10장 기계학습 알고리즘1

jihyun

2019 8 28

1. 로지스틱 회기모델

아이리스 데이터로부터 로지스틱 회기 모델을 작성해 보자

```
data(iris)
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

sepal.length, sepal.width, petal.length, petal.width 로 species를 예측하는 모델을 만들어 보자.

로지스틱 회기 모델은 두 종류의 예측값만 구별이 가능하므로, virginica, versicolor 두 종류만 남긴다.

```
## 'data.frame':    100 obs. of  5 variables:
##  $ Sepal.Length: num  7 6.4 6.9 5.5 6.5 5.7 6.3 4.9 6.6 5.2 ...
##  $ Sepal.Width : num  3.2 3.2 3.1 2.3 2.8 2.8 3.3 2.4 2.9 2.7 ...
##  $ Petal.Length: num  4.7 4.5 4.9 4 4.6 4.5 4.7 3.3 4.6 3.9 ...
##  $ Petal.Width : num  1.4 1.5 1.5 1.3 1.5 1.3 1.6 1 1.3 1.4 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 2 2 2 2 2 2 2 2 2 2 ...
```

Species 의 factor종류가 3개 이므로 2개로 수정해준다.

```
d$Species <- factor(d$Species)
str(d)
```

```
## 'data.frame':    100 obs. of  5 variables:
##  $ Sepal.Length: num  7 6.4 6.9 5.5 6.5 5.7 6.3 4.9 6.6 5.2 ...
##  $ Sepal.Width : num  3.2 3.2 3.1 2.3 2.8 2.8 3.3 2.4 2.9 2.7 ...
##  $ Petal.Length: num  4.7 4.5 4.9 4 4.6 4.5 4.7 3.3 4.6 3.9 ...
##  $ Petal.Width : num  1.4 1.5 1.5 1.3 1.5 1.3 1.6 1 1.3 1.4 ...
##  $ Species     : Factor w/ 2 levels "versicolor","virginica": 1 1 1 1 1 1 1 1 1 1 ...
```

모델을 만들어 보자.

```
(m <- glm(Species ~ ., data = d, family = 'binomial'))
```

```
##
## Call:  glm(formula = Species ~ ., family = "binomial", data = d)
##
## Coefficients:
##  (Intercept)  Sepal.Length   Sepal.Width  Petal.Length   Petal.Width
##      -42.638        -2.465        -6.681         9.429        18.286
##
## Degrees of Freedom: 99 Total (i.e. Null);  95 Residual
## Null Deviance:       138.6
## Residual Deviance: 11.9  AIC: 21.9
```

모델dl 적합된 값은 fittied()를 통해 알 수 있다.

```
fitted(m)[c(1:5, 51:55)]
```

```
##           51           52           53           54           55
## 1.171672e-05 4.856237e-05 1.198626e-03 4.220049e-05 1.408470e-03
##          101          102          103          104          105
## 1.000000e+00 9.996139e-01 9.999990e-01 9.997188e-01 9.999999e-01
```

로지스틱 회기 모델은 0 또는 1로 예측을 하는데 위 결과를 보면 virginica에 해당하는 1:5행은 0, versicolor에 해당하는 51:55 행은 1로 잘 예측된 것을 알 수 있다.

train데이터 중 일부를 test데이터 처럼 사용해 보자. predict()를 통해 모델에 예측을 시킬 수 있다. type 을 response로 지정하고 예측을 수행하면 0~1 사이의 확률을 구해준다.

```
predict(m, newdata = d[c(1,10,55),], type = "response")
```

```
##           51           60          105
## 1.171672e-05 1.481064e-05 9.999999e-01
```

앞의 두개와는 근사값이 0, 뒤의 55번째는 근사값이 1로 두 종이 다른것을 잘 구별했다고 할 수 있다.

2. 다항 로지스틱 회기 분석

위의 로지스틱 회기 분석은 yes/no의 대답만 할 수있다. 하지만 이를 확장하면 다항 로지스틱 회기 분석을 할 수 있다.

iris data를 이용해 다항 로지스틱 회기 분석을 해 보자.

