Classification Analysis

Lavinia Carabet

Classification Analysis

Used to identify which population a particular observation comes from

Discriminant Analysis

Multivariate technique concerned with separating distinct groups into different classes and allocating new observations to the previously identified groups

Supervised method - the training of the model is optimized by providing class membership information

Generative classifier

Load GEO lung cancer data set

```
dat <- read.table('./GEOLungCancer.txt', header=T, row.names=1)
colnames(dat)
```

```
##  [1] "Adeno1"  "Adeno2"  "Adeno3"  "Adeno4"  "Adeno5"  "Adeno6"  "Adeno7"
##  [8] "Adeno8"  "Adeno9"  "Adeno10" "SCLC1"   "SCLC2"   "SCLC3"   "SCLC4"
## [15] "SCLC5"   "SCLC6"   "SCLC7"   "SCLC8"   "SCLC9"   "Normal1" "Normal2"
## [22] "Normal3" "Normal4" "Normal5"
```

```
dim(dat)
```

```
## [1] 3013   24
```

Prepare data matrix to include class membership

```
library(MASS)
clas <- factor(c(rep('Adeno',10), rep('SCLC',9), rep('Normal',5)))
clas
```

```
##  [1] Adeno  Adeno  Adeno  Adeno  Adeno  Adeno  Adeno  Adeno  Adeno  Adeno
## [11] SCLC   SCLC   SCLC   SCLC   SCLC   SCLC   SCLC   SCLC   SCLC   Normal
## [21] Normal Normal Normal Normal
## Levels: Adeno Normal SCLC
```

```
length(clas)
```

```
## [1] 24
```

```
dat <- data.frame(clas,t(dat))
dim(dat)
```

```
## [1]   24 3014
```

```
dat[, 1:5]
```

```
##           clas X1007_s_at X1053_at X121_at X1294_at
## Adeno1   Adeno       9.26     5.15    7.35     6.14
## Adeno2   Adeno       8.69     4.57    7.32     6.32
## Adeno3   Adeno       8.51     5.20    7.13     6.09
## Adeno4   Adeno       8.38     4.92    7.50     6.08
## Adeno5   Adeno       7.70     4.60    7.40     5.98
## Adeno6   Adeno       8.16     5.24    7.35     6.09
## Adeno7   Adeno       7.73     4.64    7.32     6.43
## Adeno8   Adeno       8.33     5.64    7.32     5.86
## Adeno9   Adeno       7.47     5.64    7.09     6.63
## Adeno10  Adeno       8.39     5.42    7.08     6.11
## SCLC1     SCLC       7.88     5.85    7.06     5.69
## SCLC2     SCLC       8.43     5.31    7.12     5.97
## SCLC3     SCLC       7.84     6.18    7.20     5.71
## SCLC4     SCLC       8.31     5.76    7.09     5.81
## SCLC5     SCLC       8.37     5.90    7.08     5.44
## SCLC6     SCLC       8.24     5.65    7.05     5.61
## SCLC7     SCLC       8.16     5.84    7.03     6.02
## SCLC8     SCLC       8.19     5.69    7.03     5.79
## SCLC9     SCLC       8.23     5.10    7.07     5.66
## Normal1 Normal       8.96     4.57    7.34     6.02
## Normal2 Normal       8.15     4.82    6.89     6.08
## Normal3 Normal       9.40     5.31    7.23     6.33
## Normal4 Normal       8.96     4.93    7.14     6.22
## Normal5 Normal       9.32     4.79    7.29     6.43
```

