06. Classification Methods

KIM SANG HYUN(202211545)

2025-06-13

```
#if (!requireNamespace("BiocManager", quietly = TRUE))
#install.packages("BiocManager")
#BiocManager::install("multtest")
#BiocManager::install("ALL")
#BiocManager::install("hgu95av2.db")
#install.packages("ROCR")
#install.packages("rpart")
#install.packages("rpart.plot")
#install.packages("randomForest")
#install.packages("e1071")
```

```
library(multtest)
library(ALL)
library(hgu95av2.db)
library(ROCR)
library(rpart)
library(rpart.plot)
library(randomForest)
library(e1071)
```

Introduction

- In, bioinformatics, an importanr question is whether or not the diagnosis of a patient can be predicted by gene expression.
- A related question is which of the thousands of genes play on important role in the prediction of class membership.
- These important genes are often called biomarkers.
- To evaluate the quality of any predictions, the fundamental concepts of sensitivity and specificity are frequently used.
- The prediction performance can be summarized in a single number by the AUC (ROC curve).

Classification of micro RNA

- Based on gene expression measurements, we need to classify microRNA vs non-microRNA.
- 지금 여기에선 permutation p-values를 classifer로 이용한다.

일단 내가 이해한 바로는 microRNA 3424에서 rkr microRNA를 1000번 섞어서 순열 pvalue 계산을 진행. 그럼 총 3424개의 p value가 나오고 이걸 non microRNA도 동일하게 해서 총 6848개의 p value를 구해서 0.01이라는 classifier로 분류 진행. -> 근데 교수님이 일단은 그냥 결과 해석에 초점을 맞추라고 하심.

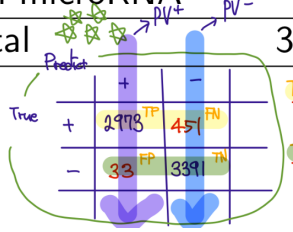
- This procedure yielded a total of 3,424 p-values.
- The same procedure is conducted for non-microRNA molecules which were chosen to have similar length and nucleotide percentages as the microRNAs.
- The number of sequences with p-values below the threshold value of 0.01 is given in Table below:

이거에 따른 다양한 분류기 생성 ⇒ confusion matrix 여러 생성.

⇒ 과연의 classifier가 좋은 걸까?

classifier

	test positive ($p \leq 0.01$)	test negative ($p > 0.01$)	total
(+)microRNA	2,973	451	3,424
(-)non-microRNA	33	3,391	3,424
Total	3,006	3,842	6,848



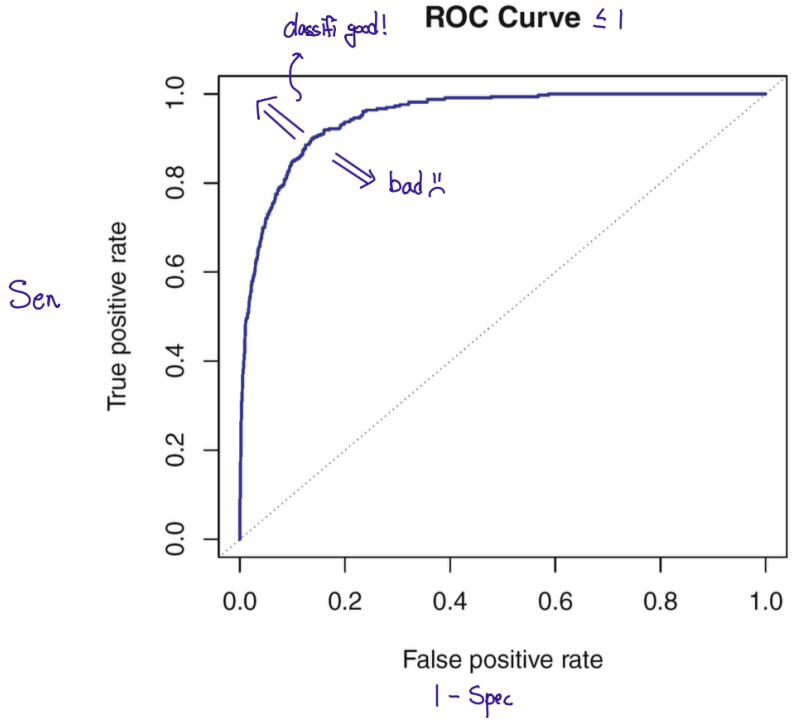
$TPR = \frac{TP}{TP+FN}$ = "Sensitivity" = P(predict "+" | True "+")

$TNR = \frac{TN}{FP+TN}$ = "Specificity" = P(predict "−" | True "−")

8 / 67

Assessment of the Performance of Classifiers

- The ROC curve is a popular graphic for sum displaying the two types of errors for all possible thresholds.
- The overall performance of a classifier, summarized over all possible thresholds, is given by AUC.



- The ROC plot displays both true positive rate and false positive rate simultaneously.

ROC curve

- We have observed that the expression values of the gene CCND3 tend to be greater for ALL patients.
- We may us CCND3 expression data as a test for predicting ALL using a certain CCND3 cutoff value for classification.
- For gene expression values larger than a certain cutoff we declare the the positive in the sense of indicating ALL.
- Suppose that x_i is a gene expression value of the i-th individual and t is some threshold.
- The true and fales positives can be computed for each possible cutoff value - which is how a ROC curve is calculated.

  아!!!!

  AUC 값는 하나의 cutoff에 대한 예측 성능을 평가 하는 것이 아니다!!

  특정 gene(CCND3)이 ALL, AML을 잘 예측 하는 지를 평가하는 것!

  특정 gene이 가질 수 있는 모든 cutoff를 이용해서 그림을 그린 것이 ROC curve이다!

Example of Classification

```
data(golub, package = "multtest")

Labels = factor(golub.cl, levels = 0:1, labels = c("ALL", "AML"))

ccnd3 = grep("CCND3", golub.gnames[,2], ignore.case = TRUE)

sort(golub[ccnd3, ])
```

```
##  [1] -0.74333  0.12758  0.42904  0.45827  0.49470  0.63637  0.73784  0.82667
##  [9]  0.88941  1.02250  1.10546  1.12058  1.27645  1.32551  1.36844  1.45014
## [17]  1.52405  1.78352  1.80861  1.81649  1.83051  1.83485  1.85111  1.90496
## [25]  1.92776  1.96403  1.99391  1.99927  2.06597  2.10892  2.17622  2.18119
## [33]  2.31428  2.33597  2.37351  2.44562  2.59385  2.76610
```

```
decision = golub[ccnd3, ] >= 1.27 #

decision # TRUE = positive(ALL), FALSE = negative(AML)
```

```
##  [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [13]  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE
## [25]  TRUE  TRUE  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [37] FALSE FALSE
```

```
Pred = factor(decision, levels = c("TRUE", "FALSE"),
                        labels = c("ALL", "notALL"))

data.frame(predict = Pred, labels = Labels)
```

```
##    predict labels
## 1      ALL    ALL
## 2      ALL    ALL
## 3      ALL    ALL
## 4      ALL    ALL
## 5      ALL    ALL
## 6      ALL    ALL
## 7      ALL    ALL
## 8      ALL    ALL
## 9      ALL    ALL
## 10     ALL    ALL
## 11     ALL    ALL
## 12     ALL    ALL
## 13     ALL    ALL
## 14     ALL    ALL
## 15     ALL    ALL
## 16     ALL    ALL
## 17  notALL    ALL
## 18     ALL    ALL
## 19     ALL    ALL
## 20     ALL    ALL
```

```
## 21  notALL    ALL
## 22     ALL    ALL
## 23     ALL    ALL
## 24     ALL    ALL
## 25     ALL    ALL
## 26     ALL    ALL
## 27     ALL    ALL
## 28  notALL    AML
## 29     ALL    AML
## 30  notALL    AML
## 31  notALL    AML
## 32  notALL    AML
## 33  notALL    AML
## 34  notALL    AML
## 35  notALL    AML
## 36  notALL    AML
## 37  notALL    AML
## 38  notALL    AML
```

```
table(Pred, Labels)
```

```
##         Labels
## Pred     ALL AML
##   ALL     25   1
##   notALL   2  10
```

```
perf <- function(pred, label) {
  tab <- table(pred, label)
  sensitivity <- tab[1, 1]/sum(tab[ ,1])
  specificity <- tab[2, 2]/sum(tab[ ,2])
  PV.positive <- tab[1, 1]/sum(tab[1, ])
  PV.negative <- tab[2, 2]/sum(tab[2, ])
  c(sensitivity, specificity, PV.positive, PV.negative)
}
perf(Pred, Labels)
```

```
## [1] 0.9259259 0.9090909 0.9615385 0.8333333
```

```
decision2 = golub[ccnd3, ] >= 2.18
Pred2 = factor(decision2, levels = c("TRUE", "FALSE"), labels = c("ALL", "notALL"))
perf(Pred2, Labels)
```

```
## [1] 0.2592593 1.0000000 1.0000000 0.3548387
```

- 지금 cutoff 값을 높혔다.
- 그럼 True보다 false가 많아질 것이다.
- 즉, false라고 예측한 비율이 높아질거다.
- 그럼 spec (= 진짜 false 중에서 false라고 예측한 ratio)는 당연히 높아질것이다.

```
decision3 = golub[ccnd3, ] >= 0.8
Pred3 = factor(decision3, levels = c("TRUE", "FALSE"), labels = c("ALL", "notALL"))
perf(Pred3, Labels)
```

```
## [1] 0.9629630 0.5454545 0.8387097 0.8571429
```

