e−04
Explanation Fit: 0.20
Case: 12
Label: High
Probability: 1.0e+00
Explanation Fit: 0.35
3033 < ram
px_height <= 287
1230 < battery_power <= 1616
874 < px_width <= 1247
three_g = Yes
12 < sc_h <= 16
1.5 < clock_speed <= 2.2
171 < mobile_wt
dual_sim = No
wifi = Yes
Case: 12
Label: Low
Probability: 6.6e−05
Explanation Fit: 0.35
Case: 18
Label: High
Probability: 1.0e+00
Explanation Fit: 0.37
3033 < ram
battery_power <= 854
px_width <= 874
287 < px_height <= 570
110 < mobile_wt <= 141
9 < sc_h <= 12
blue = No
n_cores <= 3
sc_w <= 2
0.6 < clock_speed <= 1.5
Case: 18
Label: Low
Probability: 1.9e−04
Explanation Fit: 0.37
Case: 25
Label: High
Probability: 3.9e−01
Explanation Fit: 0.19
2130 < ram <= 3033
battery_power <= 854
11 < talk_time <= 16
px_height <= 287
three_g = Yes
0.2 < m_dep <= 0.5
wifi = No
int_memory <= 16
9 < sc_h <= 12
blue = No
Case: 25
Label: Low
Probability: 6.1e−01
Explanation Fit: 0.19
Case: 30
Label: High
Probability: 4.8e−05
Explanation Fit: 0.40
ram <= 1209
battery_power <= 854
956 < px_height
three_g = Yes
wifi = Yes
0.2 < m_dep <= 0.5
touch_screen = No
four_g = No
clock_speed <= 0.6
1 < fc <= 3
Case: 30
Label: Low
Probability: 1.0e+00
Explanation Fit: 0.40
Case: 31
Label: High
Probability: 1.0e+00
Explanation Fit: 0.36
3033 < ram
956 < px_height
1638 < px_width
1230 < battery_power <= 1616
touch_screen = No
three_g = No
fc <= 1
16 < sc_h
m_dep <= 0.2
blue = Yes
Case: 31
Label: Low
Probability: 7.3e−06
Explanation Fit: 0.36
−0.4
0.0
0.4
Feature
Weight
Supports
Contradicts

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
##             0.516             2 0.999           1       0.999 800  0  1 799
##
## Predictor summary:
##     Data     Min.       5%  1st Qu. Median   Mean 3rd Qu.    95%  Max.     SD NAs
##  Overall 1.05e-05 2.72e-05 1.00e-04 0.5190 0.4995 0.99992 1.0000 1.000 0.4888   0
##     High 5.16e-01 9.14e-01 9.99e-01 0.9999 0.9854 0.99996 1.0000 1.000 0.0542   0
##      Low 1.05e-05 2.23e-05 4.06e-05 0.0001 0.0135 0.00142 0.0863 0.576 0.0491   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  194   4
##       Low     6 196
##
##                Accuracy : 0.975
##                  95% CI : (0.955, 0.988)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.95
##
##  Mcnemar's Test P-Value : 0.752
##
##             Sensitivity : 0.970
##             Specificity : 0.980
##          Pos Pred Value : 0.980
##          Neg Pred Value : 0.970
##              Prevalence : 0.500
##          Detection Rate : 0.485
##    Detection Prevalence : 0.495
##       Balanced Accuracy : 0.975
##
##        'Positive' Class : High
##
```

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. 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. Positively associated features are displayed in blue, while negatively correlated features are displayed in red. We can 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.

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 0.516, which yields an accuracy of 0.999, a sensitivity of 1, and a specificity of 0.999. On the test dataset, we have an accuracy of 0.975, sensitivity of 0.97, 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.