STATS - HW 4

Wenjing Li

3/12/2020

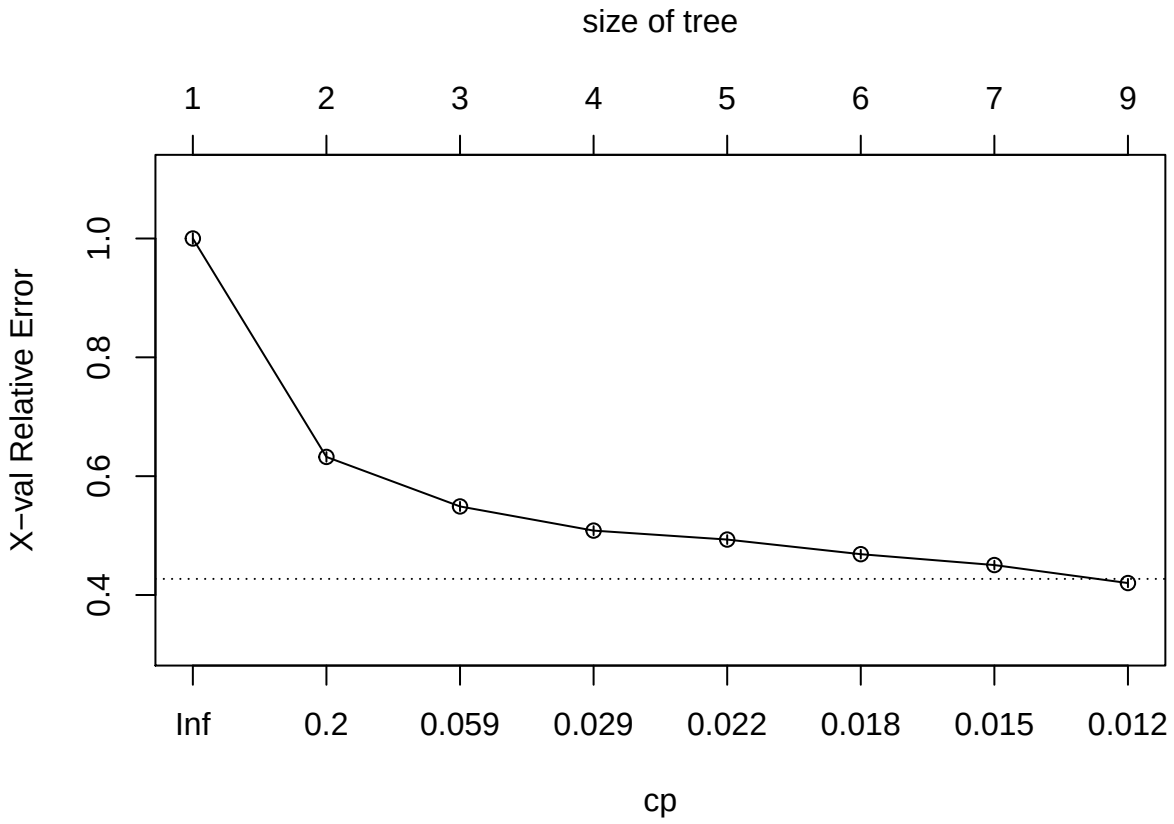
Question 2

2(1)

Firstly, I used the gini method to fit the model and find the best cp value based on the cross vaildation generated from the "plotcp" function.

