```r
### init.R

library(MASS)
mush0 = read.csv("mydata.csv")
mush = mush0[,-17]


### util.R

cmp = function(data, cIndex) {
  print(paste("colname:", names(data)[cIndex], " [edible,poisonous]"))
  print(summary(data[which(data$class=="e"),cIndex]))
  print(summary(data[which(data$class=="p"),cIndex]))
}


### myclass.R

#myclass = rep("?", nrow(mush))

#for (i in 1:nrow(mush))
#{
# if (mush[i,]$odor %in% c("a","l") | mush[i,]$spore.print.color %in% c("b",
# {
#   myclass[i] = "e"
# }
# else if (mush[i,]$odor %in% c("c","f","m","p","s","y") | mush[i,]$spore.pr
# {
#   myclass[i] = "p"
# }
#}

myclass = ifelse(mush$odor %in% c("a","l") | mush$spore.print.color %in% c("
  ifelse(mush$odor %in% c("c","f","m","p","s","y") | mush$spore.print.color
    ifelse(mush$spore.print.color != "w", "e",
      ifelse(mush$gill.size=="b", "e",
        ifelse(mush$gill.spacing=="c" | mush$stalk.surface.above.ring=="k",
          ifelse(mush$population=="c", "p",
"?"
          )
        )
      )
    )
  )
)
```

```
46 myclass = factor(myclass)
47 foo = mush[which(myclass=="?"),]
48
49 print("------------------------------------------------------------")
50 for (i in (1:ncol(mush))[-c(1,17)])
51 {
52   cmp(foo,i)
53 }
54
55 print(table(myclass, mush$class))
56
57 ### br.R
58
59 start = Sys.time()
60 print(start)
61
62 # Constants
63 nCol = ncol(mush)
64 index = 1:nrow(mush)
65 k = 5
66
67 # Feature value summary table setup.
68 setsize = 1:nCol
69 for (i in 1:nCol)
70 {
71   setsize[i] = length(unique(mush[,i]))
72 }
73 maxFcnt = max(setsize)
74
75 # Results structures.
76 preds = data.frame(mush$class,mush$class,mush$class,mush$class,mush$class,re
77 names(preds) = c("real", "nbc", "lda", "qda", "log.reg", "fold")
78
79 # Set of test features. (not including class)
80 testFeatureSet = 2:nCol
81
82 for (fold in 0:(k-1))
83 {
84   # Partition Data for NBC
85   test = mush[which(index %% k == fold),]
86   train = mush[which(index %% k != fold),]
87   traine = train[which(train$class == "e"),]
88   trainp = train[which(train$class == "p"),]
89
90   # Initialize results structures.
```

```r
  test.preds = preds[which(index %% k == fold),]
  test.preds$fold = fold
  test.preds$real = test$class

  pe = log(nrow(traine)) # P(edible)
  pp = log(nrow(trainp)) # P(poisonous)

  # Create concise matrix of log(P(feature value|class)) estimates
  pfe = matrix(nrow=nCol, ncol=maxFcnt)
  pfp = matrix(nrow=nCol, ncol=maxFcnt)
  eFeatValLogit = matrix(nrow=nCol, ncol=maxFcnt)
  for (c in 1:nCol)
  {
    l = levels(mush[1,c])
    for (v in 1:length(l))
    {
      valueTotal = max(log(sum(train[,c] == l[v])), 0) * 100
      pfe[c,v] = max(log(sum(traine[,c] == l[v])) , -valueTotal)
      pfp[c,v] = max(log(sum(trainp[,c] == l[v])) , -valueTotal)
    }
  }
  # Logit(ish) Estimate for numerical methods
  eFeatValLogit = pfe - pfp # Matrix subtraction

  # Create probabalistic data table.
  mush.eprob = data.frame(mush[,1],rep(list(rep(-1,nrow(mush))),nCol))
  names(mush.eprob) = names(mush)
  for (c in 2:nCol)