- 이제 이 gene에서 가능한 모든 cutoff를 이용해서 이 gene ALL, not ALL를 예측하기 좋은 classifier인지 검증해보자.

```
cutoff = sort(golub[ccnd3, ])
cutoff = c(cutoff, Inf)
cutoff
```

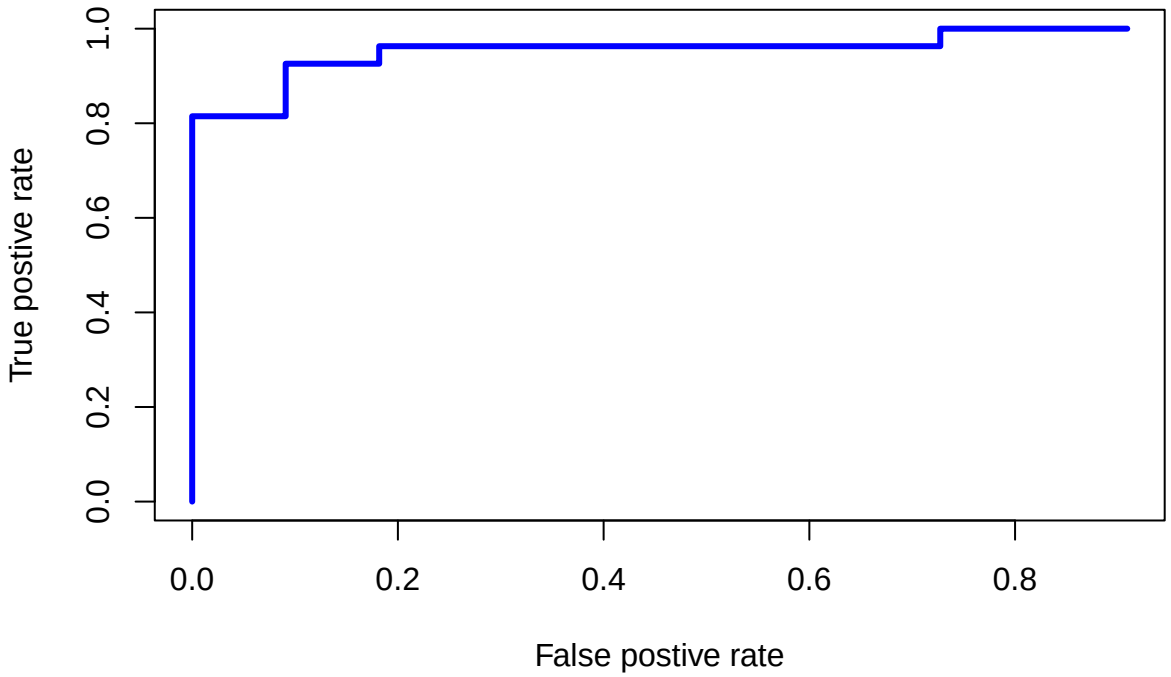
```
##  [1] -0.74333  0.12758  0.42904  0.45827  0.49470  0.63637  0.73784  0.82667
##  [9]  0.88941  1.02250  1.10546  1.12058  1.27645  1.32551  1.36844  1.45014
## [17]  1.52405  1.78352  1.80861  1.81649  1.83051  1.83485  1.85111  1.90496
## [25]  1.92776  1.96403  1.99391  1.99927  2.06597  2.10892  2.17622  2.18119
## [33]  2.31428  2.33597  2.37351  2.44562  2.59385  2.76610      Inf
```

```
res = matrix(0, length(cutoff), 4) # 4 -> sen, spec ,pv+, pv-
rownames(res) = round(cutoff, 3)
colnames(res) = c("sensitivity", "specificity", "PV.positive", "PV.negative")

for (i in 1:length(cutoff)){
  decision = golub[ccnd3, ] > cutoff[i]
  Pred = factor(decision, levels = c("TRUE", "FALSE"), labels = c("ALL", "notALL"))
  res[i, ] = perf(Pred, Labels)
}

head(res);tail(res)
```

```
##        sensitivity specificity PV.positive PV.negative
## -0.743    1.000000  0.09090909   0.7297297   1.0000000
## 0.128     1.000000  0.18181818   0.7500000   1.0000000
## 0.429     1.000000  0.27272727   0.7714286   1.0000000
## 0.458     0.962963  0.27272727   0.7647059   0.7500000
## 0.495     0.962963  0.36363636   0.7878788   0.8000000
## 0.636     0.962963  0.45454545   0.8125000   0.8333333
```

```
##       sensitivity specificity PV.positive PV.negative
## 2.336  0.14814815           1           1   0.3235294
## 2.374  0.11111111           1           1   0.3142857
## 2.446  0.07407407           1           1   0.3055556
## 2.594  0.03703704           1           1   0.2972973
## 2.766  0.00000000           1         NaN   0.2894737
## Inf    0.00000000           1         NaN   0.2894737
```

```
plot(1-res[,2], res[,1], type="l", xlab="False postive rate",
ylab="True postive rate", col="blue", lwd=3)
```



- 이제 이 곡선의 아래 면적을 구하면 CCND라는 gene이 classifier로 얼마나 좋은 성능을 보여주는지 알 수 있다.

Example of ROC Curve.

```
library(ROCR)

true = factor(golub.cl , levels = 0:1, labels = c("TRUE", "FALSE"))

predccnd3 = prediction(golub[ccnd3,], true)
```

- `prediction()` 함수(ROCR 패키지)는 예측 점수(혹은 확률, 연속형 값)와 실제 라벨을 받아서,

  ROC, PR 곡선, AUC 등 각종 성능평가 지표를 **만들기 위한 내부 데이터 구조(객체)**를 생성하는 역할을 한다.

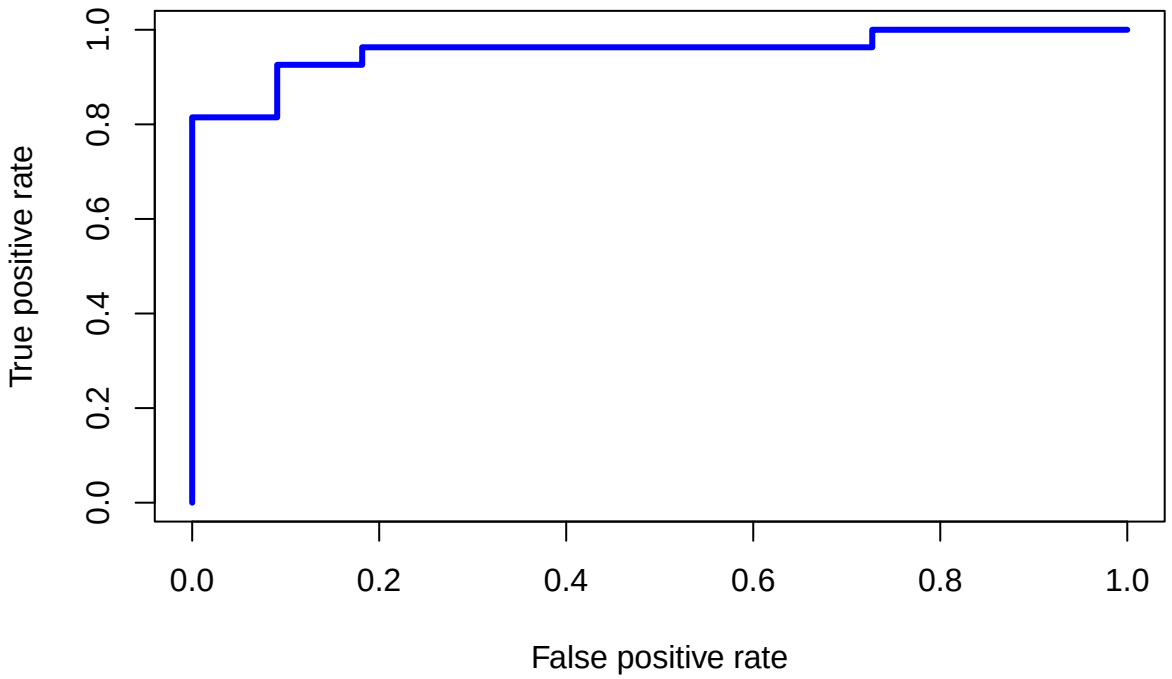
- 입력
  - 첫 번째 인자: 예측 점수

    (여기서는 `golub[ccnd3, ]`, 즉 각 샘플의 CCND3 유전자 발현값)
  - 두 번째 인자: 정답 라벨

    (`true` = ALL/AML 클래스, TRUE/FALSE로 인코딩됨)