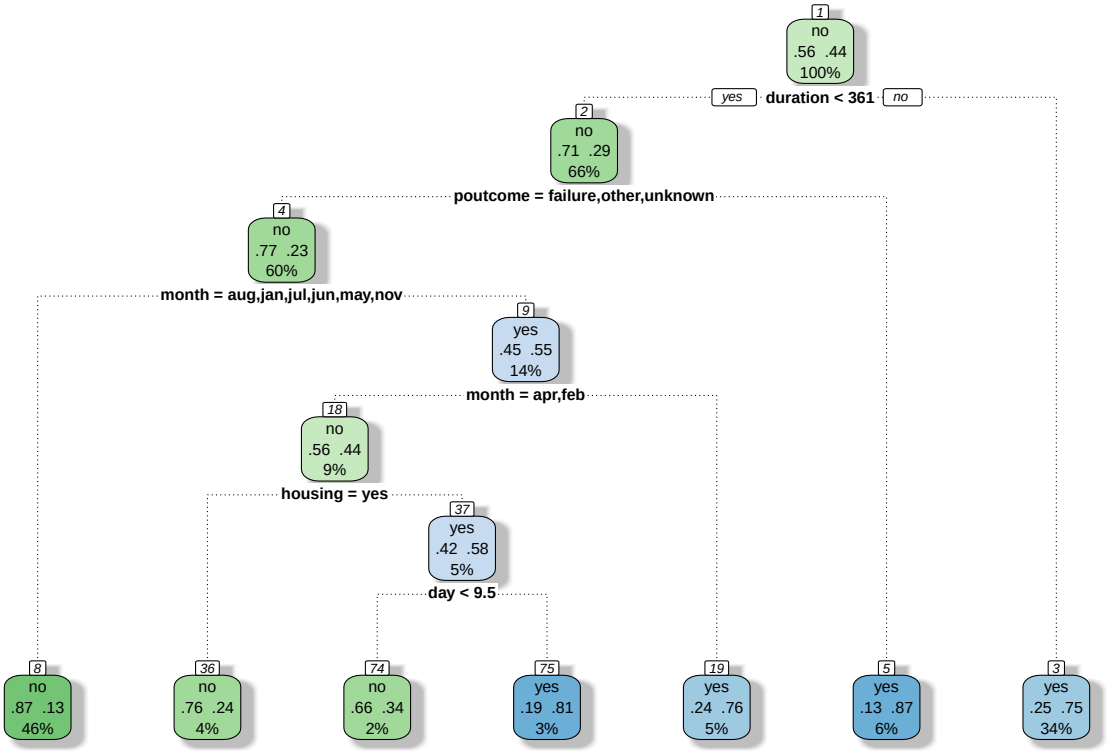
```
prop = table(dat_train$deposit) / dim(dat_train)[1]
w = rep(1/prop[2], dim(dat_train)[1])
w[which(dat_train$deposit == "yes")] = 1/prop[1]

set.seed(123)
tree_gini = rpart(deposit ~ ., data = dat_train,
                  parms = list(split = "gini"), method = "class", weights = w)
plotcp(tree_gini)
```



The best cp value from the plot above is 0.015. Next, I used the training data set to fit the model and get a tree plot as below.

```
set.seed(123)
tree_gini = rpart(deposit ~ ., data = dat_train,
                  parms = list(split = "gini"), method = "class", cp = 0.015, weights = w)
fancyRpartPlot(tree_gini)
```



The tree model contains 7 terminal nodes. Then, I computed the tesing error of the model.

```
tree_pred = predict(tree_gini, dat_test, type = "class")
table(tree_pred, dat_test$deposit)
```

```
##
## tree_pred   no  yes
##       no  1390  272
##       yes  355 1332
```

Of all the 'no' clients of the test set, 16.36 percentage was misclassified and of all the 'yes' clients of the test set, 21.04 percentage was misclassified as 'no'.

And the overall testing error can be computed as:

