HW4 ML

Yaqi

5/8/2019

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.

```r
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------------------------------- tidyverse 1.2.

## v ggplot2 3.1.0       v purrr   0.3.0
## v tibble  2.0.1       v dplyr   0.8.0.1
## v tidyr   0.8.3       v stringr 1.3.1
## v readr   1.3.1       v forcats 0.3.0

## -- Conflicts ------------------------------------------------------------------ tidyverse_conflicts(
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
library(caret)
```

```
## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift
```

```r
library(gmodels)
library(MLmetrics)
```

```
##
## Attaching package: 'MLmetrics'

## The following objects are masked from 'package:caret':
##
##     MAE, RMSE

## The following object is masked from 'package:base':
##
##     Recall
```

```r
library(C50)
```

1. Credit Analysis using machine learning Load dataset credit.csv first and split the data into training and validation part.

```
credit = read_csv('credit.csv')
```

```
## Parsed with column specification:
## cols(
##   .default = col_character(),
##   months_loan_duration = col_double(),
##   amount = col_double(),
##   installment_rate = col_double(),
##   residence_history = col_double(),
##   age = col_double(),
##   existing_credits = col_double(),
##   dependents = col_double(),
##   default = col_double()
## )
```

```
## See spec(...) for full column specifications.
```

```
credit = data.frame(credit)
```

Recode 1,2 in default to "no" and "yes" respectively and split the data.

```
credit = credit %>%
  mutate(default = if_else(default==1,'No','Yes'))
credit$default = as.factor(credit$default)
# change character variables into levels
credit[,c("checking_balance","credit_history","purpose","savings_balance","employment_length","personal_
# create a list of 80% of the rows in the original dataset we can use for training
validation_index <- createDataPartition(credit$default, p=0.80, list=FALSE)
# select 20% of the data for validation
credit_valid <- credit[-validation_index,]
# use the remaining 80% of data to training and testing the models
credit_train <-credit[validation_index,]


print("Number of rows in our training dataset:")
```

```
## [1] "Number of rows in our training dataset:"
```

```
dim(credit_train)[1]
```

```
## [1] 800
```

```
print("Number of features in our training dataset:")
```

```
## [1] "Number of features in our training dataset:"
```

```
dim(credit_train)[2]-1
```

```
## [1] 20
```

