2021 머신러닝 여름 스터디

Parameter tuning

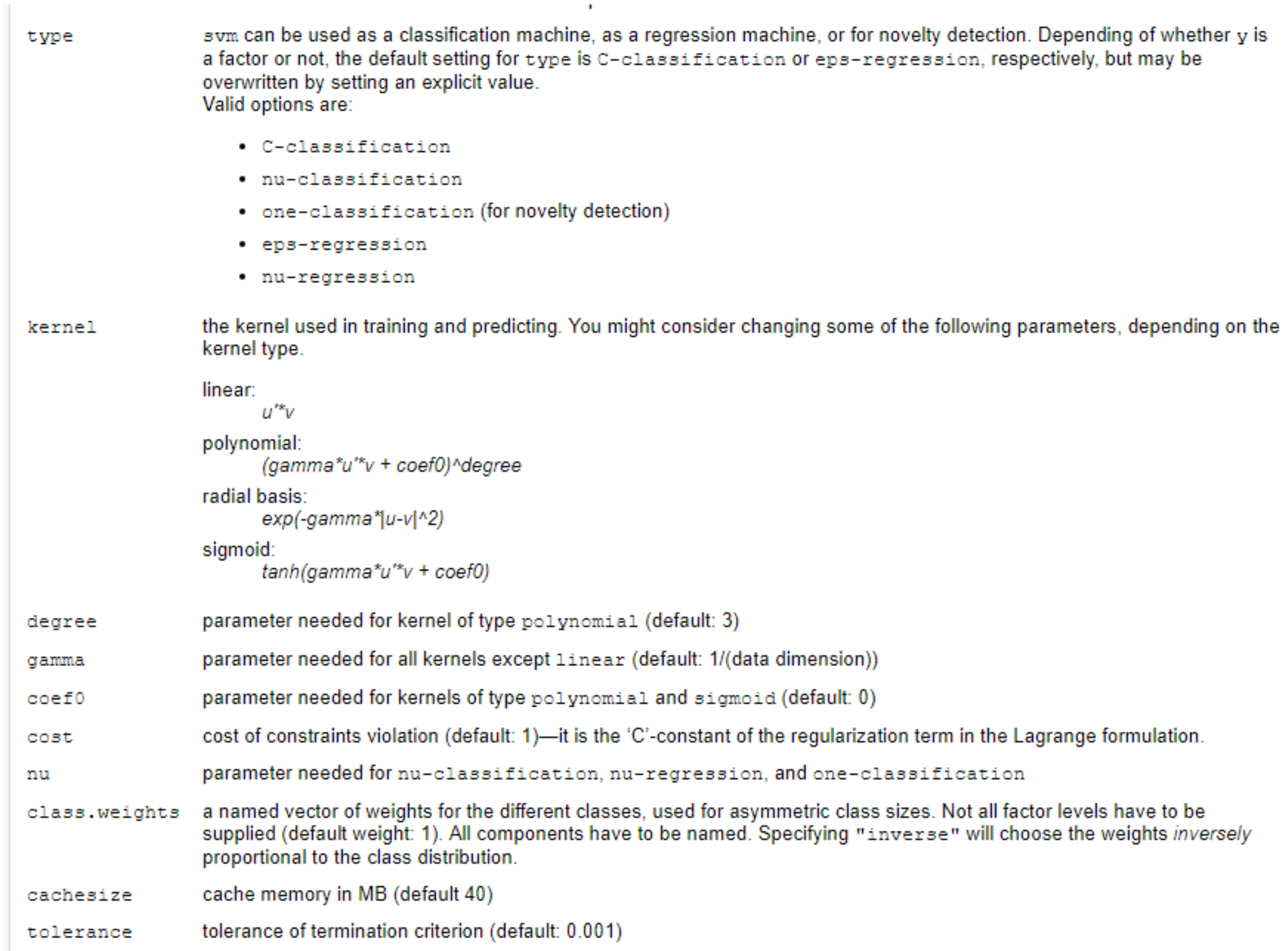
Sejong Oh
Dankook University

요약

- 모든 예측 모델 알고리즘은 모델의 성능에 영향을 미치는 parameter 들이 있음
- 조율 모수, hyper parameter 라는 용어로 주로 사용
- Parameter 의 값을 어떻게 설정하느냐에 따라 모델의 성능이 달라짐
- 모델이 최고의 성능을 내는 parameter 값들의 조합을 찾아내는 것이 과제임
- 튜닝 방법
  - 직접 구현
  - 알고리즘에서 제공하는 튜닝 함수를 이용 (예: svm)
  - Caret 패키지 이용

Parametr 의 예

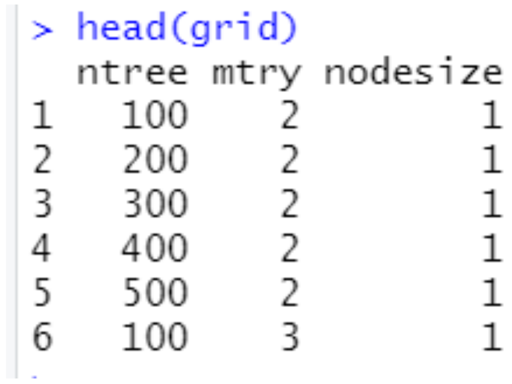
- svm 모델



`type`	`svm` can be used as a classification machine, as a regression machine, or for novelty detection. Depending of whether `y` is a factor or not, the default setting for `type` is `C-classification` or `eps-regression`, respectively, but may be overwritten by setting an explicit value. Valid options are: • `C-classification` • `nu-classification` • `one-classification` (for novelty detection) • `eps-regression` • `nu-regression`
`kernel`	the kernel used in training and predicting. You might consider changing some of the following parameters, depending on the kernel type. linear: *u'*v* polynomial: *(gamma*u'*v + coef0)^degree* radial basis: *exp(-gamma*|u-v|^2)* sigmoid: *tanh(gamma*u'*v + coef0)*
`degree`	parameter needed for kernel of type `polynomial` (default: 3)
`gamma`	parameter needed for all kernels except `linear` (default: 1/(data dimension))
`coef0`	parameter needed for kernels of type `polynomial` and `sigmoid` (default: 0)
`cost`	cost of constraints violation (default: 1)—it is the 'C'-constant of the regularization term in the Lagrange formulation.
`nu`	parameter needed for `nu-classification`, `nu-regression`, and `one-classification`
`class.weights`	a named vector of weights for the different classes, used for asymmetric class sizes. Not all factor levels have to be supplied (default weight: 1). All components have to be named. Specifying `"inverse"` will choose the weights *inversely* proportional to the class distribution.
`cachesize`	cache memory in MB (default 40)
`tolerance`	tolerance of termination criterion (default: 0.001)

직접구현 한 tuening

```
library(cvTools)
library(randomForest)

data("PimaIndiansDiabetes2", package = "mlbench")
pima.data <- na.omit(PimaIndiansDiabetes2)

# define grid
ntree <- c(100,200,300,400, 500)   # number of decision trees
mtry <- c(2,3,4,5)                 # number of features for split
nodesize <- c(1,2,3,4,5)           # min sample for terminal node

grid <- expand.grid(ntree, mtry, nodesize)
names(grid) <- c("ntree", "mtry", "nodesize")
```