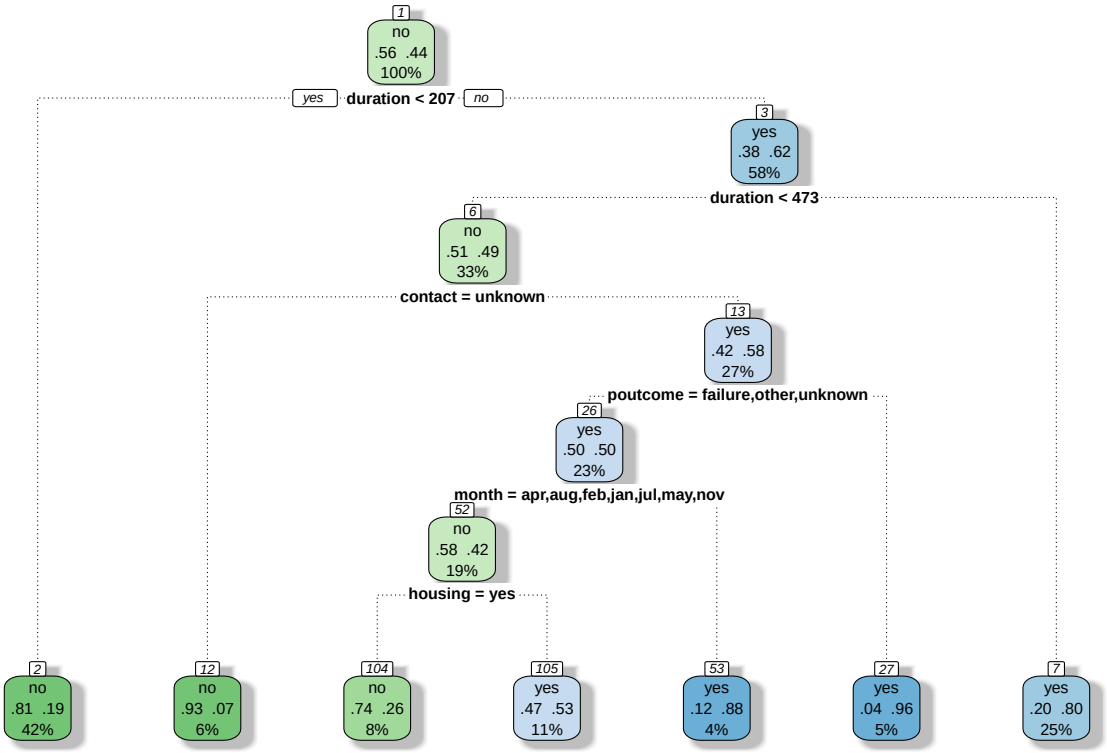
```
sum(tree_pred != dat_test$deposit) / dim(dat_test)[1]
```

```
## [1] 0.1872201
```

So the gini method give us a model with testing error as 0.1872.

However, using the info seems give us a better model as below.

```
set.seed(123)
tree.info = rpart(deposit ~ ., data = dat_train,
                  parms = list(split = "information"), cp = 0.016, weights = w)
fancyRpartPlot(tree.info)
```




```
train.pred = predict(tree.info, dat_test, type="class", cp = 0.016)
sum(train.pred != dat_test$deposit) / dim(dat_test)[1]
```

```
## [1] 0.2096148
```

```
#create the error table
table(train.pred, dat_test$deposit)
```

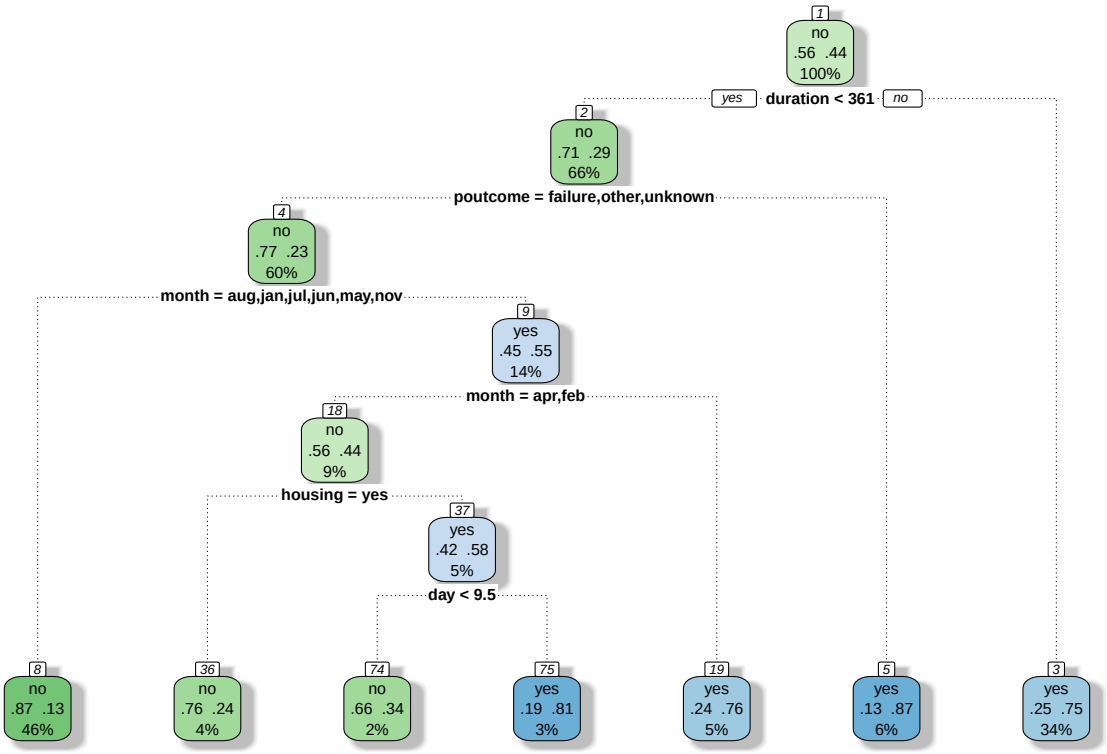
```
##
## train.pred   no  yes
##        no  1402  359
##        yes  343 1245
```