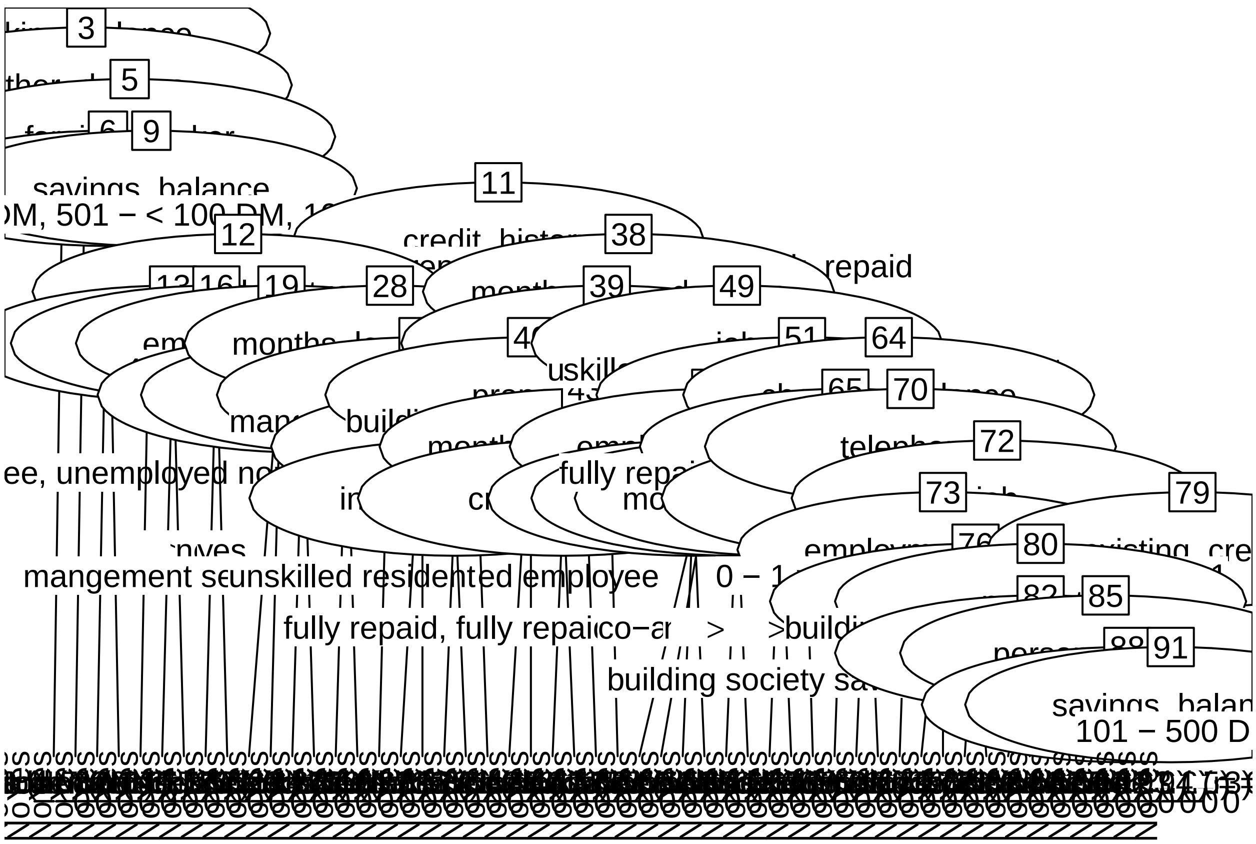
```
# Inspect the class of all the variables
sapply(credit_train, class)
```

```
##     checking_balance months_loan_duration       credit_history
##             "factor"            "numeric"             "factor"
##              purpose               amount      savings_balance
##             "factor"            "numeric"             "factor"
##    employment_length     installment_rate      personal_status
##             "factor"            "numeric"             "factor"
```

```
##        other_debtors    residence_history              property
##             "factor"            "numeric"              "factor"
##                  age     installment_plan               housing
##            "numeric"             "factor"              "factor"
##     existing_credits                  job            dependents
##            "numeric"             "factor"             "numeric"
##            telephone       foreign_worker               default
##             "factor"             "factor"              "factor"
```

Now we can fit a decision tree model to predict the default

```r
tree_mod <- C5.0(x = credit_train[,-21], y = as.factor(credit_train$default))
plot(tree_mod)
```




```r
credit_pred = predict(tree_mod,newdata = credit_valid)
CrossTable(credit_valid$default, credit_pred,
           prop.chisq = FALSE, prop.c = FALSE, prop.r = FALSE, dnn = c('actual default', 'predicted defa
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  200
##
```

```
##
##                 | predicted default
## actual default |        No |       Yes | Row Total |
## ---------------|-----------|-----------|-----------|
##             No |       113 |        27 |       140 |
##                |     0.565 |     0.135 |           |
## ---------------|-----------|-----------|-----------|
##            Yes |        33 |        27 |        60 |
##                |     0.165 |     0.135 |           |
## ---------------|-----------|-----------|-----------|
##   Column Total |       146 |        54 |       200 |
## ---------------|-----------|-----------|-----------|
##
##
```

```
F1_Score(credit_valid$default, credit_pred)
```

```
## [1] 0.7902098
```

The decision tree model using all the variables as predictors we have a F1 Score of 78.2%, which is pretty good. Let's try the boosting decision tree now.

```
tree_boost <- C5.0(x = credit_train[,-21], y = as.factor(credit_train$default),trials = 100)
credit_pred_boost = predict(tree_boost,newdata = credit_valid)
CrossTable(credit_valid$default, credit_pred_boost,
           prop.chisq = FALSE, prop.c = FALSE, prop.r = FALSE, dnn = c('actual default', 'predicted defa
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  200
##
##
##                 | predicted default
## actual default |        No |       Yes | Row Total |
## ---------------|-----------|-----------|-----------|
##             No |       124 |        16 |       140 |
##                |     0.620 |     0.080 |           |
## ---------------|-----------|-----------|-----------|
##            Yes |        37 |        23 |        60 |
##                |     0.185 |     0.115 |           |
## ---------------|-----------|-----------|-----------|
##   Column Total |       161 |        39 |       200 |
## ---------------|-----------|-----------|-----------|
##
##
```

```
F1_Score(credit_valid$default, credit_pred_boost)
```

```
## [1] 0.8239203
```