  출력
  - ROCR 내부에서 이후 `performance()` 함수로 ROC 곡선, AUC, PR curve 등 다양한 성능 곡선을 계산할 수 있는 prediction 객체를 만듭니다.
    이 객체는 모든 threshold를 시뮬레이션 할 수 있도록 점수와 라벨을 잘 정렬해 저장합니다.

```
perfccnd3 = performance(predccnd3, "tpr", "fpr")
plot(perfccnd3, lwd = 3, col = 'blue')
```




```
slotNames(perfccnd3)
```

```
## [1] "x.name"       "y.name"       "alpha.name"   "x.values"     "y.values"
## [6] "alpha.values"
```

```
# Cut off
list(perfccnd3@alpha.name, perfccnd3@alpha.values)
```

```
## [[1]]
## [1] "Cutoff"
##
## [[2]]
## [[2]][[1]]
##  [1]      Inf  2.76610  2.59385  2.44562  2.37351  2.33597  2.31428  2.18119
##  [9]  2.17622  2.10892  2.06597  1.99927  1.99391  1.96403  1.92776  1.90496
## [17]  1.85111  1.83485  1.83051  1.81649  1.80861  1.78352  1.52405  1.45014
## [25]  1.36844  1.32551  1.27645  1.12058  1.10546  1.02250  0.88941  0.82667
## [33]  0.73784  0.63637  0.49470  0.45827  0.42904  0.12758 -0.74333
```

```
sort(cutoff, decreasing=TRUE)
```

```
##  [1]      Inf  2.76610  2.59385  2.44562  2.37351  2.33597  2.31428  2.18119
##  [9]  2.17622  2.10892  2.06597  1.99927  1.99391  1.96403  1.92776  1.90496
## [17]  1.85111  1.83485  1.83051  1.81649  1.80861  1.78352  1.52405  1.45014
## [25]  1.36844  1.32551  1.27645  1.12058  1.10546  1.02250  0.88941  0.82667
## [33]  0.73784  0.63637  0.49470  0.45827  0.42904  0.12758 -0.74333
```

```r
# False postive rate
list(perfccnd3@x.name, perfccnd3@x.values)
```

```
## [[1]]
## [1] "False positive rate"
##
## [[2]]
## [[2]][[1]]
##  [1] 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
##  [7] 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
## [13] 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
## [19] 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.09090909
## [25] 0.09090909 0.09090909 0.09090909 0.18181818 0.18181818 0.27272727
## [31] 0.36363636 0.45454545 0.54545455 0.63636364 0.72727273 0.72727273
## [37] 0.81818182 0.90909091 1.00000000
```

```r
sort(as.numeric(1-res[,2])) # 1 -fpr = spec
```

```
##  [1] 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
##  [7] 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
## [13] 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
## [19] 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
## [25] 0.09090909 0.09090909 0.09090909 0.09090909 0.18181818 0.18181818
## [31] 0.27272727 0.36363636 0.45454545 0.54545455 0.63636364 0.72727273
## [37] 0.72727273 0.81818182 0.90909091
```

```r
# True postive rate(sen)
list(perfccnd3@y.name, perfccnd3@y.values)
```

```
## [[1]]
## [1] "True positive rate"
##
## [[2]]
## [[2]][[1]]
##  [1] 0.00000000 0.03703704 0.07407407 0.11111111 0.14814815 0.18518519
##  [7] 0.22222222 0.25925926 0.29629630 0.33333333 0.37037037 0.40740741
## [13] 0.44444444 0.48148148 0.51851852 0.55555556 0.59259259 0.62962963
## [19] 0.66666667 0.70370370 0.74074074 0.77777778 0.81481481 0.81481481
## [25] 0.85185185 0.88888889 0.92592593 0.92592593 0.96296296 0.96296296
## [31] 0.96296296 0.96296296 0.96296296 0.96296296 0.96296296 1.00000000
## [37] 1.00000000 1.00000000 1.00000000
```

```r
sort(as.numeric(res[,1]))
```

```
##  [1] 0.00000000 0.00000000 0.03703704 0.07407407 0.11111111 0.14814815
##  [7] 0.18518519 0.22222222 0.25925926 0.29629630 0.33333333 0.37037037
## [13] 0.40740741 0.44444444 0.48148148 0.51851852 0.55555556 0.59259259
## [19] 0.62962963 0.66666667 0.70370370 0.74074074 0.77777778 0.81481481
## [25] 0.81481481 0.85185185 0.88888889 0.92592593 0.92592593 0.96296296
## [31] 0.96296296 0.96296296 0.96296296 0.96296296 0.96296296 0.96296296
## [37] 1.00000000 1.00000000 1.00000000
```

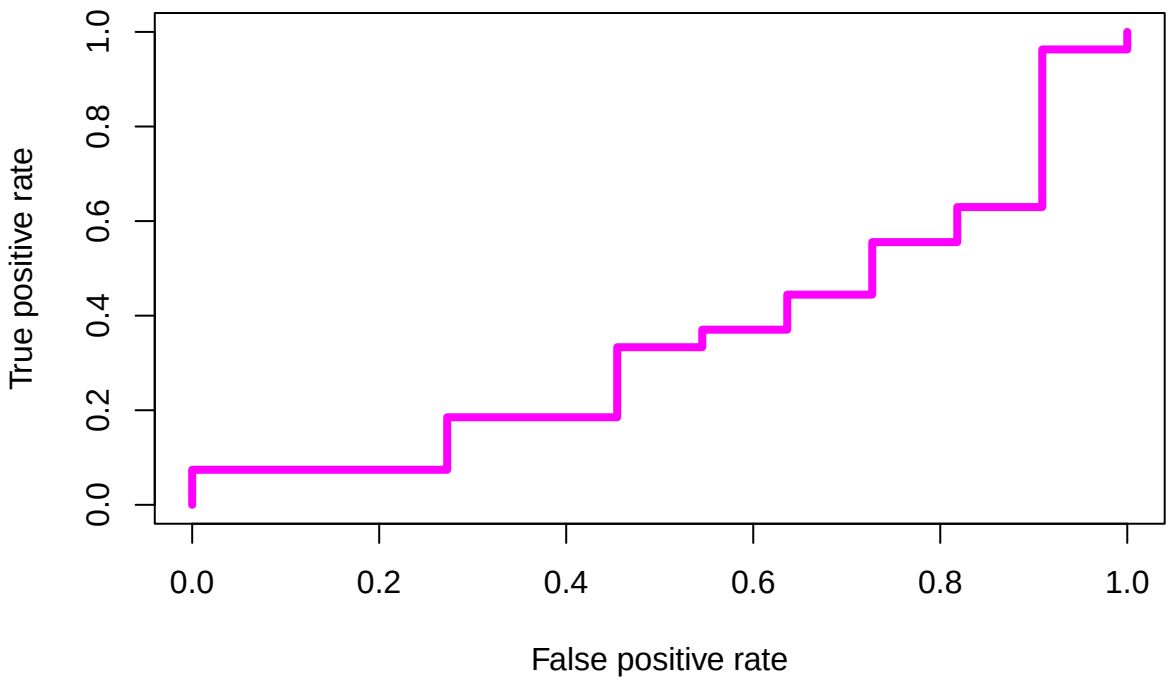
이게 중요!!!!

```
# value of AUC
performance(predccnd3, "auc")@y.values
```

```
## [[1]]
## [1] 0.956229
```

- 0.956으로 1에 매우 가까운 값!
- 즉, CCND3 gene은 ALL, not ALL를 나눈데 큰 역할을 하는 classifier이다..!
- 이번에는 Gdf5 gene으로 해볼까??

```
gdf5 = grep('GDF5', golub.gnames[, 2], ignore.case = TRUE)
true = factor(golub.cl , levels = 0:1, labels = c("TRUE", "FALSE"))
predgdf5 = prediction(golub[gdf5, ], true)
perfgdf5 = performance(predgdf5, "tpr", "fpr")
plot(perfgdf5, lwd = 4, col = 'magenta')
```




```
performance(predgdf5, "auc")@y.values
```

```
## [[1]]
## [1] 0.3535354
```

- In fact, this GDF5 expression classifer is performing worse than random guessing.

```r
true2 <- factor(golub.cl, levels=0:1, labels=c("FALSE", "TRUE"))
predgdf5 <- prediction(golub[gdf5, ], true2)
perfgdf5 <- performance(predgdf5, "tpr", "fpr" )
performance(predgdf5, "auc")@y.values
```

```
## [[1]]
## [1] 0.6464646
```

- 그럼 Gdf5 gene은 AML(not ALL)의 biomarker라고 할 수 있다..!!

Classification Probability

- Given a feature vector X and qualitative response Y (label : 1, 2, 3, 4..) taking values in the set C, the classification task is to build a function $C(X)$ that takes as input feature vector X and predicts it values for Y.
- Often we are more interested in estimating the prob that X belongs to each category in C/
- Suppose that we have only 2 classes such as “Positive” and “Negative”.
- Then, we just need to compare two probs,

$$P(Y = +, |X = x) \quad P(Y = -, |X = X)$$

- Equivalently, we just need to know if

$$P(Y = +, |X = x) > 0.5$$

- In general, Y is a “Positive if

$$P(Y = +, |X = x) > \alpha$$

- Within the framework of logistic regression, the diagnosis of a patient is seen as a response. For $i = 1, 2, \ldots, n$,
  - Response: $y_i = 0$ or 1
  - Predictor: $x_i = (x_{i1}, x_{i2}, \ldots, x_{ip})^{\mathrm{T}}$
- When the response has the values healthy or diseased, it may be assumed that the binomial distribution holds with a success probability p_i.
- The value of p_i is closely related to one or more predictor variables $x_1, x_2, \ldots, x_p$ via a linear combination such as

$$\eta_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}$$

- Then, we need to link y_i with η_i similar to linear regression.
- Let's write $p(x_i) = P(y_i = 1|x_i)$ for $x_i = \{x_{i1}, \ldots, x_{ip}\}$.

$$p(x_i) = \frac{e^{\eta_i}}{1 + e^{\eta_i}} = \frac{\exp(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip})}{1 + \exp(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip})}$$

- Note that $p(x_i)$ will have values between 0 and 1.
- A bit of rearrangement gives

$$\log\left(\frac{p(x_i)}{1 - p(x_i)}\right) = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}.$$

- This monotone transformation is called the log odds or logit transformation of $p(x_i)$.
- Logistic regression ensures that our estimate for $p(x_i)$ lies between 0 and 1.