Of all the 'no' clients of the test set, 20.39 percentage was misclassified and of all the 'yes' clients of the test set, 21.60 percentage was misclassified as 'no'. And the testing error is 0.2096.

2(2)

Using the best model from the task one and get the subtree as below.

```
#another method:
#tree.info1 <- prp(tree.info,snip=TRUE)$obj
#prp(tree.info1)
#however, the method above need interactive which is hard to complish in markdown file
#use the method below
#
fancyRpartPlot(tree_gini)
```



From the subplot, the variables that are used in this model are: duration, poutcome, month, housing, day. There variables tend to be more important and useful to split training data nodes.

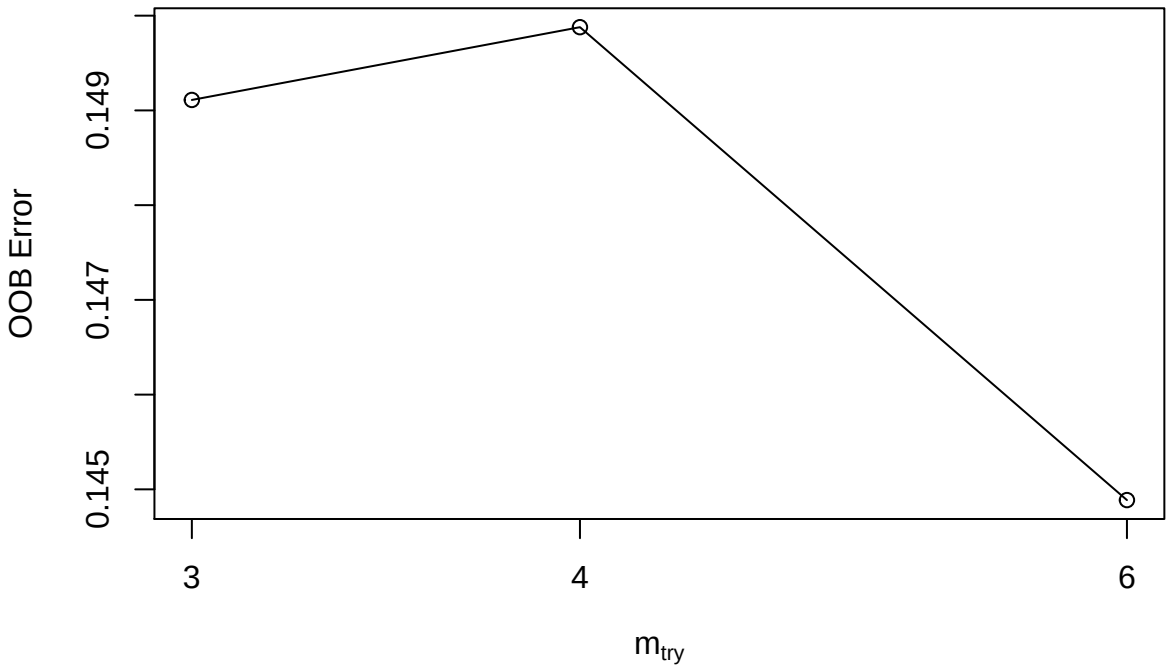
2(3)

Step 1:

Since we evaluate the model based on error(accuracy) instead of AUC, then this is a good way to tune a model, as the selection considers (OOB) error.

```
#random forest - choose parameter value
set.seed(123)
mytry <- tuneRF(dat_train[,-17], dat_train[,17], stepFactor = 1.5, weights=w)
```

```
## mtry = 4  OOB error = 14.99%
## Searching left ...
## mtry = 3     OOB error = 14.91%
## 0.005123826 0.05
## Searching right ...
## mtry = 6     OOB error = 14.49%
## 0.03330487 0.05
```



From the plot above, I am going to choose 6 as mtry value since it is reasonablely small.