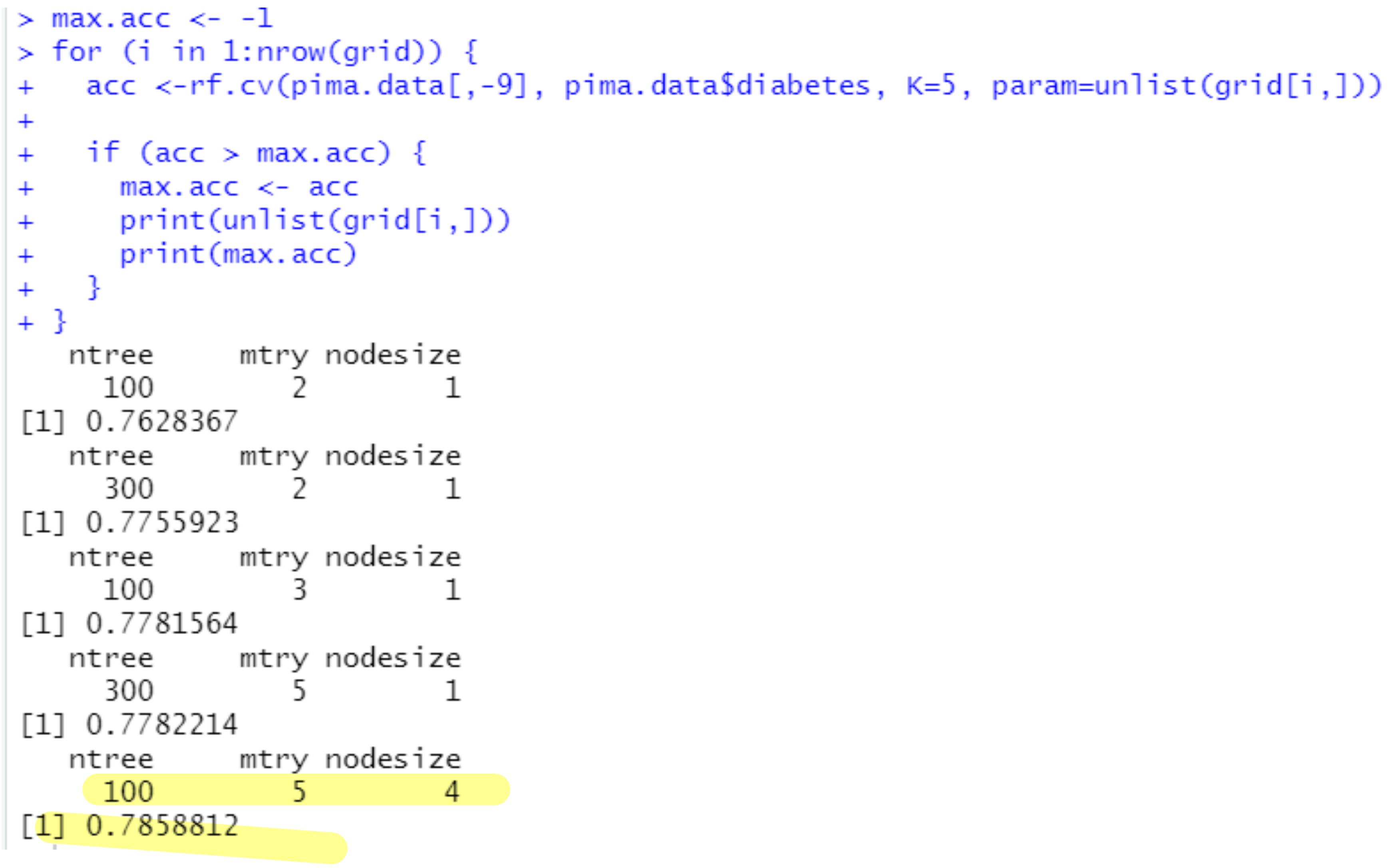
```
> head(grid)
  ntree mtry nodesize
1   100    2        1
2   200    2        1
3   300    2        1
4   400    2        1
5   500    2        1
6   100    3        1
```

```
rf.cv <- function (ds, cl, K=10, param) {
  set.seed(100)
  folds <- cvFolds(nrow(ds), K)
  acc <- c()
  for (i in 1:K) {
    tr.idx <- which(folds$which==i)
    ds.tr <- ds[-tr.idx,]
    ds.ts <- ds[tr.idx,]
    cl.tr <- cl[-tr.idx]
    cl.ts <- cl[tr.idx]
    model <- randomForest(ds.tr, cl.tr,
                          ntree=param[1],
                          mtry=param[2],
                          nodesize=param[3])

    pred <- predict(model, ds.ts)
    acc[i] <- mean(pred==cl.ts)
  }
  return(mean(acc))
}
```