With the boosting method, we can improve the accuracy to 84.14%

```
tree_boost <- C5.0(x = credit_train[,-21], y = as.factor(credit_train$default),trials = 100,rules = TRU
credit_pred_boost = predict(tree_boost,newdata = credit_valid)
CrossTable(credit_valid$default, credit_pred_boost,
           prop.chisq = FALSE, prop.c = FALSE, prop.r = FALSE, dnn = c('actual default', 'predicted def
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  200
##
##
##                | predicted default
## actual default |        No |       Yes | Row Total |
## ---------------|-----------|-----------|-----------|
##             No |       124 |        16 |       140 |
##                |     0.620 |     0.080 |           |
## ---------------|-----------|-----------|-----------|
##            Yes |        32 |        28 |        60 |
##                |     0.160 |     0.140 |           |
## ---------------|-----------|-----------|-----------|
##   Column Total |       156 |        44 |       200 |
## ---------------|-----------|-----------|-----------|
##
##
```

```
F1_Score(credit_valid$default, credit_pred_boost)
```

```
## [1] 0.8378378
```

The boosting rule-based model works even better with 85.43% F1 Score.

The different types of misclassification cause different costs to the company. Now we should take the cost of default and missed opportunity into consideration. Assume that loan default costs the bank four times as much as a missed opportunity and we can rerun our decision tree model.

```
cost_mat <- matrix(c(0, 4, 1, 0), nrow = 2)
rownames(cost_mat) <- colnames(cost_mat) <- c("Yes", "No")
cost_mat
```

```
##     Yes No
## Yes   0  1
## No    4  0
```

```
tree_cost <- C5.0(x = credit_train[,-21], y = as.factor(credit_train$default), costs = cost_mat)
credit_pred_cost = predict(tree_cost,newdata = credit_valid)
CrossTable(credit_valid$default, credit_pred_cost,
           prop.chisq = FALSE, prop.c = FALSE, prop.r = FALSE, dnn = c('actual default', 'predicted def
```

```
##
##
```

```
##    Cell Contents
## |-------------------------|
## |                       N |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  200
##
##
##                | predicted default
## actual default |        No |       Yes | Row Total |
## ---------------|-----------|-----------|-----------|
##             No |        75 |        65 |       140 |
##                |     0.375 |     0.325 |           |
## ---------------|-----------|-----------|-----------|
##            Yes |        11 |        49 |        60 |
##                |     0.055 |     0.245 |           |
## ---------------|-----------|-----------|-----------|
##   Column Total |        86 |       114 |       200 |
## ---------------|-----------|-----------|-----------|
##
##
```

```r
F1_Score(credit_valid$default, credit_pred_cost)
```

```
## [1] 0.6637168
```

TWhen we add the cost matrix into the model, the average accuracy decreases while the false positive rate decreases from 0.18 to 0.07 because we penalize false positive more than false negative.