Step 2:

Build the random forest model.

- ntree: Number of trees to grow. We need a large number to make sure the input gets predicted. With the increasement of the value of ntree, the error for beth type tends to become smaller and become stable. However, we usually choose ntree = 500, since larger trees work better and become more computational-consuming.
- mtry: Number of variables randomly sampled as candidates at each split. mtry larger, the model tends to become more complicated, and the testing error tends to be smaller at first and then larger until the end. The error of "no" becomes smaller at first and then tends to be larger, while the error of "yes" continously becomes "larger".
- nodezise: Minimum size of terminal nodes. Setting this number larger causes smaller trees to be grown and that helps to reduce the effect of overfitting. If the value of nodesize become larger, the model tends to avoid overfitting at first, and then becomes underfitting if the value of nodesize is large enough. So with the nodesize becoming larger, the error tends to decrease at first and then increase. The error

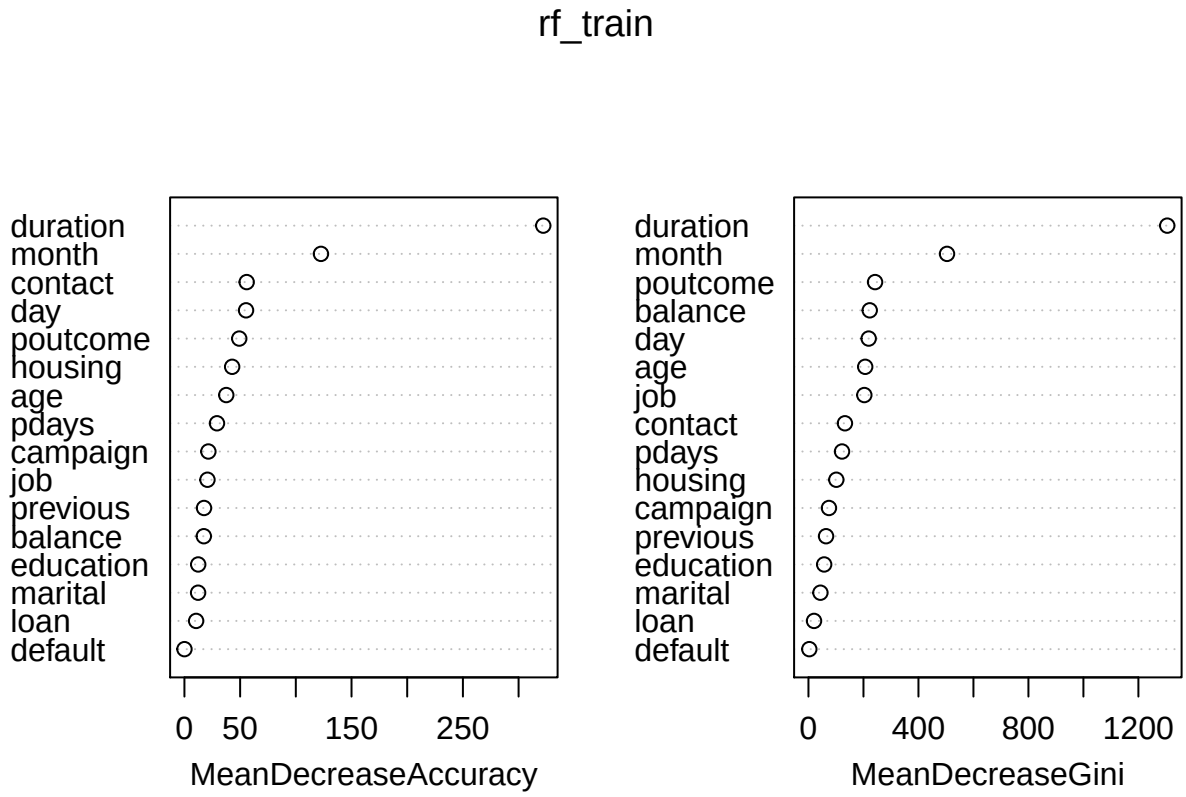
of "no" becomes larger, as well as the error of "yes" becomes smaller at first and then becomes larger as well.

```
set.seed(123)
rf_train = randomForest(deposit ~ ., data = dat_train, mtry = 6, importance = TRUE, nodesize=5, weights=
importance(rf_train)
```

```
##                   no        yes MeanDecreaseAccuracy MeanDecreaseGini
## age        36.504920  15.503781          37.55279008       206.344728
## job        21.060830   7.422784          20.68394938       203.194644
## marital     4.379978  11.840854          12.27261135        43.475421
## education  11.288808   5.343467          12.37871393        57.118350
## default    -2.548482   1.507447           0.03332761         3.104959
## balance    13.545593  11.014755          17.41314687       222.849690
## housing    33.557410  28.603366          42.84990157       101.280541
## loan        2.711209  10.922024          10.51996469        20.248080
## contact    50.033724  20.970828          55.89425530       132.591616
## day        57.417388  11.583935          55.33617549       219.052978
## month     106.106002  50.843090         122.58633287       503.921928
## duration  226.366697 275.646723         322.13885837      1304.878740
## campaign   12.823946  18.021405          21.49851532        74.589999
## pdays      21.239061  21.343438          29.22253394       122.034360
## previous   16.892740   9.507203          17.59153209        63.936986
## poutcome   52.749554  15.220289          49.23165023       242.232094
```