  {
    # Fancy vector method for setting a column of values at a time.
    mush.eprob[,c] = eFeatValLogit[(as.integer(mush[,c])-1) * nCol + c]
  }
  test.eprob = mush.eprob[which(index %% k == fold),]
  train.eprob = mush.eprob[which(index %% k != fold),]

  # NBC
  res = rep("?", nrow(test))
  for (ti in 1:nrow(test))
  {
    pi.e = pe + sum(pfe[(as.integer(test[ti,testFeatureSet])-1)*nCol + testF
    pi.p = pp + sum(pfp[(as.integer(test[ti,testFeatureSet])-1)*nCol + testF

    res[ti] = ifelse(pi.e > pi.p, "e", "p")
  }
  test.preds$nbc = factor(res, levels=levels(mush[1,1]))
```

```
136
137   f = as.formula(paste("class~",paste(names(mush)[testFeatureSet], collapse=
138   # LDA
139   lda.fit = lda(f, data=train.eprob)
140   test.preds$lda = predict(lda.fit, test.eprob)$class
141
142   # QDA
143   qda.fit = qda(f, data=train.eprob)
144   test.preds$qda = predict(qda.fit, test.eprob)$class
145
146   # Logistic Regression
147   glm.fit = glm(f, family=binomial, data=train.eprob)
148   glm.prob = predict(glm.fit, test.eprob, type="response")
149   test.preds$log.reg = factor(ifelse(glm.prob < .5, "e", "p"), levels=levels
150
151   preds[which(index %% k == fold),] = test.preds
152
153 }
154 print(table(preds$nbc, preds$real))
155 print(table(preds$lda, preds$real))
156 print(table(preds$qda, preds$real))
157 print(table(preds$log.reg, preds$real))
158 print(Sys.time() - start)
159
160
161 ### nbc.R
162
163 mylog = log
164
165 index = 1:nrow(mush)
166 k = 5
167 caution = .5
168 cautionOff = 0
169 cautionRange = .01
170 setsize = 1:ncol(mush)
171 for (i in 1:ncol(mush))
172 {
173   setsize[i] = length(unique(mush[,i]))
174 }
175 maxFcnt = max(setsize)
176
177 pred = factor(c("e", "p"))
178 real = factor(c("e", "p"))
179 confuse = table(pred, real)
180 confuse[,] = 0
```

```
181
182 coef = matrix(nrow=nrow(mush), ncol=2)
183 pprob = rep(0, nrow(mush))
184 eprob = pprob
185 realClass = rep('?', nrow(mush))
186 coefi = 0
187
188 for (fold in 0:(k-1))
189 {
190   test = mush[which(index %% k == fold),]
191   train = mush[which(index %% k != fold),]
192   traine = train[which(train$class == "e"),]
193   trainp = train[which(train$class == "p"),]
194
195   pe = mylog(nrow(traine))# * 2
196   pp = mylog(nrow(trainp))# * 2
197
198   pfe = matrix(nrow=ncol(mush), ncol=maxFcnt)
199   pfp = pfe
200   for (c in 1:ncol(mush))
201   {
202     l = levels(mush[1,c])
203     for (v in 1:length(l))
204     {
205       emptyWeight = 0#-max(mylog(sum(train[,c] == l[v])), 0)
206       pfe[c,v] = max(mylog(sum(traine[,c] == l[v])), emptyWeight)
207       pfp[c,v] = max(mylog(sum(trainp[,c] == l[v])), emptyWeight)
208     }
209   }
210
211   res = rep("?", nrow(test))
212   for (ti in 1:nrow(test))
213   {
214     pi.e = pe
215     pi.p = pp
216     #print(paste(pi.e, pi.p))
217     #for (fi in 2:ncol(mush))
218     for (fi in c(6,21))
219     {
220       pi.e = pi.e + pfe[fi,as.integer(test[ti,fi])]
221       pi.p = pi.p + pfp[fi,as.integer(test[ti,fi])]
222     }
223
224     res[ti] = #ifelse(
225       #abs(pi.e - pi.p) < cautionRange
```

```
      #abs(.5 - pi.e / (pi.e + pi.p)) < cautionRange
      #, "?",
      ifelse(pi.e * (1-caution) > pi.p * caution, "e", "p")
    #)

    #coef[coefi,] = c(pi.e / (pi.e + pi.p), ifelse(res[ti] == test[ti,1], 1,
    pprob[coefi] = pi.e
    eprob[coefi] = pi.p
    realClass[coefi] = test[ti,1]
    coefi = coefi + 1
  }
  resf = factor(res)#, levels=c(1,2,3), labels=c("e","p","?"))

  curTab = table(res, test$class)
  print(curTab)
  confuse = confuse + curTab
}
print(confuse)
```