```
library(nnet)
(m <- multinom(Species ~ ., data=iris))
```

```
## # weights:  18 (10 variable)
## initial  value 164.791843
## iter  10 value 16.177348
## iter  20 value 7.111438
## iter  30 value 6.182999
## iter  40 value 5.984028
## iter  50 value 5.961278
## iter  60 value 5.954900
## iter  70 value 5.951851
## iter  80 value 5.950343
```

```
## iter  90 value 5.949904
## iter 100 value 5.949867
## final  value 5.949867
## stopped after 100 iterations
```

```
## Call:
## multinom(formula = Species ~ ., data = iris)
##
## Coefficients:
##            (Intercept) Sepal.Length Sepal.Width Petal.Length Petal.Width
## versicolor    18.69037    -5.458424   -8.707401     14.24477   -3.097684
## virginica    -23.83628    -7.923634  -15.370769     23.65978   15.135301
##
## Residual Deviance: 11.89973
## AIC: 31.89973
```

적성한 모델이 훈련데이터에 어떻게 적합되었는지는 fitted()를 통해 구할 수 있다.

```
head(fitted(m))
```

```
##      setosa   versicolor    virginica
## 1 1.0000000 1.526406e-09 2.716417e-36
## 2 0.9999996 3.536476e-07 2.883729e-32
## 3 1.0000000 4.443506e-08 6.103424e-34
## 4 0.9999968 3.163905e-06 7.117010e-31
## 5 1.0000000 1.102983e-09 1.289946e-36
## 6 1.0000000 3.521573e-10 1.344907e-35
```

fitted()의 결과는 각 행의 데이터가 각 분류에 속할 확률을 말한다. 위의 결과로 보면 첫 6개의 행은 전부 setosa종인 확률이 높은 것으로 보인다.

다음은 predict()로 각 종의 데이터 한 행씩을 뽑아 예측을 하게 해보자

```
  predict(m, newdata = iris[c(1,51,101),],type = 'class')
```

```
## [1] setosa     versicolor virginica
## Levels: setosa versicolor virginica
```

type에 probs를 두면 확률을 알 수 있다.

```
  predict(m, newdata = iris[c(1,51,101),],type = 'probs')
```

```
##           setosa   versicolor    virginica
## 1   1.000000e+00 1.526406e-09 2.716417e-36
## 51  2.427101e-07 9.999877e-01 1.201699e-05
## 101 9.453717e-25 2.718072e-10 1.000000e+00
```

모델의 정확도를 구해보자. 정확도는 예측값과 실제값의 차이를 통해 알 수 있다.

```
predicted <- predict(m, newdata = iris)
sum(predicted == iris$Species)/NROW(predicted)
```

```
## [1] 0.9866667
```

세부적인 표는 xtabs를 통해 볼 수있다.