Create training and test sets

```
#training set - first 6 Adeno (adenocarcinoma), first 7 SCLC (small cell lung cancer),
#               and first 3 Normal samples
training.set <- dat[c(paste('Adeno',1:6,sep=''),
                      paste('SCLC',1:7,sep=''),
                      paste('Normal',1:3,sep='')),]
# or
#training.set <- dat[c(1:6,11:17,20:22),]
dim(training.set)
```

```
## [1]   16 3014
```

```
#first 2 genes training set
training.set[,2:3]
```

```
##         X1007_s_at X1053_at
## Adeno1        9.26     5.15
```

```
## Adeno2        8.69     4.57
## Adeno3        8.51     5.20
## Adeno4        8.38     4.92
## Adeno5        7.70     4.60
## Adeno6        8.16     5.24
## SCLC1         7.88     5.85
## SCLC2         8.43     5.31
## SCLC3         7.84     6.18
## SCLC4         8.31     5.76
## SCLC5         8.37     5.90
## SCLC6         8.24     5.65
## SCLC7         8.16     5.84
## Normal1       8.96     4.57
## Normal2       8.15     4.82
## Normal3       9.40     5.31
```

```
#test set - remaining samples
test.set <- dat[!row.names(dat) %in% row.names(training.set),]
#or
#test.set <-dat[setdiff(row.names(dat), row.names(training.set)),]
#or
#test.set <- dat[c(-(1:6),-(11:17),-(20:22)),]
dim(test.set)
```

```
## [1]    8 3014
```

```
#first 2 genes test set
test.set[,2:3]
```

```
##         X1007_s_at X1053_at
## Adeno7        7.73     4.64
## Adeno8        8.33     5.64
## Adeno9        7.47     5.64
## Adeno10       8.39     5.42
## SCLC8         8.19     5.69
## SCLC9         8.23     5.10
## Normal4       8.96     4.93
## Normal5       9.32     4.79
```

```
#save sample class test set
sample.class <- test.set$clas
#or
#sample.class <- test.set[,1]
sample.class
```

```
## [1] Adeno  Adeno  Adeno  Adeno  SCLC   SCLC   Normal Normal
## Levels: Adeno Normal SCLC
```

```
#remove first column (sample class) test set
test.set <- test.set[,-1]
dim(test.set)
```

```
## [1]    8 3013
```

```
test.set[,1:2]
```

```
##         X1007_s_at X1053_at
## Adeno7        7.73     4.64
## Adeno8        8.33     5.64
## Adeno9        7.47     5.64
## Adeno10       8.39     5.42
## SCLC8         8.19     5.69
## SCLC9         8.23     5.10
## Normal4       8.96     4.93
## Normal5       9.32     4.79
```

Run a classifier to see if we can predict the lung cancer types and discriminate them from both each other and the normal samples

Train the model using `linear discriminant analysis (LDA)` method on the training set, and then predict sample classes and assess the model accuracy using the test set

LDA aims to find the linear combinations of the features/original explanatory variables (genes) that maximize the separation between groups

Fit the model using `lda` function in MASS package

Use only the first 2 genes as discriminators

```
clas.train <- training.set$clas

# the lda model
lda.model <- lda(formula=clas.train~., data=training.set[,2:3])
#or
#lda.model <- lda(clas.train~X1007_s_at+X1053_at, training.set[,2:3])
#or
#lda.model <- lda(training.set[,2:3], clas.train)
lda.model
```

```
## Call:
## lda(clas.train ~ ., data = training.set[, 2:3])
##
## Prior probabilities of groups:
##  Adeno Normal   SCLC
## 0.3750 0.1875 0.4375
##
## Group means:
##        X1007_s_at X1053_at
## Adeno    8.450000 4.946667
## Normal   8.836667 4.900000
## SCLC     8.175714 5.784286
##
## Coefficients of linear discriminants:
##                   LD1       LD2
## X1007_s_at -1.029112 -2.063670
## X1053_at    3.218055 -1.052519
##
```

```
## Proportion of trace:
##    LD1    LD2
## 0.9723 0.0277
```