Example of Logistic Regression

```
data(golub, package = "multtest")
Factor = factor(golub.cl, levels = 0:1, labels = c("ALL", "AML"))
ccnd3 = grep("ccnd3", golub.gnames[,2], ignore.case = TRUE)

head(data.frame(x = golub[ccnd3, ], y = Factor));tail(data.frame(x = golub[ccnd3, ], y = Factor))
```

```
##         x   y
## 1 2.10892 ALL
## 2 1.52405 ALL
## 3 1.96403 ALL
## 4 2.33597 ALL
## 5 1.85111 ALL
## 6 1.99391 ALL
```

```
##           x   y
## 33  1.02250 AML
## 34  0.12758 AML
## 35 -0.74333 AML
## 36  0.73784 AML
## 37  0.49470 AML
## 38  1.12058 AML
```

```
g = glm(Factor ~ golub[ccnd3, ], family = 'binomial')
summary(g)
```

```
##
## Call:
## glm(formula = Factor ~ golub[ccnd3, ], family = "binomial")
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)       4.844      1.849   2.620  0.00880 **
## golub[ccnd3, ]   -4.440      1.488  -2.984  0.00284 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 45.728  on 37  degrees of freedom
## Residual deviance: 18.270  on 36  degrees of freedom
## AIC: 22.27
##
## Number of Fisher Scoring iterations: 6
```

```
predict(g) # value of linear predictors,  #   = logit(^)
```

```
##           1           2           3           4           5           6
## -4.51938148 -1.92258632 -3.87607673 -5.52747275 -3.37471727 -4.00874252
```

```
##           7           8           9          10          11          12
## -4.32868551 -3.22100610 -4.81819030 -3.18601927 -6.01431357 -3.61380872
##          13          14          15          16          17          18
## -7.43722962 -1.04107811 -6.67244776 -3.71503964 -0.06406651 -0.82325403
##          19          20          21          22          23          24
## -3.28325424 -3.07462086  2.80942650 -4.84025687 -5.43117018 -4.03254066
##          25          26          27          28          29          30
## -1.23168528 -5.69414858 -3.30252363  0.89518528 -1.59442942  2.93920632
##          31          32          33          34          35          36
##  1.17374792  2.01867092  0.30427197  4.27767447  8.14447371  1.56814892
##          37          38
##  2.64767903 -0.13119859
```

$$\log\left(\frac{p(x_i)}{1-p(x_i)}\right) = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}.$$

`predict(g)`의 **기본값 `type = "link"`**이 적용됩니다.

- `glm()`에서 **link scale = 선형 예측값(linear predictor, η)** 을 의미합니다.
- 로지스틱 회귀(binomial·link = logit)에서는

$$\eta = \log\left(\frac{\pi}{1-\pi}\right)$$

  즉 **로그 오즈(logit)** 가 곧 선형 예측값입니다.

```
eta = cbind(1, golub[ccnd3, ]) %*% g$coef
head(data.frame(pred = predict(g), eta))
```

```
##        pred       eta
## 1 -4.519381 -4.519381
## 2 -1.922586 -1.922586
## 3 -3.876077 -3.876077
## 4 -5.527473 -5.527473
## 5 -3.374717 -3.374717
## 6 -4.008743 -4.008743
```

```
prob1 = predict(g, type = "response") # # ^ = expit()
prob1
```

```
##            1            2            3            4            5            6
## 0.0107783228 0.1275734351 0.0203109171 0.0039602786 0.0330950246 0.0178324424
##            7            8            9           10           11           12
```

```
## 0.0130132890 0.0383828334 0.0080166134 0.0396952452 0.0024375688 0.0262418189
##           13           14           15           16           17           18
## 0.0005885678 0.2609420259 0.0012636989 0.0237754373 0.4839888492 0.3050733574
##           19           20           21           22           23           24
## 0.0361501565 0.0441663434 0.9431830941 0.0078430240 0.0043588867 0.0174203788
##           25           26           27           28           29           30
## 0.2258865987 0.0033543122 0.0354847152 0.7099590724 0.1687616213 0.9497508627
##           31           32           33           34           35           36
## 0.7638218009 0.8827435097 0.5754865021 0.9863149871 0.9997097489 0.8275195617
##           37           38
## 0.9338677944 0.4672473194
```

```
prob2 = exp(eta) / (1 + exp(eta))
prob3 = exp(predict(g)) / (1 + exp(predict(g)))

head(data.frame(x = golub[ccnd3, ], pred1 = prob1, pred2 = prob2, pred3 = prob3))
```

```
##         x       pred1       pred2       pred3
## 1 2.10892 0.010778323 0.010778323 0.010778323
## 2 1.52405 0.127573435 0.127573435 0.127573435
## 3 1.96403 0.020310917 0.020310917 0.020310917
## 4 2.33597 0.003960279 0.003960279 0.003960279
## 5 1.85111 0.033095025 0.033095025 0.033095025
## 6 1.99391 0.017832442 0.017832442 0.017832442
```

```
tail(data.frame(x = golub[ccnd3, ], pred1 = prob1, pred2 = prob2, pred3 = prob3))
```

```
##           x     pred1     pred2     pred3
## 33  1.02250 0.5754865 0.5754865 0.5754865
## 34  0.12758 0.9863150 0.9863150 0.9863150
## 35 -0.74333 0.9997097 0.9997097 0.9997097
## 36  0.73784 0.8275196 0.8275196 0.8275196
## 37  0.49470 0.9338678 0.9338678 0.9338678
## 38  1.12058 0.4672473 0.4672473 0.4672473
```

- 현재 데이터 프레임에서 보이는 확률 값은 AML (level = 1)일 확률을 말하고 있다.

```
pred = predict(g, type = "response") < 0.5 # 0.5      ALL,    AML!!

est = factor(pred, levels = c(TRUE, FALSE), labels = c("ALL", "notALL"))

head(data.frame(pred = est, Labele = Factor));tail(data.frame(pred = est, Labele = Factor))
```

```
##   pred Labele
## 1  ALL    ALL
## 2  ALL    ALL
## 3  ALL    ALL
## 4  ALL    ALL
## 5  ALL    ALL
## 6  ALL    ALL
```

```
##      pred Labele
## 33 notALL    AML
## 34 notALL    AML
## 35 notALL    AML
## 36 notALL    AML
## 37 notALL    AML
## 38    ALL    AML
```

```
table(est, Factor)
```

```
##         Factor
## est      ALL AML
##   ALL     26   2
##   notALL   1   9
```

```
perf(est, Factor)
```

```
## [1] 0.9629630 0.8181818 0.9285714 0.9000000
```

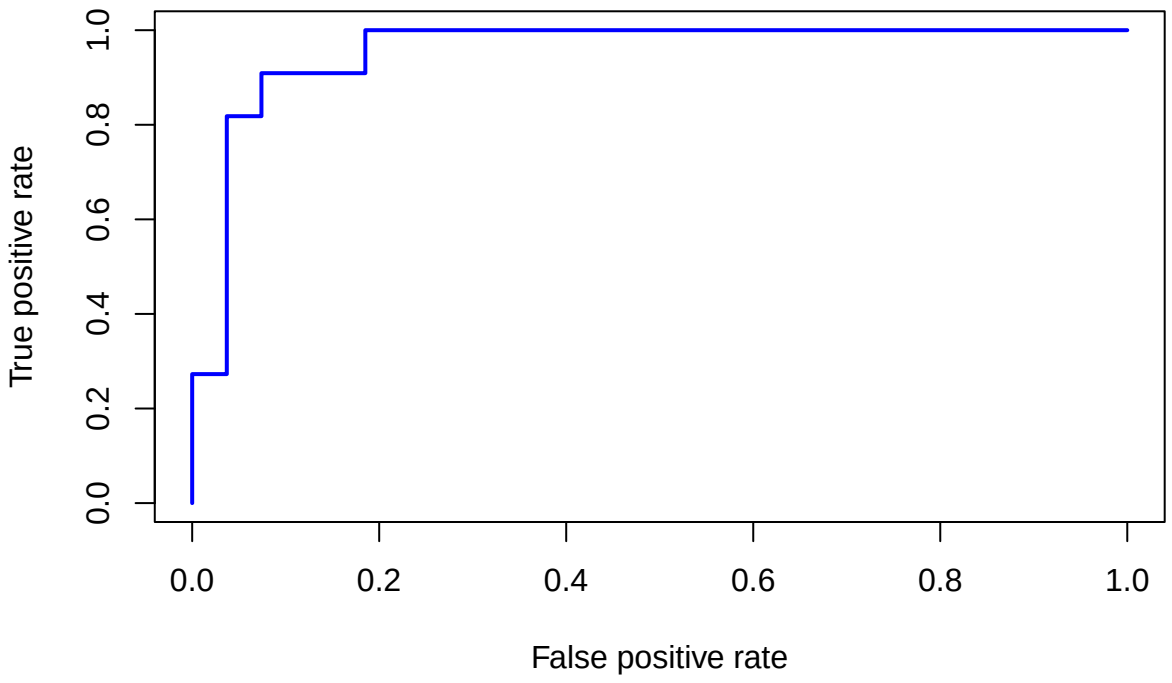
```
#
data(golub, package = "multtest")

Factor = factor(golub.cl, levels = 0:1, labels = c("ALL", "AML"))

ccnd3 = grep("ccnd3", golub.gnames[,2], ignore.case = TRUE)

g = glm(Factor ~ golub[ccnd3, ], family = 'binomial')

pred = predict(g, type = "response") < 0.5

est = factor(pred, levels = c(TRUE, FALSE), labels = c("ALL", "notALL"))

table(est, Factor)
```

```
##         Factor
## est      ALL AML
##   ALL     26   2
##   notALL   1   9
```