```
xtabs(~predicted + iris$Species)
```

```
##             iris$Species
## predicted    setosa versicolor virginica
##   setosa         50          0         0
##   versicolor      0         49         1
##   virginica       0          1        49
```

3. 의사결정나무

의사결정나무는 지니 불순도(Gini Impurity), 또는 정보이득(Information Gain)등의 기준을 사용하여 노드를 재귀적으로 분할하면서 나무 모델을 만드는 방법이다. 의사결정나무는 if-else와 같은 조건문 형태라서 이해하기 쉽고, 속도가 빠르며, 특히 나무 모델 중 random forest는 꽤 괜찮은 성능을 보여주어 캐글에서도 많이 쓰인다고 알려져 있다.

의사결정나무를 이해하는데 도움이 되는 용어들

- 루트노드 : 최상단에 위치한 노드
- 리프노드 : 더이상 자식 노드가 없는 노드
- 불순도 : 지니 불순도를 이용하여 주로 계산한다(지니 불순도는 f(p) = p(1-p)로 정의 되며, 노드에 서로 다른 특징들이 절반씩 있을때 가장 높아지는 그래프 형태이다.). 불순도를 생각했을 때 노드를 나누는 가장 좋은 질문은 자식 노드들의 불순도가 가장 낮아지는 질문이다.

가장 일반적인 의사결정나무(rpart 패키지 이용)

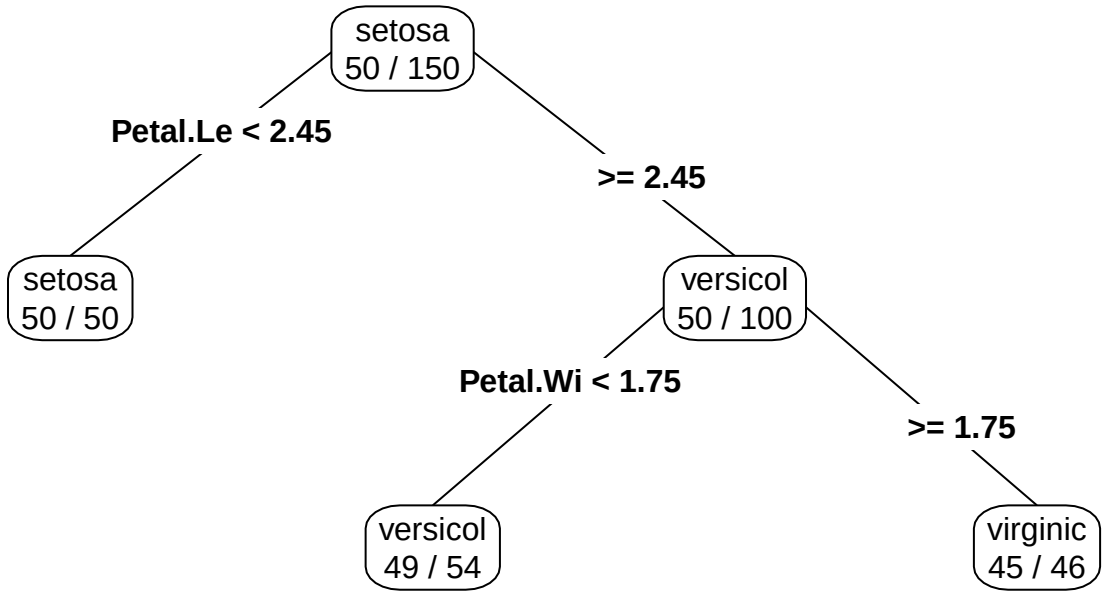
아이리스 데이터에 대해 rpart를 이용해 의사결정 나무를 작성해 보자.

```
#install.packages("rpart")
library(rpart)
(m<-rpart(Species ~ ., data = iris,))
```

```
## n= 150
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 150 100 setosa (0.33333333 0.33333333 0.33333333)
##   2) Petal.Length< 2.45 50   0 setosa (1.00000000 0.00000000 0.00000000) *
##   3) Petal.Length>=2.45 100  50 versicolor (0.00000000 0.50000000 0.50000000)
##     6) Petal.Width< 1.75 54   5 versicolor (0.00000000 0.90740741 0.09259259) *
##     7) Petal.Width>=1.75 46   1 virginica (0.00000000 0.02173913 0.97826087) *
```

위의 내용을 plot(), rpart.plot() 함수를 이용하면 좀 더 쉽게 볼 수 있다.

```
#install.packages("rpart.plot")
library(rpart.plot)
prp(m, type = 4, extra=2, digits =3)
```



rpart 또한 predict()를 통해 예측을 어떻게 했는지 볼 수 있다.