2.1 Another ML model

```r
library(caret)
# attach the iris dataset to the environment
data(iris)
# lets call our data 'df'
df_iris <- iris

validation_index <- createDataPartition(df_iris$Species, p=0.80, list=FALSE)
# select 20% of the data for validation
df_valid <- df_iris[-validation_index,]
# use the remaining 80% of data to training and testing the models
df <- df_iris[validation_index,]

# split features and labels
X <- df[,1:4]
y <- df[,5]

control <- trainControl(method="cv", number=10)
metric <- "Accuracy"

set.seed(7)
fit.ada <- train(Species~., data=df, method="treebag", metric=metric, trControl=control)

predictions <- predict(fit.ada, df_valid)
```

```r
confusionMatrix(predictions, df_valid$Species)
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction   setosa versicolor virginica
##   setosa         10          0         0
##   versicolor      0          7         2
##   virginica       0          3         8
##
## Overall Statistics
##
##                Accuracy : 0.8333
##                  95% CI : (0.6528, 0.9436)
##     No Information Rate : 0.3333
##     P-Value [Acc > NIR] : 2.444e-08
##
##                   Kappa : 0.75
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: setosa Class: versicolor Class: virginica
## Sensitivity                 1.0000            0.7000           0.8000
## Specificity                 1.0000            0.9000           0.8500
## Pos Pred Value              1.0000            0.7778           0.7273
## Neg Pred Value              1.0000            0.8571           0.8947
## Prevalence                  0.3333            0.3333           0.3333
## Detection Rate              0.3333            0.2333           0.2667
## Detection Prevalence        0.3333            0.3000           0.3667
## Balanced Accuracy           1.0000            0.8000           0.8250
```

As we can see, the accuracy of the model bagged tree is 86.67%

2.2 Hyperparameter Optimization

```r
library(randomForest)
```

```
## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:dplyr':
##
##     combine

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```r
library(mlbench)
library(caret)

# Load Dataset
```

```r
data(Sonar)
dataset <- Sonar
x <- dataset[,1:60]
y <- dataset[,61]
# summarize the balance of the target
table(dataset$Class)
```

```
##
##   M   R
## 111  97
```

The data is not very skewed. Now we can set a set of parameters before running our model.

```r
# Create model with default paramters
control <- trainControl(method="repeatedcv", number=10, repeats=1)
seed <- 7
metric <- "Accuracy"
set.seed(seed)
mtry <- sqrt(ncol(x))

# We use the expand.grid function, but we pass a scalar value
# as an argument so that our model is only trained on a
# single value for mtry = 7.746
tunegrid <- expand.grid(.mtry=mtry)
# train using the values of mtry stored above in tunegrid
rf <- train(Class~., data=dataset,
                    method="rf",
                    metric=metric,
                    tuneGrid=tunegrid,
                    trControl=control)
# And lets look at the results
print(rf)
```