The `coefficients of linear discriminants` that transform observations (samples) to discriminant functions (LD1, LD2) are contained in the `scaling` component of the lda object

```
lda.model$scaling
```

```
##                  LD1       LD2
## X1007_s_at -1.029112 -2.063670
## X1053_at    3.218055 -1.052519
```

```
# the first discriminant function (LD1) is a linear combination of the variables (X1007_s_at
# and X1053_at) with LD1 coefficients
lda.model$scaling[1,1] * training.set[,2] + lda.model$scaling[2,1] * training.set[,3]
```

```
##  [1]  7.043404  5.763526  7.976141  7.208870  6.878889  8.465053 10.716218
##  [8]  8.412456 11.819341  9.984075 10.372856  9.702127 10.395886  5.485666
## [15]  7.123761  7.414217
```

```
 #the second discriminant function
lda.model$scaling[1,2] * training.set[,2] + lda.model$scaling[2,2] * training.set[,3]
```

```
##  [1] -24.53006 -22.74330 -23.03493 -22.47195 -20.73185 -22.35475 -22.41895
##  [8] -22.98561 -22.68374 -23.21161 -23.48278 -22.95137 -22.98626 -23.30049
## [15] -21.89205 -24.98737
```

The `proportion of trace` is the proportion of between-class variance that is explained by successive discriminant functions calculated from the 'svd' component of the lda object

`svd` values are the singular values which give the ratio of the between- and within-group standard deviations of the linear discriminant variables

The singular values are analogous to the eigenvalues of PCA, except that while PCA maximizes the variance of a component, LDA instead maximizes the separability (defined by the between and within-group standard deviation of a discriminator)

97.2% of the between-class variance is explained by the first linear discriminant function (LD1)

```
# svd values
lda.model$svd
```

```
## [1] 4.4682743 0.7545329
```

```
#proportion of trace
round(lda.model$svd^2 / sum(lda.model$svd^2),4)
```

```
## [1] 0.9723 0.0277
```

Predict sample classes for the test (hold-out) set using `predict` function applied to the LDA model using the first two variables the model was trained on

```
lda.prediction <- predict(lda.model,test.set[,1:2])
lda.prediction
```

```
## $class
## [1] Adeno  SCLC   SCLC   SCLC   SCLC   Adeno  Adeno  Normal
## Levels: Adeno Normal SCLC
##
## $posterior
##               Adeno      Normal        SCLC
## Adeno7  0.90942927 0.087314447 0.003256278
## Adeno8  0.05508919 0.007068463 0.937842348
## Adeno9  0.00731741 0.000153027 0.992529563
## Adeno10 0.35168988 0.063475044 0.584835076
## SCLC8   0.02483051 0.002258416 0.972911070
## SCLC9   0.78218952 0.137647762 0.080162721
## Normal4 0.50707745 0.491268120 0.001654431
## Normal5 0.29641114 0.703487621 0.000101237
##
## $x
##                 LD1         LD2
## Adeno7  -1.4459175  2.08708533
## Adeno8   1.1546705 -0.20363566
## Adeno9   2.0397072  1.57112039
## Adeno10  0.3849516 -0.09590165
## SCLC8    1.4596490  0.03265216
## SCLC9   -0.4801682  0.57109164
## Normal4 -1.7784897 -0.75645909
## Normal5 -2.5994979 -1.35202755
```

`lda.prediction$class` - the predicted class based on maximum posterior probability (MAP) `lda.prediction$posterior` - posterior probabilities for the classes `lda.prediction$x` - LDA test scores (the transformed values in LDA space) for all observations

```
cbind(round(lda.prediction$posterior, 4), class=lda.prediction$class)
```

```
##          Adeno Normal   SCLC class
## Adeno7  0.9094 0.0873 0.0033     1
## Adeno8  0.0551 0.0071 0.9378     3
## Adeno9  0.0073 0.0002 0.9925     3
## Adeno10 0.3517 0.0635 0.5848     3
## SCLC8   0.0248 0.0023 0.9729     3
## SCLC9   0.7822 0.1376 0.0802     1
## Normal4 0.5071 0.4913 0.0017     1
## Normal5 0.2964 0.7035 0.0001     2
```