1. `Factor` **생성**	`levels = 0:1`, `labels = c("ALL","AML")`	→ **레벨 0 = "ALL"**, 레벨 1 = "AML"
2. `glm()`	`family = "binomial"` → **link = logit**	R의 binomial GLM은 **"두 번째 레벨(= AML)"**의 확률을 모델링
3. `predict(g, type = "response")`	**AML일 확률** $\hat{\pi} = P(\text{AML})$ 반환	값 범위 0–1
4. `pred = … < 0.5`	AML 확률이 0.5보다 **작으면 TRUE**	→ "AML이 아닐 가능성이 더 큼"
5. `est` **factor 변환**	`TRUE → "ALL"` `FALSE → "notALL"`	결국 $\hat{\pi}_{AML} < 0.5$ → **ALL로 분류**
6. `table(est, Factor)`	예측 vs 실제 혼동행렬	성능 확인

ROC Curve rr

```
library(ROCR)
true = factor(golub.cl, levels = 0:1, labels = c("FALSE", "TRUE"))
yprob = predict(g, type = "response")

pred = prediction(yprob, true)
perf_ = performance(pred, "tpr", "fpr")
plot(perf_, lwd = 2, col= "blue")
```

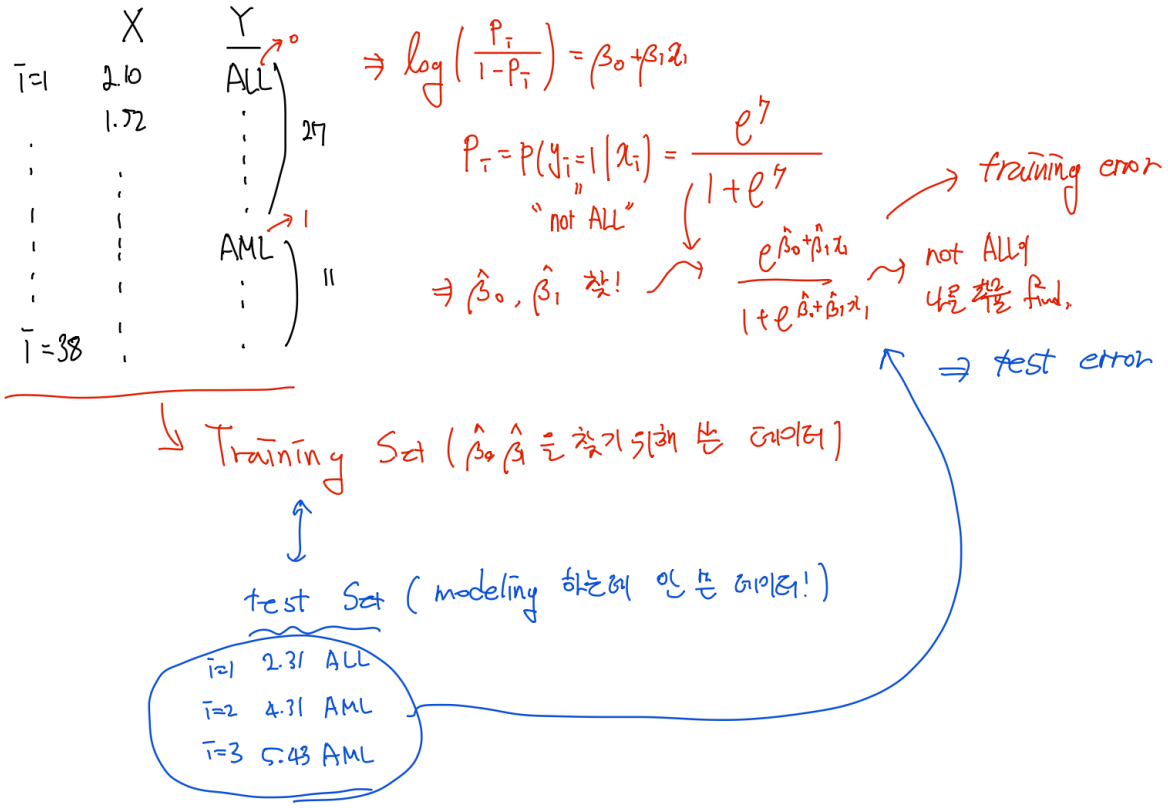


단계	코드	내부 동작	결과
① 라벨 지정	`Factor = factor(golub.cl, levels = 0:1, labels = c("ALL","AML"))`	레벨 0 → **"ALL"** (첫 번째 레벨) 레벨 1 → **"AML"** (두 번째 레벨)	factor 객체
② 로지스틱 회귀	`g = glm(Factor ~ golub[ccnd3, ], family = "binomial")`	R 의 binomial `glm()` 은 **두 번째 레벨**을 *success*로 간주 → **P(AML)** 을 학습	`predict(g, type="response")` ⇒ **AML** 일 확률
③ **ROC용 라벨**	`true = factor(golub.cl, levels = 0:1, labels = c("FALSE","TRUE"))`	레벨 0 → "FALSE" 레벨 1 → "TRUE" → **"TRUE" = AML**	ROCR에서 **양성(positive)** 클래스가 AML
④ `prediction()`	`prediction(scores, true)`	scores = AML 확률 true = AML ⇔ TRUE	ROC/AUC 계산 시 **AML 검출 능력** 평가

```
performance(pred, "auc")@y.values
```

```
## [[1]]
## [1] 0.956229
```

Training Sets and Test Sets



Example of Test Errors

```
set.seed(1234)

train = sample(1:ncol(golub), 19) # train data  19      !

test = setdiff(1:ncol(golub), train)
```

```
Data = data.frame(x = golub[ccnd3, ], y = Factor)
head(Data); tail(Data)
```

```
##         x   y
## 1 2.10892 ALL
## 2 1.52405 ALL
## 3 1.96403 ALL
## 4 2.33597 ALL
## 5 1.85111 ALL
## 6 1.99391 ALL
```

```
##           x   y
## 33  1.02250 AML
## 34  0.12758 AML
## 35 -0.74333 AML
## 36  0.73784 AML
## 37  0.49470 AML
## 38  1.12058 AML
```

```
g = glm(y ~ x, data = Data, family = 'binomial', subset = train)
p = predict(g, Data, type = "response")
```

`predict(~, Data,)` : Data 넣는 이유

- Train 데이터만 평가할 거면 생략해도 OK.
- Test 데이터까지 포함하거나, 모델 학습에 쓰지 않은 새 표본을 예측하려면 반드시 **`Data`** (또는 **`newdata = ...`**)를 명시해야 한다.

```
# p => AML
pred.train = p[train] > 0.5 # 0.5     AML(TRUE) ,    ALL(FALSE)
pred.test = p[test] > 0.5
```

```
t1 = table(pred.train, Factor[train])
t1
```

```
##
## pred.train ALL AML
##      FALSE  14   1
##      TRUE    0   4
```

```r
1 - sum(diag(t1)) / sum(t1) # training error
```

```
## [1] 0.05263158
```

```r
t2 = table(pred.test, Factor[test])
1 - sum(diag(t2)) / sum(t2) # test error
```

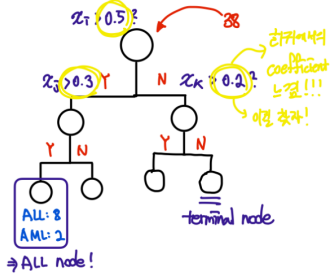
```
## [1] 0.1052632
```

```r
set.seed(12345)

K = 100

miss = matrix(0 , K, 2)

colnames(miss) = c("training", "test")

Data = data.frame(x = golub[ccnd3, ], y = Factor)

for(k in 1:K){

  # split test train
  train = sample(1:ncol(golub), 19)
  test = setdiff(1:ncol(golub), train)

  # make model
  g = glm(y ~ x, data = Data , family = binomial, subset = train)

  # est probs
  p = predict(g, Data, type = "response")
  pred.train = p[train] > 0.5
  pred.test = p[test] > 0.5

  t1 = table(pred.train, Factor[train])
  t2 = table(pred.test, Factor[test])

  miss[k, 1] = 1 - sum(diag(t1)) / sum(t1)
  miss[k, 2] = 1 - sum(diag(t2)) / sum(t2)
}

apply(miss , 2, summary)
```

```
##           training       test
## Min.    0.00000000 0.00000000
## 1st Qu. 0.10526316 0.05263158
## Median  0.10526316 0.10526316
## Mean    0.09578947 0.11368421
## 3rd Qu. 0.10526316 0.15789474
## Max.    0.15789474 0.36842105
```

Classification Trees

- A tree model resembles a linear model, where the criterion is the factor indicating class membership and the predictor variables are the gene expression values.
- For a classification tree, we predict that each observation belongs to the most commonly occurring class of training observations in the region to which it belongs.
- In Golub data, the gene expression values can serve as predictors to form a decision tree.
- An example decision is, if $x_j < t$ then the patient j is AML, and otherwise if $x_j \geq t$ then patient j is ALL.



- A training set is used to estimate the threshold values that construct the tree.