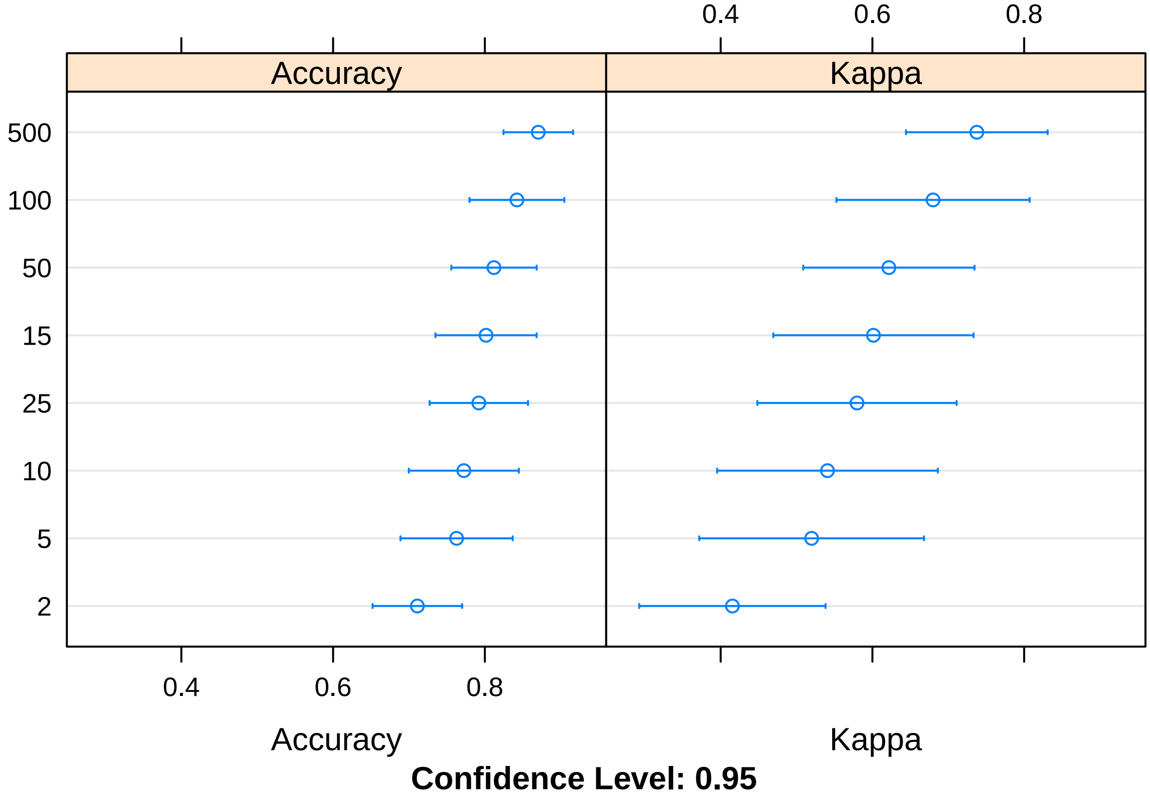
```
## Random Forest
##
## 208 samples
##  60 predictor
##   2 classes: 'M', 'R'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 1 times)
## Summary of sample sizes: 186, 187, 188, 187, 188, 187, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.8406494  0.6790832
##
## Tuning parameter 'mtry' was held constant at a value of 7.745967
```

```r
# Manual Search
control <- trainControl(method="repeatedcv", number=10, repeats=1, search="grid")

#stores trained models with different parameters
modellist <- list()

for (ntree in c(2, 5, 10, 15, 25, 50, 100, 500)) {
```

```r
  set.seed(seed)
  fit <- train(Class~., data=dataset,
               method="rf",
               metric=metric,
               trControl=control,
               ntree=ntree)
  key <- toString(ntree)
  #save the fitted model in model list by naming
  modellist[[key]] <- fit
}

# compare results
results <- resamples(modellist)
summary(results)
```

```
##
## Call:
## summary.resamples(object = results)
##
## Models: 2, 5, 10, 15, 25, 50, 100, 500
## Number of resamples: 10
##
## Accuracy
##          Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## 2   0.5714286 0.6625000 0.7071429 0.7110606 0.7889610 0.8095238    0
## 5   0.5500000 0.7261905 0.7809524 0.7627706 0.8160173 0.9047619    0
## 10  0.6000000 0.6904762 0.8047619 0.7723160 0.8452381 0.9047619    0
## 15  0.6000000 0.8000000 0.8095238 0.8016017 0.8452381 0.9523810    0
## 25  0.7000000 0.7261905 0.7809524 0.7920779 0.8095238 1.0000000    0
## 50  0.6500000 0.7646104 0.8250000 0.8120346 0.8571429 0.9047619    0
## 100 0.7142857 0.8023810 0.8095238 0.8423160 0.8944805 1.0000000    0
## 500 0.7619048 0.8214286 0.8818182 0.8704113 0.9047619 0.9523810    0
##
## Kappa
##          Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## 2   0.1209302 0.3157709 0.4197470 0.4155250 0.5711787 0.6181818    0
## 5   0.1000000 0.4455460 0.5582538 0.5199184 0.6240087 0.8073394    0
## 10  0.2079208 0.3720725 0.6075358 0.5407971 0.6859278 0.8090909    0
## 15  0.2079208 0.6039604 0.6111111 0.6013888 0.6879163 0.9041096    0
## 25  0.3939394 0.4415323 0.5622542 0.5797230 0.6146789 1.0000000    0
## 50  0.3000000 0.5248647 0.6519802 0.6216889 0.7116659 0.8090909    0
## 100 0.4166667 0.6006394 0.6164304 0.6799781 0.7861769 1.0000000    0
## 500 0.5161290 0.6364155 0.7603344 0.7376448 0.8090909 0.9041096    0
```

```r
dotplot(results)
```


Accuracy
Kappa
0.4
0.6
0.8
500
100
50
15
25
10
5
2
Confidence Level: 0.95