Confusion matrix (prediction error)

How many total samples were misclassified?

```
dat.pred <- lda.prediction$class
conf.matrix <- table(dat.pred, sample.class)
conf.matrix
```

```
##          sample.class
## dat.pred Adeno Normal SCLC
##   Adeno      1      1    1
##   Normal     0      1    0
##   SCLC       3      0    1
```

So, 3 Adenos were wrongly predicted as SCLC, 1 SCLC was wrongly predicted as Adeno and 1 Normal was wrongly predicted as Adeno

```
# add up correct classifications and incorrect ones
sum(sample.class=="Adeno")      # total number of Adeno -> 4
```

```
## [1] 4
```

```
sum(dat.pred[sample.class=="Adeno"]=="Adeno")   # number of correct Adeno classifications -> 1
```

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

```
sum(dat.pred[sample.class=="Adeno"]=="SCLC")    # number of Adeno misclassified for SCLC -> 3
```

```
## [1] 3
```

```
sum(dat.pred[sample.class=="Adeno"]=="Normal")  # number of Adeno misclassified for Normal -> 0
```

```
## [1] 0
```

```
sum(sample.class=="SCLC")       # total number of SCLC -> 2
```

```
## [1] 2
```

```
sum(dat.pred[sample.class=="SCLC"]=="SCLC") # number of correct SCLC classifications -> 1
```

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

```
sum(dat.pred[sample.class=="SCLC"]=="Adeno")    # number of SCLC misclassified for Adeno -> 1
```

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

```
sum(dat.pred[sample.class=="SCLC"]=="Normal")   # number of SCLC misclassified for Normal -> 0
```

```
## [1] 0
```

```
sum(sample.class=="Normal")     # total number of Normal -> 2
```

```
## [1] 2
```

```
sum(dat.pred[sample.class=="Normal"]=="Normal") # number of correct Normal classifications -> 1
```

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

```
sum(dat.pred[sample.class=="Normal"]=="Adeno")  # number of Normal misclassified for Adeno -> 1
```

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

```
sum(dat.pred[sample.class=="Normal"]=="SCLC")   # number of Normal misclassified for SCLC -> 0
```

```
## [1] 0
```

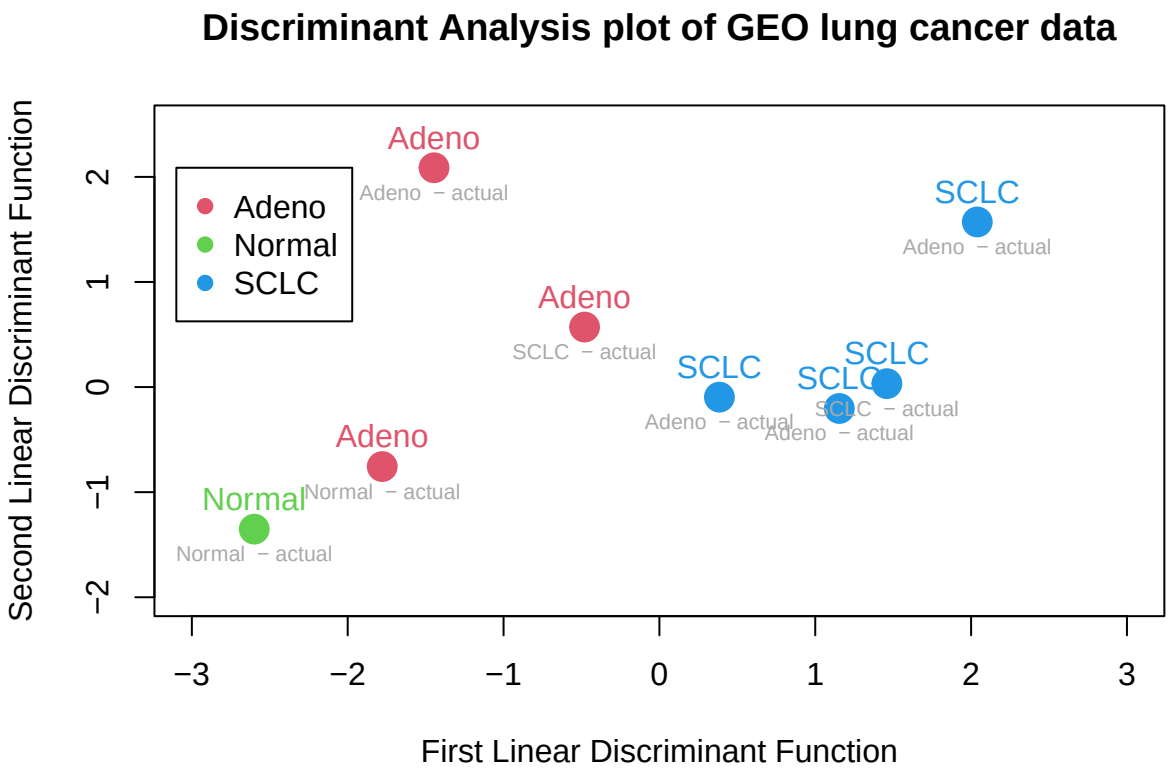
Misclassification rate

```
err = (length(sample.class) - sum(diag(conf.matrix)))/length(sample.class)
err
```

```
## [1] 0.625
```