- The goal is to find boxes $R_1, \ldots, R_J$ that minimize the classification error rate, also called misclassification rate.
- The classification error rate is simply the fraction of the training observations in that region that do not belong to the most common class;

$$\text{Classification Error Rate} = \sum_{j=1}^{J} \sum_{k=1}^{K} \left(1 - \max_{k}(\hat{p}_{jk})\right),$$

  where $\hat{p}_{jk}$ represents the proportion of training observations in the jth region that are from the kth class.
- Unfortunately, it is computationally infeasible to consider every possible partition of the feature space into J boxes.

- When many predictor variables are involved, say 3051, then we have a teemendous gene(variable) selection problem.

  `rpart()` -> 자동으로 해줌!

Example of Classification Trees

```
library(rpart)
library(rpart.plot)
```

```
n = 10
factor = factor(c(rep(1, n), rep(2, n), rep(3, n)))
levels(factor) = c("ALL", "ALL2", "AML")
factor
```

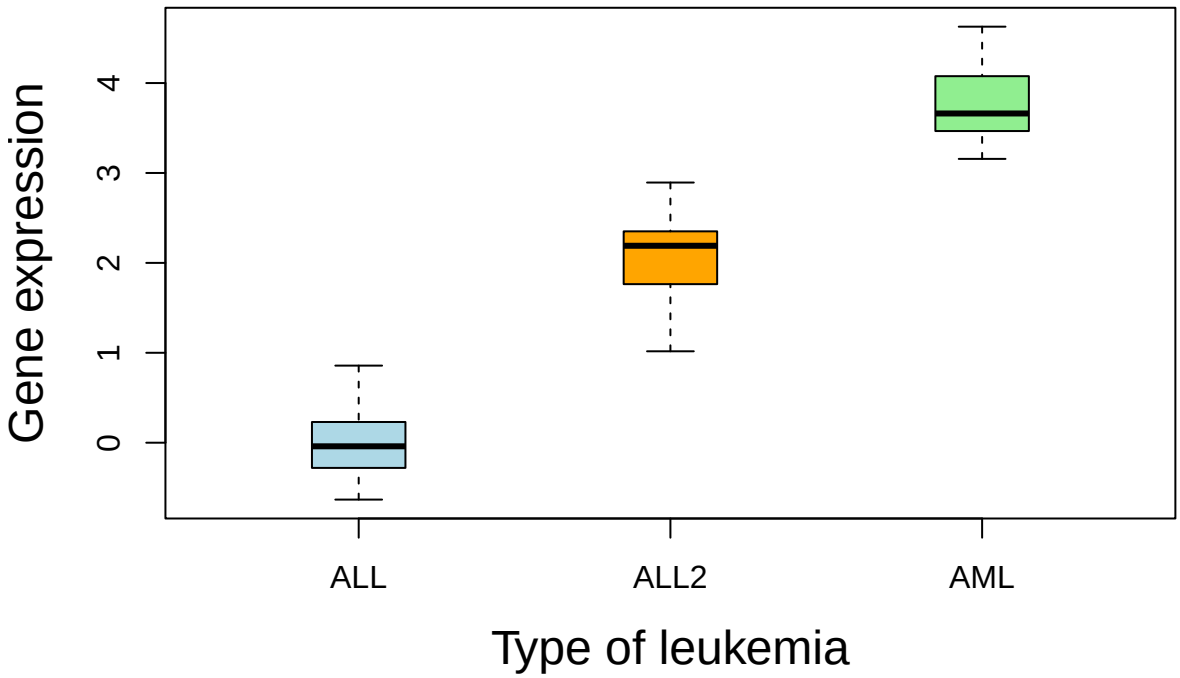
```
##  [1] ALL  ALL  ALL  ALL  ALL  ALL  ALL  ALL  ALL  ALL  ALL2 ALL2 ALL2 ALL2 ALL2
## [16] ALL2 ALL2 ALL2 ALL2 ALL2 AML  AML  AML  AML  AML  AML  AML  AML  AML  AML
## Levels: ALL ALL2 AML
```

```
set.seed(123)
geneA <- c(rnorm(n, 0, 0.5), rnorm(n, 2, 0.5),
           rnorm(n, 4, 0.5))
```

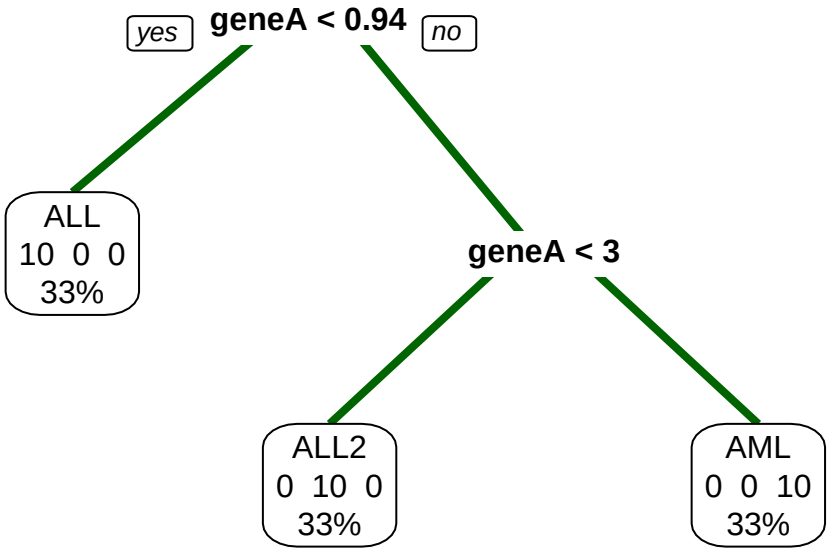
```
tapply(geneA, factor, range)
```

```
## $ALL
## [1] -0.6325306  0.8575325
##
## $ALL2
## [1] 1.016691 2.893457
##
## $AML
## [1] 3.156653 4.626907
```

```
boxplot(geneA ~ factor, cex.lab=1.5, main=NULL, boxwex=0.3,
col=c("lightblue", "orange", "lightgreen"),
xlab="Type of leukemia", ylab="Gene expression")
```




```
data = data.frame(factor, geneA)

rpartfit = rpart(factor ~ geneA, method = "class", data = data)

prp(rpartfit, branch.lwd=4, branch.col="darkgreen", extra=101)
```




```
rpartfit
```

```
## n= 30
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
```

```
## 1) root 30 20 ALL (0.3333333 0.3333333 0.3333333)
##   2) geneA< 0.937112 10  0 ALL (1.0000000 0.0000000 0.0000000) *
##   3) geneA>=0.937112 20 10 ALL2 (0.0000000 0.5000000 0.5000000)
##     6) geneA< 3.025055 10  0 ALL2 (0.0000000 1.0000000 0.0000000) *
##     7) geneA>=3.025055 10  0 AML (0.0000000 0.0000000 1.0000000) *
```