From the MeanDecreaseAccuracy, we can get the top five most important variables are: duration, month, contact, day and poutcome, which is similar to the result from the tree(info) model. And we can take a look at the importance plot:

```
varImpPlot(rf_train)
```



Step 3:

Evaluate on the test data.

```
rf_pred = predict(rf_train, newdata = dat_test)
table(rf_pred, dat_test$deposit)
```

```
##
## rf_pred   no  yes
##     no  1445  181
##     yes  300 1423
```

Of all the 'no' clients of the test set, 11.06 percentage was misclassified and of all the 'yes' clients of the test set, 17.63 percentage was misclassified as 'no'.

```
mean(rf_pred != dat_test$deposit)
```

```
## [1] 0.143625
```

The testing error is 0.1436.

2(3)

We can use the logit boosting as the first glance.

```r
#boosting
##process data
dat_train$deposit = ifelse(dat_train$deposit == "yes", 1, 0)
dat_test$deposit = ifelse(dat_test$deposit == "yes", 1, 0)

##build model
set.seed(123)
logit_spam = gbm(deposit ~ ., data = dat_train, distribution = "bernoulli",
                 n.trees = 1000, interaction.depth = 3, shrinkage = 0.1, weights=w)
logit_pred_response = predict(logit_spam, newdata = dat_test,
                              n.trees = 1000,
                              type = "response")
logit_pred = ifelse(logit_pred_response > 0.5, 1, 0)
table(logit_pred, dat_test$deposit)
```

```
##
## logit_pred    0    1
##          0 1490  227
##          1  255 1377
```