```
head(predict(m, newdata = iris, type = 'class'))
```

```
##      1      2      3      4      5      6
## setosa setosa setosa setosa setosa setosa
## Levels: setosa versicolor virginica
```

의사결정나무의 성능을 높이는 방법으로는 prunnung, rpart.control 등의 성능튜닝 방법이 있다.

조건부 추론 나무

조건부 추론 나무는 CART(rpart가 구현한 의사 결정 나무 알고리즘) 같은 의사결정나무의 두가지 문제를 해결한다. 1. 통계적유의성 없이 노드를 분할하는 데에 따른 과적합 2. 다양한 값으로 분할 가능한 변수가 다른 변수들에 비해 선호되는 문제

조건부 추로 ㄴ나무는 조건부 분포에 따라 노드 분할에 사용할 변수를 선택한다. 또한 다중가설검정을 고려한 절차를 이용해서 적당한 시기에 노드의 분활을 중지한다.

아이리스 데이터에 ctree()fmf 적용하여 species 예측 모델을 만들어 보자.

```
#install.packages('party')
library(party)
```

```
## Loading required package: grid
```

```
## Loading required package: mvtnorm
```

```
## Loading required package: modeltools
```

```
## Loading required package: stats4
```

```
## Loading required package: strucchange
```

```
## Loading required package: zoo
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

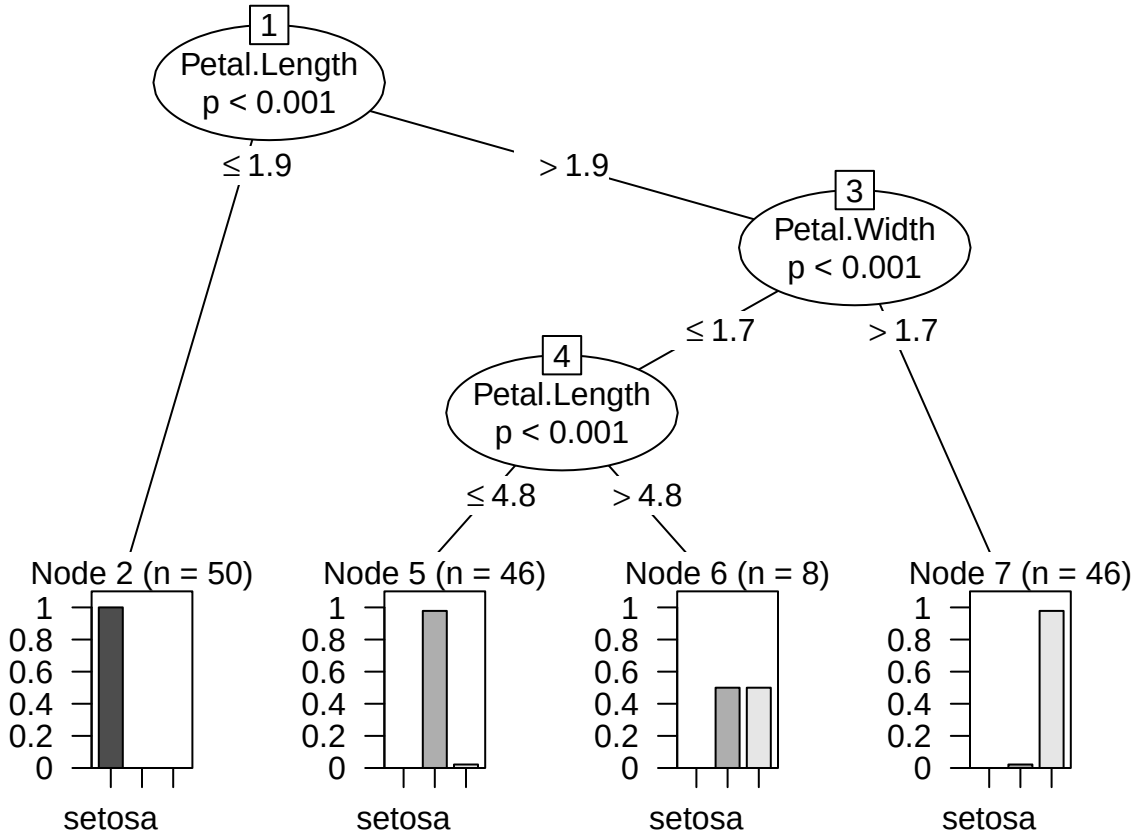
```
## Loading required package: sandwich
```

```
(m <- ctree(Species ~ ., data = iris))
```

```
##
##   Conditional inference tree with 4 terminal nodes
##
## Response:  Species
## Inputs:  Sepal.Length, Sepal.Width, Petal.Length, Petal.Width
## Number of observations:  150
##
## 1) Petal.Length <= 1.9; criterion = 1, statistic = 140.264
##   2)*  weights = 50
## 1) Petal.Length > 1.9
##   3) Petal.Width <= 1.7; criterion = 1, statistic = 67.894
##     4) Petal.Length <= 4.8; criterion = 0.999, statistic = 13.865
##       5)*  weights = 46
##     4) Petal.Length > 4.8
##       6)*  weights = 8
##   3) Petal.Width > 1.7
##     7)*  weights = 46
```