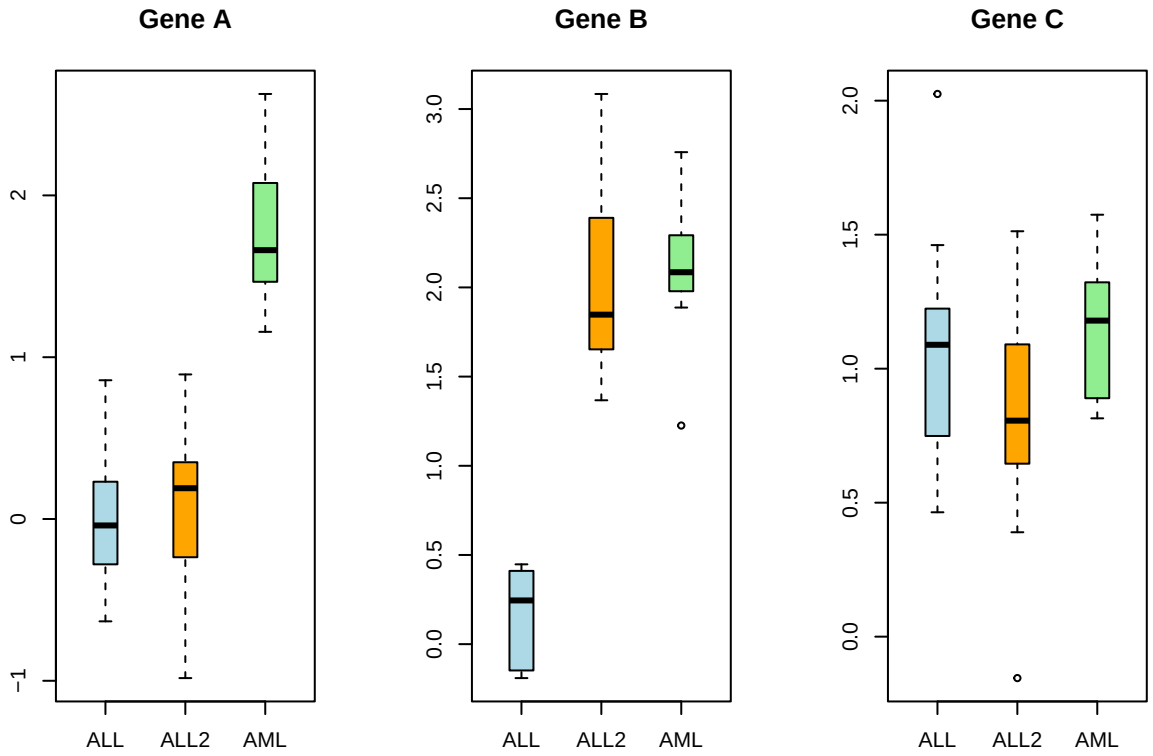
Example of Classification Trees : Gene Selection

```
set.seed(123)
geneA <- c(rnorm(20, 0, 0.5), rnorm(10, 2, 0.5))
geneB <- c(rnorm(10, 0, 0.5), rnorm(20, 2, 0.5))
geneC <- c(rnorm(30, 1, 0.5))
data <- data.frame(factor, geneA, geneB, geneC)
data
```

```
##    factor       geneA       geneB      geneC
## 1     ALL -0.28023782  0.21323211  1.1898197
## 2     ALL -0.11508874 -0.14753574  0.7488383
## 3     ALL  0.77935416  0.44756283  0.8333963
## 4     ALL  0.03525420  0.43906674  0.4907123
## 5     ALL  0.06464387  0.41079054  0.4641044
## 6     ALL  0.85753249  0.34432013  1.1517643
## 7     ALL  0.23045810  0.27695883  1.2241049
## 8     ALL -0.63253062 -0.03095586  1.0265021
## 9     ALL -0.34342643 -0.15298133  1.4611337
## 10    ALL -0.22283099 -0.19023550  2.0250423
## 11   ALL2  0.61204090  1.65264651  0.7544844
## 12   ALL2  0.17990691  1.89604136 -0.1545844
## 13   ALL2  0.20038573  1.36730182  1.5028693
## 14   ALL2  0.05534136  3.08447798  0.6453996
## 15   ALL2 -0.27792057  2.60398100  0.6559957
## 16   ALL2  0.89345657  1.43844571  1.5127857
## 17   ALL2  0.24892524  1.79855758  0.8576135
## 18   ALL2 -0.98330858  1.76667232  0.3896411
## 19   ALL2  0.35067795  2.38998256  1.0906517
## 20   ALL2 -0.23639570  1.95831547  0.9305543
## 21    AML  1.46608815  2.12665926  1.0028821
## 22    AML  1.89101254  1.98572662  1.1926402
## 23    AML  1.48699778  1.97856477  0.8146700
## 24    AML  1.63555439  2.68430114  1.3221883
## 25    AML  1.68748037  1.88711451  0.8897567
## 26    AML  1.15665334  2.75823530  1.1658910
## 27    AML  2.41889352  1.22562360  1.5484195
## 28    AML  2.07668656  2.29230687  1.2175907
## 29    AML  1.43093153  2.06192712  0.8370342
## 30    AML  2.62690746  2.10797078  1.5744038
```

```
par(mfrow=c(1,3))
boxplot(geneA ~ factor, main="Gene A", boxwex=0.3, ylab="",
col=c("lightblue", "orange", "lightgreen"), xlab="")
boxplot(geneB ~ factor, main="Gene B", boxwex=0.3, ylab="",
```

```
col=c("lightblue", "orange", "lightgreen"), xlab="")
boxplot(geneC ~ factor, main="Gene C", boxwex=0.3, ylab="",
col=c("lightblue", "orange", "lightgreen"), xlab="")
```




```
tapply(geneA, factor, range)
```

```
## $ALL
## [1] -0.6325306  0.8575325
##
## $ALL2
## [1] -0.9833086  0.8934566
##
## $AML
## [1] 1.156653 2.626907
```

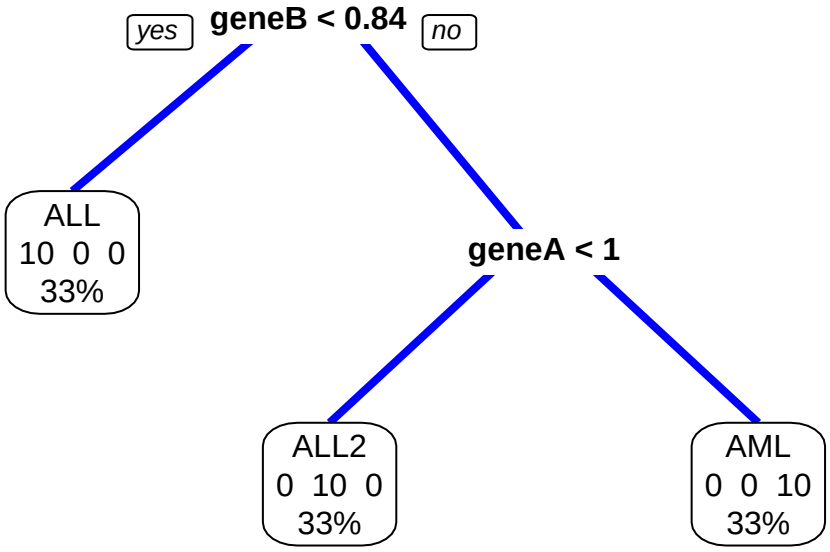
```
tapply(geneB, factor, range)
```

```
## $ALL
## [1] -0.1902355  0.4475628
##
## $ALL2
## [1] 1.367302 3.084478
##
## $AML
## [1] 1.225624 2.758235
```

```
tapply(geneC, factor, range)
```

```
## $ALL
## [1] 0.4641044 2.0250423
##
## $ALL2
## [1] -0.1545844  1.5127857
##
## $AML
## [1] 0.814670 1.574404
```

```
rpartFit = rpart(factor ~ geneA + geneB + geneC, method = 'class', data = data)

prp(rpartFit, branch.lwd=4, branch.col="blue", extra=101)
```

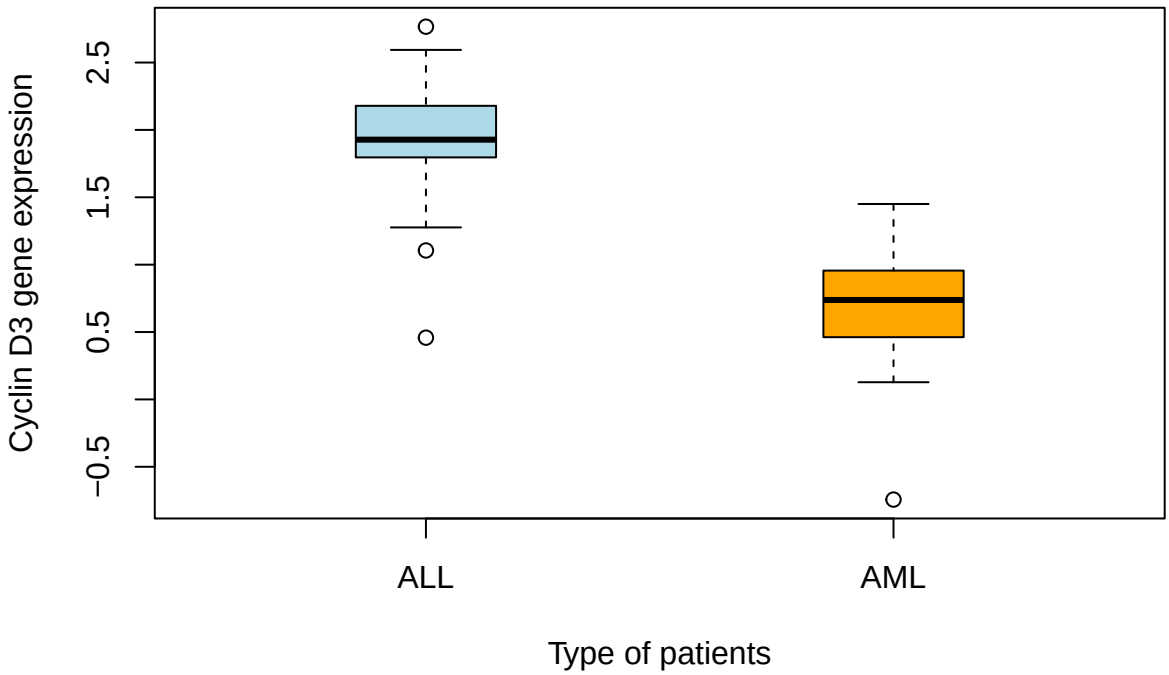


- Gene C는 분류를 할 때 전혀 이용되지 않았다!

Example of Classification Trees : Classification by CCND3

```
data(golub, package = 'multtest')
golubFactor = factor(golub.cl, levels = 0:1, labels = c("ALL", "AML"))
ccnd3 = grep("CCND3", golub.gnames[, 2], ignore.case = TRUE)
```

```
boxplot(golub[ccnd3, ] ~ golubFactor, main="", boxwex=0.3,
col=c("lightblue", "orange"), xlab="Type of patients",
      ylab="Cyclin D3 gene expression")
```

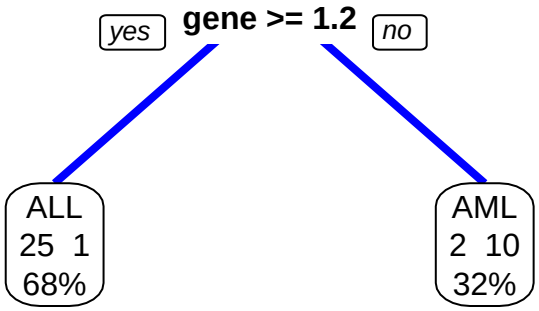


- 대충 boxplot를 보면 CCND3는 ALL과 AML를 잘 classification 해주는 classifier가 될 듯??!?!

```
tapply (golub[ccnd3, ], golubFactor, range)
```

```
## $ALL
## [1] 0.45827 2.76610
##
## $AML
## [1] -0.74333  1.45014
```

```
gene = golub[ccnd3, ]

tree = rpart(golubFactor ~ gene, method = "class")

prp(tree, branch.lwd=4, branch.col="blue", extra=101)
```