3개의 col

```
# tuning
max.acc <- -1
for (i in 1:nrow(grid)) {
  acc <-rf.cv(pima.data[,-9], pima.data$diabetes, K=5,
              param=unlist(grid[i,]))
                                  i번째 행
  if (acc > max.acc) {
    max.acc <- acc
    print(unlist(grid[i,]))
    print(max.acc)
  }
}
```

unlist(grid[i,]): grid[i,]의 자료구조가 data frame 인데
이것을 vector로 변환



```
> max.acc <- -1
> for (i in 1:nrow(grid)) {
+   acc <-rf.cv(pima.data[,-9], pima.data$diabetes, K=5, param=unlist(grid[i,]))
+
+   if (acc > max.acc) {
+     max.acc <- acc
+     print(unlist(grid[i,]))
+     print(max.acc)
+   }
+ }
   ntree     mtry nodesize
     100        2        1
[1] 0.7628367
   ntree     mtry nodesize
     300        2        1
[1] 0.7755923
   ntree     mtry nodesize
     100        3        1
[1] 0.7781564
   ntree     mtry nodesize
     300        5        1
[1] 0.7782214
   ntree     mtry nodesize
     100        5        4
[1] 0.7858812
```

caret 을 이용한 tuning

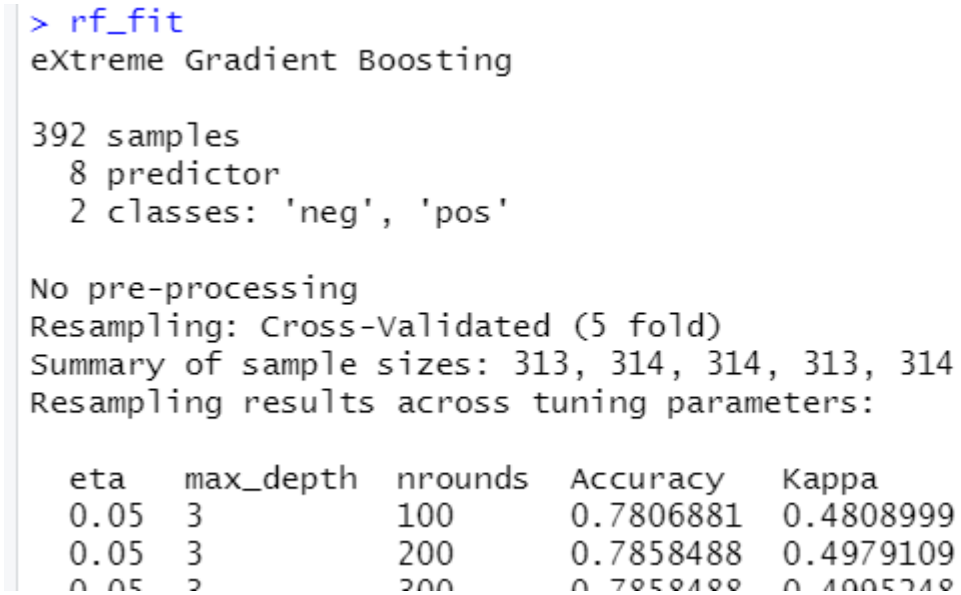
eXtreme Gradient Boosting	xgbDART	Classification, Regression	xgboost, plyr	nrounds, max_depth, eta, gamma, subsample, colsample_bytree, rate_drop, skip_drop, min_child_weight
eXtreme Gradient Boosting	xgbLinear	Classification, Regression	xgboost	nrounds, lambda, alpha, eta
eXtreme Gradient Boosting	xgbTree	Classification, Regression	xgboost, plyr	nrounds, max_depth, eta, gamma, colsample_bytree, min_child_weight, subsample