plot()을 통해 만들어진 모델을 보기쉽게 나타낼 수 있다.

```
plot(m)
```




```
# plot          setosa          levles(iris$Species)                .
```

랜덤 포레스트(random forest)

랜덤포레스트는 앙상블 학습 기법을 이용한 모델이다. 앙상블 학습은 주어진 데이터로부터 여러개의 모델을 학습한 다음, 예측 시 여러 모델의 예측 결과를 종합해서 정확도를 높히는 방법을 말한다.

랜덤포레스트는 두가지 방법을 이용해서 다양한 의사결정나무를 만든다. 1. 데이터의 일부를 복원추출로 꺼내서 해당 데이터에 대해서만 의사결정나무 만든다. 2. 일부 변수만 대상으로 하여 가지를 나눌 기준을 찾는다.

새로운 데이터에 대한 예측을 할 때는 여러 의사결정나무들이 내놓은 결과를 통해 voting하는 방식으로 최종 결정을 내린다.

랜덤포레스트는 여러개의 나무를 만드는 것으로 과적합 문제를 피한다.

아이리스 데이터에 랜덤 포레스트 모델을 만들어보자.

```
#install.packages("randomForest")
library(randomForest)
```

```
## randomForest 4.6-14
```

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

```
(m <- randomForest(Species ~ ., data = iris))
```

```
##
## Call:
##  randomForest(formula = Species ~ ., data = iris)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 4%
## Confusion matrix:
##            setosa versicolor virginica class.error
## setosa         50          0         0        0.00
## versicolor      0         47         3        0.06
## virginica       0          3        47        0.06
```

OOB : out of bag 의 줄임말으로, 모델 훈련에 사용하지 않은 데이터를 사용한 에러의 추정치이다.

예측에는 predict 함수를 사용한다.

```
head(predict(m, newdata = iris))
```

```
##      1      2      3      4      5      6
## setosa setosa setosa setosa setosa setosa
## Levels: setosa versicolor virginica
```

빠른 모델링을 위한 x y 의직접 지정.

랜덤 포레스트는 500 개의 의사결정나무를 만든다. 따라서 모델링에 걸리는 시간이 길고 데이터가 많아지면 더 오래 걸리게 된다.

포뮬러를 사용하는 방식보다, 설명변수, 종속변수를 직접 지정하면 속도를 더 빠르게 할 수있다.

```
m<- randomForest(iris[,1:4], iris[:5])
```

변수 중요도 평가

변수의 중요도를 평가 할 때에도 ramdomforest를 사용 할 수 있다. 이렇게 구한 변수 중요도는 다른 모델(선형 회기) 에 사용할 변수를 선택 하는데 사용 될 수 있다.

