강의록 코드 구현 불가능 정리

1. R의 predict 함수의 method : 파라미터 추정이 다루어지는 방식을 결정한다. 정하지 않으면 plug-in 방식 사용.
    a. method = "predictive": an unbiased estimator of the log posterior probabilities is used
    b. method = "debiased": the parameter estimates are integrated out using a vague prior.(=non-informative prior, flat prior)

R

```
predplot(cush.lda, "LDA")
predplot(cush.qda, "QDA")
predplot(cush.qda, "QDA (predictive)", method = "predictive")
predplot(cush.qda, "QDA (debiased)", method = "debiased")
```

R

---

2. LDA 함수의 method="t"를 구현 못 함.
    a. method=”t” : robust estimates of mean and variance matrix based on a t distribution.

R

```
fgl.ld <- predict(lda(type ~ ., fgl), dimen = 2)$x

fgl.rld <- predict(lda(type ~ ., fgl, method = "t"), dimen = 2)$x
```

R

---

4. tr$cptable 구현을 못 함.
    a. cp: If the cost of adding another variable to the decision tree from the current node is above the value of cp, then tree building does not continue.

R

```
library(rpart)
res.rpart <- CVtest(
  function(x, ...) {
```

```
    tr <- rpart(type ~ ., fgl[x,], ...)
    cp <- tr$cptable
    r <- cp[, 4] + cp[, 5]
    rmin <- min(seq(along = r)[cp[, 4] < min(r)])
    cp0 <- cp[rmin, 1]
    cat("size chosen was", cp[rmin, 2] + 1, "\n")
    prune(tr, cp = 1.01*cp0)
  },
  function(obj, x)
    predict(obj, fgl[x, ], type = "class"),
  cp = 0.001
)

cptable: a matrix of information on the optimal prunings based on a complexity parameter.
```

R

---

5. olvq1 함수가 파이썬에는 없음.
    a. ovlq1: **Optimized Learning Vector Quantization. The method improve the result of Learning Vector Quantization (LVQ) by optimizing the weight vectors.**

R

```
CV.lvq <- function()
{
    res <- fgl$type
    for(i in sort(unique(rand))) {
        cat("doing fold", i, "\n")
        cd0 <- lvqinit(fgl0[rand != i,], fgl$type[rand != i],
                       prior = rep(1, 6)/6, k = 3)
        cd1 <- olvq1(fgl0[rand != i,], fgl$type[rand != i], cd0)
        cd1 <- lvq3(fgl0[rand != i,], fgl$type[rand != i],
                    cd1, niter = 10000)
        res[rand == i] <- lvqtest(cd1, fgl0[rand == i, ])
    }
    res
}
```

7. CVnn 구현 불가.
    a. CVnn: 교차 검증된 오류율을 기반으로 신경망의 hidden layer를 선택해서 neural network를 돌려준다. 오류율 계산시에 res function을 사용. 파이썬에는 res fucntion이 없음.
    b. res function: Converts correlation(r) to mean difference(d), unbised estimate of mean difference(g), Fisher’s z vlaue, log odds ratio, variance of r,d,g.

```
CVnn2 <- function(formula, data,
                  size = rep(6,2), lambda = c(0.001, 0.01),
                  nreps = 1, nifold = 5, verbose = 99, ...)
{
    CVnn1 <- function(formula, data, nreps=1, ri, verbose,  ...)
    {
        truth <- data[,deparse(formula[[2]])]
        res <-  matrix(0, nrow(data), length(levels(truth)))
        if(verbose > 20) cat("  inner fold")
        for (i in sort(unique(ri))) {
            if(verbose > 20) cat(" ", i,  sep="")
            for(rep in 1:nreps) {
                learn <- nnet(formula, data[ri !=i,], trace = FALSE, ...)
                res[ri == i,] <- res[ri == i,] +
                    predict(learn, data[ri == i,])
            }
        }
        if(verbose > 20) cat("\n")
        sum(as.numeric(truth) != max.col(res/nreps))
    }
    truth <- data[,deparse(formula[[2]])]
    res <-  matrix(0, nrow(data), length(levels(truth)))
    choice <- numeric(length(lambda))
    for (i in sort(unique(rand))) {
        if(verbose > 0) cat("fold ", i,"\n", sep="")
        ri <- sample(nifold, sum(rand!=i), replace=TRUE)
        for(j in seq(along=lambda)) {
            if(verbose > 10)
                cat("  size =", size[j], "decay =", lambda[j], "\n")
            choice[j] <- CVnn1(formula, data[rand != i,], nreps=nreps,
                               ri=ri, size=size[j], decay=lambda[j],
                               verbose=verbose, ...)
        }
        decay <- lambda[which.is.max(-choice)]
        csize <- size[which.is.max(-choice)]
        if(verbose > 5) cat("  #errors:", choice, "  ") #
        if(verbose > 1) cat("chosen size = ", csize,
                            " decay = ", decay, "\n", sep="")
        for(rep in 1:nreps) {
            learn <- nnet(formula, data[rand != i,], trace=FALSE,
                          size=csize, decay=decay, ...)
            res[rand == i,] <- res[rand == i,] +
                predict(learn, data[rand == i,])
        }
    }
    factor(levels(truth)[max.col(res/nreps)], levels = levels(truth))
}

if(FALSE) { # only run this if you have time to wait
res.nn2 <- CVnn2(type ~ ., fgl1, skip = TRUE, maxit = 500, nreps = 10)
con(true = fgl$type, predicted = res.nn2)
}
```