caret 을 이용한 tuning

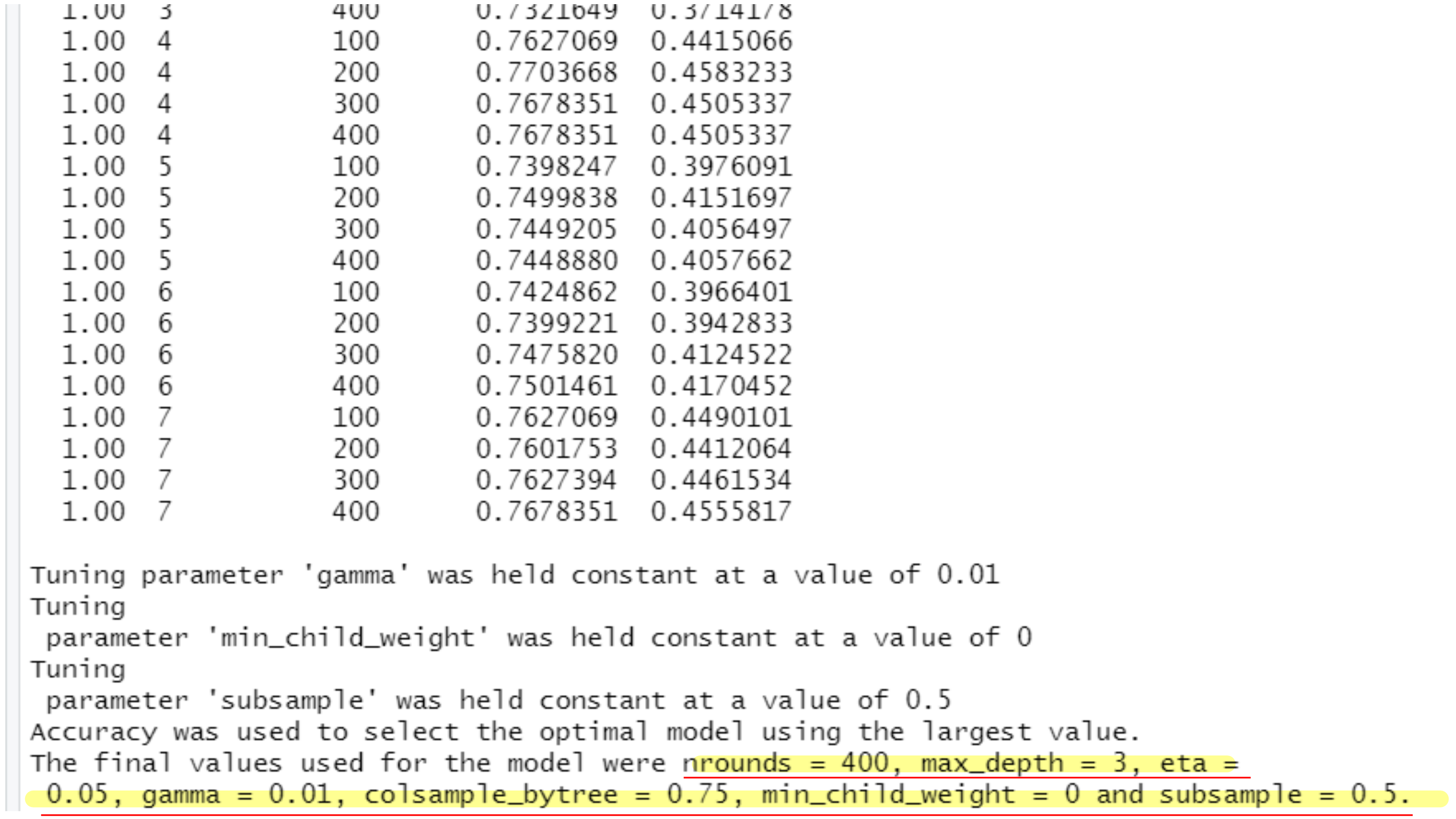
```
library(xgboost)
library(caret)

data("PimaIndiansDiabetes2", package = "mlbench")
pima <- na.omit(PimaIndiansDiabetes2)

# convert factor class to number
pima$diabetes <- as.integer(pima$diabetes)-1

trctrl <- trainControl(method = "cv", number = 5)

tune_grid <- expand.grid(nrounds=c(100,200,300,400),
                        max_depth = c(3:7),
                        eta = c(0.05, 1),
                        gamma = c(0.01),
                        colsample_bytree = c(0.75),
                        subsample = c(0.50),
                        min_child_weight = c(0))
```