변수의 중요도를 알아보려면 모델 작성시 importance = TRUE 를 지정하면 된다. 그런다음 importance(), varImpPlot() 을 통해 결과를 출력한다.

```
m <- randomForest(Species ~., data = iris, importance = TRUE)
importance(m)
```

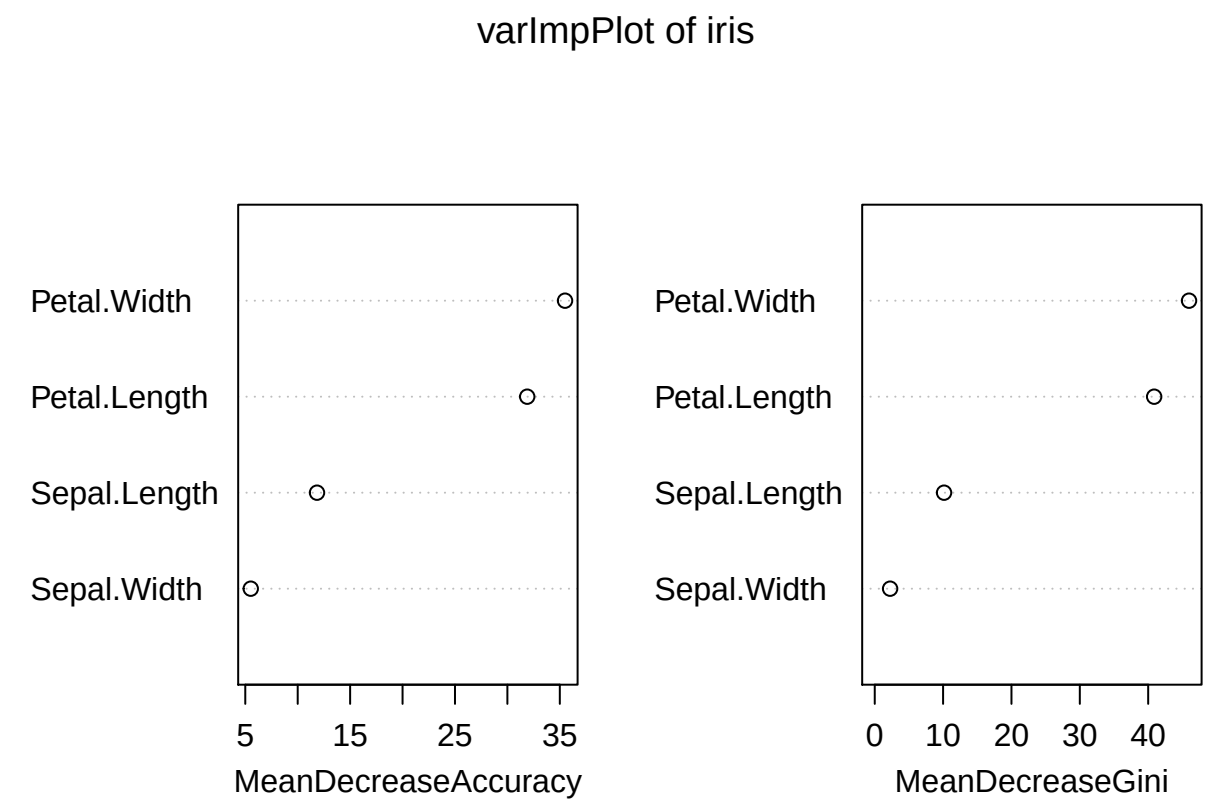
```
##                setosa versicolor virginica MeanDecreaseAccuracy
## Sepal.Length  6.235368  6.9141993  9.058515            11.840113
## Sepal.Width   4.858745  0.4553026  6.372218             5.520955
## Petal.Length 20.745513 32.9556506 26.733402            31.890614
## Petal.Width  23.739460 33.3353232 32.384423            35.496691
##              MeanDecreaseGini
## Sepal.Length        10.140773
## Sepal.Width          2.247122
## Petal.Length        40.875878
## Petal.Width         45.976574
```

정확도 에서는 (Mean Decrease Accuracy) petal.length > petal.width > sepal.length > sepal.width 순으로 중요한 것을 알 수 있다.