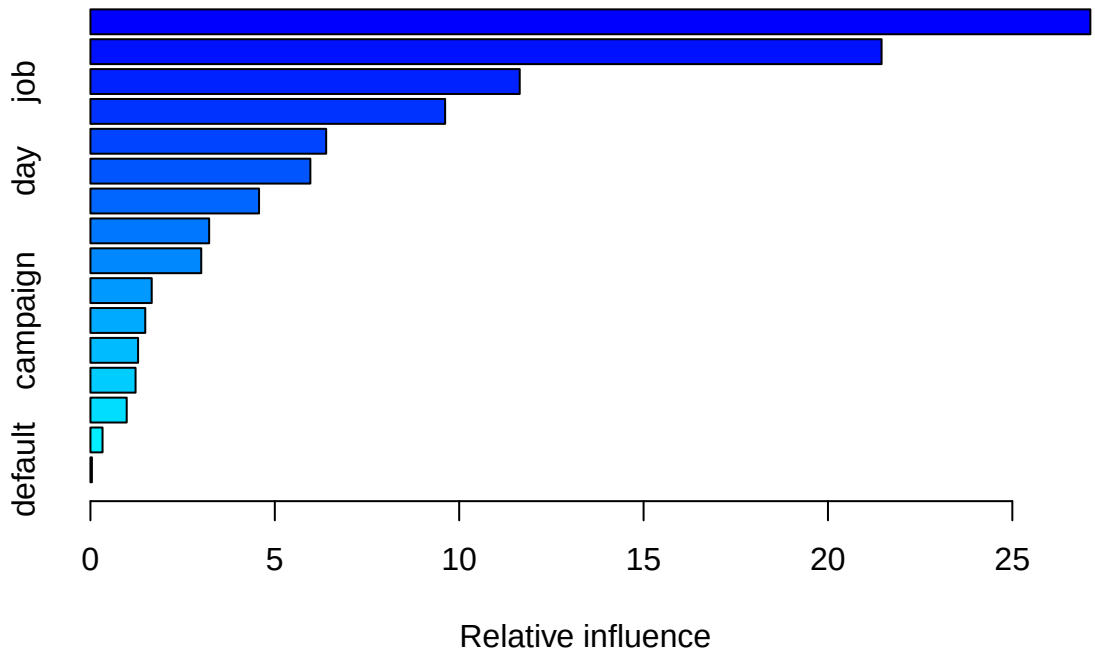
```r
mean(logit_pred != dat_test$deposit)
```

```
## [1] 0.1439236
```

Then we can use the AdaBoost method and see if this method can give us a better result.

- interaction.depth: the maximum depth of each tree. With a larger interaction.depth, the model becomes more accurate. So the errors for both types become smaller at first, and then the errors gradually become larger since the model tends to be overfitting.
- shrinkage: the learning rate. A smaller learning rate typically requires more trees. With a larger shrinkage, the errors become smaller at first and then tends to be larger if the shrinkage is large enough.
- n.trees: the total number of trees to fit. With a larger n.trees, the model becomes more accurate. So the errors for both types become smaller at first, and then the errors gradually become larger since the model tends to be overfitting.

```r
#build the AdaBoost model
set.seed(123)
ada_train = gbm(deposit ~ ., data = dat_train,
                distribution = "adaboost", n.trees = 5000,
                interaction.depth = 3, weights=w)
summary(ada_train)
```




```
##                 var     rel.inf
## duration   duration 27.11585908
## month         month 21.45442647
## job             job 11.64123578
## balance     balance  9.61851119
## age             age  6.39272313
## day             day  5.96395450
## poutcome   poutcome  4.57232318
## pdays         pdays  3.21977150
## contact     contact  3.00445580
## education education  1.66207796
## campaign   campaign  1.48710170
## housing     housing  1.29259511
## marital     marital  1.22340886
## previous   previous  0.98343886
## loan           loan  0.32841783
## default     default  0.03969906
```

From the influence plot above, we can find the variable "job" has a great influence as 11.83. The variable "poutcome" has a middle class influence as 4.62 while the "housing" has a small influence as 1.51. The influence result is reasonable since whether a person has deposit or not depends more on his job other than whether he has housing loan or not. And from the chart above, we can find the top five variables are: duration, month, job, balance and age, which is also similar to the result above, though the variable "balance" tends to be more important here.

Next, I am going to evaluate on the testing data:

```
ada_pred = predict(ada_train, newdata = dat_test,
                   n.trees = 5000, type = "response")
ada_pred = ifelse(ada_pred > 0.5, 1, 0)
table(ada_pred, dat_test$deposit)
```

```
##
## ada_pred    0    1
##        0 1477  235
##        1  268 1369
```

```
mean(ada_pred != dat_test$deposit)
```

```
## [1] 0.1501941
```

Of all the 'no' clients of the test set, 13.73 percentage was misclassified and of all the 'yes' clients of the test set, 16.37 percentage was misclassified as 'no'. And the testing error is 0.1502.

We summarize some representative models testing error results in below:

- tree(info): 0.1872
- random forest: 0.1436
- logit boosting: 0.1439
- adaboost: 0.1502

To wrap up, the best model is logit boosting.