caret

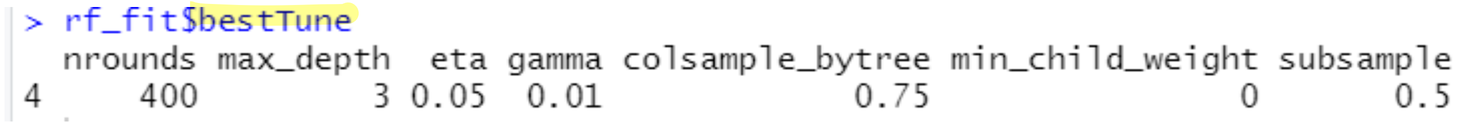
```
rf_fit <- train(diabetes ~., data = pima, method = "xgbTree",
                trControl=trctrl,
                tuneGrid = tune_grid,
                tuneLength = 10)


rf_fit             # tuning result
rf_fit$bestTune    # tuned parameter
```



```
> rf_fit
eXtreme Gradient Boosting

392 samples
  8 predictor
  2 classes: 'neg', 'pos'

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 313, 314, 314, 313, 314
Resampling results across tuning parameters:

  eta   max_depth  nrounds  Accuracy   Kappa
  0.05  3          100      0.7806881  0.4808999
  0.05  3          200      0.7858488  0.4979109
  0.05  3          300      0.7858488  0.4995248
```



```
  1.00  3         400     0.7321649  0.3714178
  1.00  4         100     0.7627069  0.4415066
  1.00  4         200     0.7703668  0.4583233
  1.00  4         300     0.7678351  0.4505337
  1.00  4         400     0.7678351  0.4505337
  1.00  5         100     0.7398247  0.3976091
  1.00  5         200     0.7499838  0.4151697
  1.00  5         300     0.7449205  0.4056497
  1.00  5         400     0.7448880  0.4057662
  1.00  6         100     0.7424862  0.3966401
  1.00  6         200     0.7399221  0.3942833
  1.00  6         300     0.7475820  0.4124522
  1.00  6         400     0.7501461  0.4170452
  1.00  7         100     0.7627069  0.4490101
  1.00  7         200     0.7601753  0.4412064
  1.00  7         300     0.7627394  0.4461534
  1.00  7         400     0.7678351  0.4555817

Tuning parameter 'gamma' was held constant at a value of 0.01
Tuning
 parameter 'min_child_weight' was held constant at a value of 0
Tuning
 parameter 'subsample' was held constant at a value of 0.5
Accuracy was used to select the optimal model using the largest value.
The final values used for the model were nrounds = 400, max_depth = 3, eta =
 0.05, gamma = 0.01, colsample_bytree = 0.75, min_child_weight = 0 and subsample = 0.5.
```



```
> rf_fit$bestTune
  nrounds max_depth  eta gamma colsample_bytree min_child_weight subsample
4     400         3 0.05  0.01             0.75                0       0.5
```

[과제]

- Xgboost 에 대해 자신만의 parameter tuning 함수를 만들어 테스트 하시오
  - 데이터셋 :

```
data("PimaIndiansDiabetes2", package = "mlbench")
pima <- na.omit(PimaIndiansDiabetes2)
```