노트 불순도 계선 (Mean Decrease Gini) petal.width > petal.length > sepal.width > sepal.length 순으로 중요한 변수라고 볼 수있다.

varImpPlot() 을 이용하면 좀더 쉽게 볼 수 있다.

```
varImpPlot(m, main = "varImpPlot of iris")
```



파라미터 튜닝

ramdomForest()에는 나무 개수(ntree), 자식 노드로 나눌 때 고려할 변수의 개수(mtry) 등의 파라미터가 있다 기본값으로도 잘 되지만, 성능을 높히기 위해서는 교차 검증을 사용하여 성능을 개선 할 수 있다.

expand.grid()를 이용하여 파라미터의 조합을 만든 후 성능을 차례로 볼 수 있다.

예시로 ntree 를 10, 100, 200 으로, mtry를 3,4 로 바꿔가며 조합해보자

```
(grid <- expand.grid(ntree = c(10, 100, 200), mtry = c(3,4)))
```

```
##   ntree mtry
## 1    10    3
## 2   100    3
## 3   200    3
## 4    10    4
```

```
## 5   100    4
## 6   200    4
```

파라미터 조합을 10개로 분할한 데이터에 적용하여 모델의 성능을 평가하는 일을 3회 반복하여 최선의 파라미터를 찾아보자

```
#install.packages('cvTools')
#install.packages('foreach')
library(cvTools)
```

```
## Loading required package: lattice
```

```
## Loading required package: robustbase
```

```
library(foreach)
library(randomForest)
set.seed(719)

K = 10
R = 3
cv <- cvFolds(NROW(iris), K=K, R=R)
grid <-  expand.grid(ntree = c(10, 100, 200), mtry = c(3,4))

result <- foreach(g = 1:NROW(grid), .combine = rbind) %do% {
  foreach(r = 1:R , .combine = rbind) %do%{
    foreach( k =1:K, .combine=rbind) %do%{
      validation_idx <- cv$subsets[which(cv$which == k), r]
      train <- iris[-validation_idx,]
      validation <- iris[validation_idx,]

      #
      m <- randomForest(Species ~.,
                        data = train,
                        ntree = grid[g, "ntree"],
                        mtry = grid[g, "mtry"])
      #
      predicted <- predict(m, newdata = validation)

      #
      precision <- sum(predicted == validation$Species) / NROW(predicted)
      return(data.frame(g=g, precision = precision))

    }
  }
}

str(result)
```

```
## 'data.frame':    180 obs. of  2 variables:
##  $ g        : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ precision: num  1 0.867 0.933 1 1 ...
```

```
library(plyr)
```

```
##
## Attaching package: 'plyr'
```

```
## The following object is masked from 'package:modeltools':
##
##     empty
```

```
ddply(result, .(g), summarise, mean_precision = mean(precision))
```

```
##   g mean_precision
## 1 1      0.9466667
## 2 2      0.9511111
## 3 3      0.9511111
## 4 4      0.9466667
## 5 5      0.9577778
## 6 6      0.9488889
```

g 값 마다 묶어서 평균을 구해보자

가장 높은 성능은 g=4, g=6일 때다. 이 조합을 grid에서 찾아보면 ntree = 10, mtry = 4인 경우와, ntree = 200, mtry = 4인 경우이다.

```
grid[c(4,6),]
```

```
##   ntree mtry
## 4    10    4
## 6   200    4
```

정규화 랜덤 포레스트(RRF, Regulized Random Forest)

정규화 랜덤 포레스트는 랜덤포레스트의 변수 선택 방식을 개선한 방식으로, 부모 노드에서 가지를 나눌 때 사용한 변수와 유사한 변수로 자식 노드에서 가지를 나눌 경우, 해당 변수에 패널티를 부여한다.

이 방식은은 랜덤 포레스트에 비해 좋은 변수를 선택하는 것으로 알려져 있으며 RRF 패키지를 통해 이용 할 수 있다.