- Note that $(25+10)/38 = 0.921$ of the ALL, AML patients are correctly classified by CCND3 gene expression.

```r
predict(tree, type = 'prob')
```

```
##          ALL        AML
## 1  0.9615385 0.03846154
## 2  0.9615385 0.03846154
## 3  0.9615385 0.03846154
## 4  0.9615385 0.03846154
## 5  0.9615385 0.03846154
## 6  0.9615385 0.03846154
## 7  0.9615385 0.03846154
## 8  0.9615385 0.03846154
## 9  0.9615385 0.03846154
## 10 0.9615385 0.03846154
## 11 0.9615385 0.03846154
## 12 0.9615385 0.03846154
## 13 0.9615385 0.03846154
## 14 0.9615385 0.03846154
## 15 0.9615385 0.03846154
## 16 0.9615385 0.03846154
## 17 0.1666667 0.83333333
## 18 0.9615385 0.03846154
## 19 0.9615385 0.03846154
## 20 0.9615385 0.03846154
## 21 0.1666667 0.83333333
## 22 0.9615385 0.03846154
## 23 0.9615385 0.03846154
## 24 0.9615385 0.03846154
## 25 0.9615385 0.03846154
## 26 0.9615385 0.03846154
## 27 0.9615385 0.03846154
## 28 0.1666667 0.83333333
## 29 0.9615385 0.03846154
## 30 0.1666667 0.83333333
## 31 0.1666667 0.83333333
## 32 0.1666667 0.83333333
## 33 0.1666667 0.83333333
## 34 0.1666667 0.83333333
## 35 0.1666667 0.83333333
## 36 0.1666667 0.83333333
## 37 0.1666667 0.83333333
## 38 0.1666667 0.83333333
```

```r
predict(tree, type = "class")
```

```
##   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20
## ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL AML ALL ALL ALL
##  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38
## AML ALL ALL ALL ALL ALL ALL AML ALL AML AML AML AML AML AML AML AML AML
## Levels: ALL AML
```

```
pred = predict(tree, type = "class")
data.frame(pred = pred, golub=golubFactor)[c(17, 21, 29),]
```

```
##    pred golub
## 17  AML   ALL
## 21  AML   ALL
## 29  ALL   AML
```

- Hence, patients 17 and 21 are erroneously predicted as AML and patient 29 is erroneously predicted to be in the ALL class.

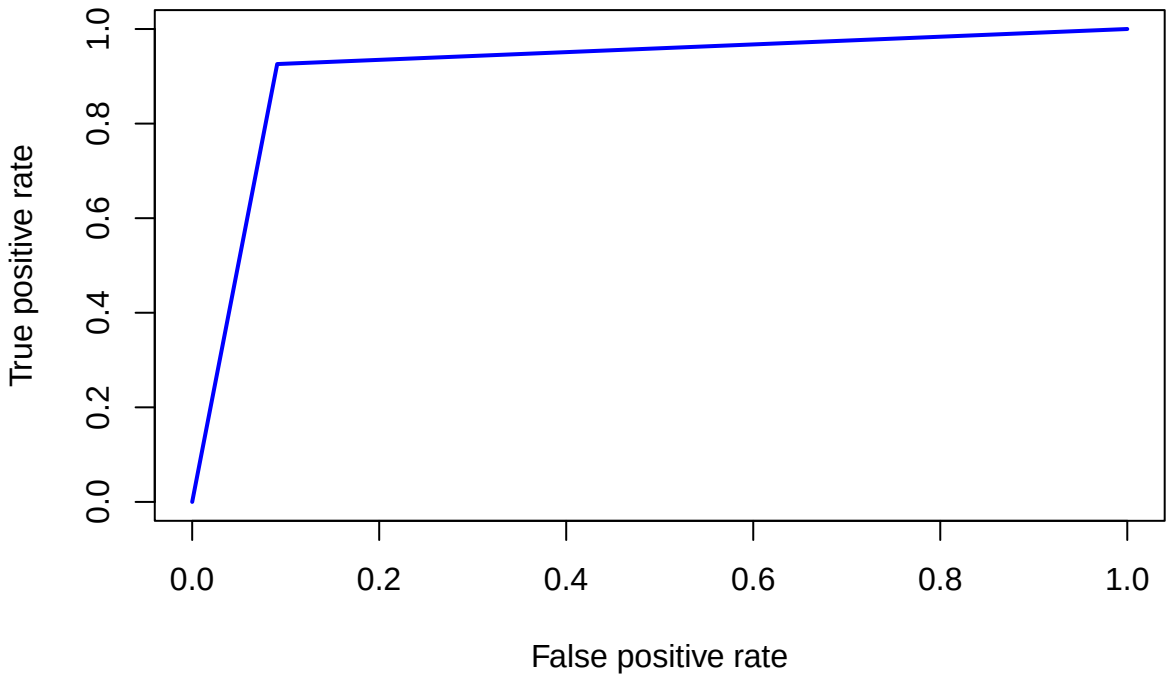
```
table(pred, golubFactor)
```

```
##      golubFactor
## pred  ALL AML
##   ALL  25   1
##   AML   2  10
```

```
perf(pred, golubFactor)
```

```
## [1] 0.9259259 0.9090909 0.9615385 0.8333333
```

```
library(ROCR)
true <- factor(golub.cl, levels=0:1, labels=c("TRUE","FALSE"))
pred_ALL = predict(tree, type = 'prob')[,1]

pred <- prediction(pred_ALL, true)
# prediction       !

perf <- performance(pred, "tpr", "fpr" )
plot(perf, lwd=2, col="blue")
```



```
performance(pred, "auc")@y.values
```

```
## [[1]]
## [1] 0.9175084
```

Example of Classification Trees : leukemia data

```
library(ALL)
data(ALL)
```

```
ALLB123 <- ALL[ ,ALL$BT %in% c("B1","B2","B3")]
ALLB123$BT
```

```
##  [1] B2 B2 B1 B2 B1 B1 B1 B2 B2 B3 B3 B3 B2 B3 B2 B3 B2 B3 B2 B2 B2 B1 B1 B2 B1
## [26] B2 B1 B2 B2 B2 B2 B1 B2 B2 B2 B2 B2 B2 B2 B2 B2 B1 B2 B2 B3 B3 B3 B3 B3 B3
## [51] B1 B1 B1 B1 B3 B3 B3 B3 B3 B3 B3 B3 B1 B3 B1 B2 B2 B1 B3 B2 B2 B3 B1 B2 B2
## [76] B2 B1 B2
## Levels: B B1 B2 B3 B4 T T1 T2 T3 T4
```

```
table(ALLB123$BT)
```

```
##
##  B B1 B2 B3 B4  T T1 T2 T3 T4
##  0 19 36 23  0  0  0  0  0  0
```

```
table(ALL$BT)
```

```
##
##  B B1 B2 B3 B4  T T1 T2 T3 T4
##  5 19 36 23 12  5  1 15 10  2
```

```
names = featureNames(ALL)
```

```
library(hgu95av2.db)
symb <- mget(names, env=hgu95av2SYMBOL)
#unlist(symb)
#unlist(symb)[1:100]
ALLBTnames <- ALLB123[names, ]
dim(ALLBTnames)
```

```
## Features  Samples
##    12625       78
```

```
dim(ALL)
```

```
## Features  Samples
##    12625      128
```

```
probeData <- as.matrix(exprs(ALLBTnames))
row.names(probeData) <- unlist(symb)
probeData[1:20, 1:5]
```

```
##            01005     01010    04006    04007     04008
## MAPK3   7.597323  7.479445 7.384684 7.905312  7.065914
## TIE1    5.046194  4.932537 4.922627 4.844565  5.147762
## CYP2C19 3.900466  4.208155 4.206798 3.416923  3.945869
## CXCR5   5.903856  6.169024 6.116890 5.687997  6.208061
## CXCR5   5.925260  5.912780 6.170245 5.615210  5.923487
## DUSP1   8.570990 10.428299 9.937155 9.983809 10.063484
## MMP10   3.656143  3.853979 3.874289 3.547361  3.771648
## DDR1    7.623562  7.543604 6.816397 7.516981  7.726716
## EIF2AK2 8.903547  9.903953 9.533983 8.871669  9.424092
## HINT1   9.371888  9.322177 9.135370 9.627175  9.189420
## RABGGTA 7.985255  8.015146 7.827044 7.834880  7.984250
## MAPK11  5.399275  5.620731 5.967741 5.185482  5.484939
## YWHAE   5.698984  6.172831 6.680342 6.658723  6.372048
## KAT2B   3.432633  3.921244 3.784770 3.833315  3.948326
## SMAD5   3.442516  3.002145 3.474333 3.851746  3.851362
## POLG    7.318896  7.653855 6.971806 7.446160  7.475994
## LIMK1   5.479742  6.110416 5.962049 5.154362  5.696765
## IL13RA2 3.384669  3.361198 3.020869 3.177107  3.207662
## MSH6    4.921528  4.115588 4.691007 5.406613  4.431900
## WNT10B  6.281061  6.147906 6.260564 5.782521  6.032982
```

```
dim(ALL)
```

Select the gene with an ANOVA p-value smaller than 0.00001

```
## Features  Samples
##    12625      128
```

```
fun = function(x) anova(lm(x ~ ALLB123$BT))$Pr[1]

dim(exprs(ALLB123))
```

```
## [1] 12625    78
```

```
anova.pValue = apply(exprs(ALLB123), 1, fun)

ww = anova.pValue < 0.00001

sum(ww)
```

```
## [1] 82
```

```
diagnosed = factor(ALLBTnames$BT)
```