Plot the first two discriminant functions in the lda.prediction$x versus each other

```
col <- as.numeric(lda.prediction$class) +1
plot(lda.prediction$x, col = col,
     xlab='First Linear Discriminant Function',
     ylab='Second Linear Discriminant Function',
     main='Discriminant Analysis plot of GEO lung cancer data',
     cex=2, pch =19, xlim=c(-3,3), ylim = c(-2.0, 2.5))
legend(min(range(lda.prediction$x[,1]))-0.5,max(range(lda.prediction$x[,2])),
       levels(lda.prediction$class),col=sort(unique(col)),pch=19,cex=1)
text(lda.prediction$x,col= col,cex=1, labels= lda.prediction$class, pos=3)
text(lda.prediction$x,col= 'dark gray',cex=0.7, labels= paste(sample.class, ' - actual'), pos=1)
```



Same LDA analysis but this time **using all of the features/genes** in the data matrix as opposed to the first two

```
# fit model
lda.model <- lda(clas.train~., training.set[,2:ncol(training.set)])
#proportion of trace
round(lda.model$svd^2 / sum(lda.model$svd^2),4)
```

```
## [1] 0.8806 0.1194
```

```
# predict on the entire test set
lda.prediction <- predict(lda.model,test.set)
lda.prediction
```

```
## $class
## [1] Adeno  Adeno  Adeno  Adeno  SCLC   SCLC   Normal Normal
## Levels: Adeno Normal SCLC
##
## $posterior
##                Adeno       Normal        SCLC
## Adeno7  0.783222108 0.1997731171 0.017004775
## Adeno8  0.926151499 0.0456882545 0.028160247
## Adeno9  0.807927848 0.0795699016 0.112502250
## Adeno10 0.788661204 0.1860731163 0.025265679
## SCLC8   0.003886686 0.0008458244 0.995267490
```

```
## SCLC9   0.017070842 0.0159140199 0.967015138
## Normal4 0.112188699 0.8759834479 0.011827853
## Normal5 0.101683932 0.8932203906 0.005095678
##
## $x
##                LD1          LD2
## Adeno7  -1.0570025  0.111979168
## Adeno8  -0.7738859  1.076233820
## Adeno9  -0.3785170  0.633662668
## Adeno10 -0.9281766  0.149479622
## SCLC8    1.8602634 -0.001610203
## SCLC9    1.2351727 -0.828661810
## Normal4 -0.9408529 -1.946533650
## Normal5 -1.1837386 -1.999744172
```

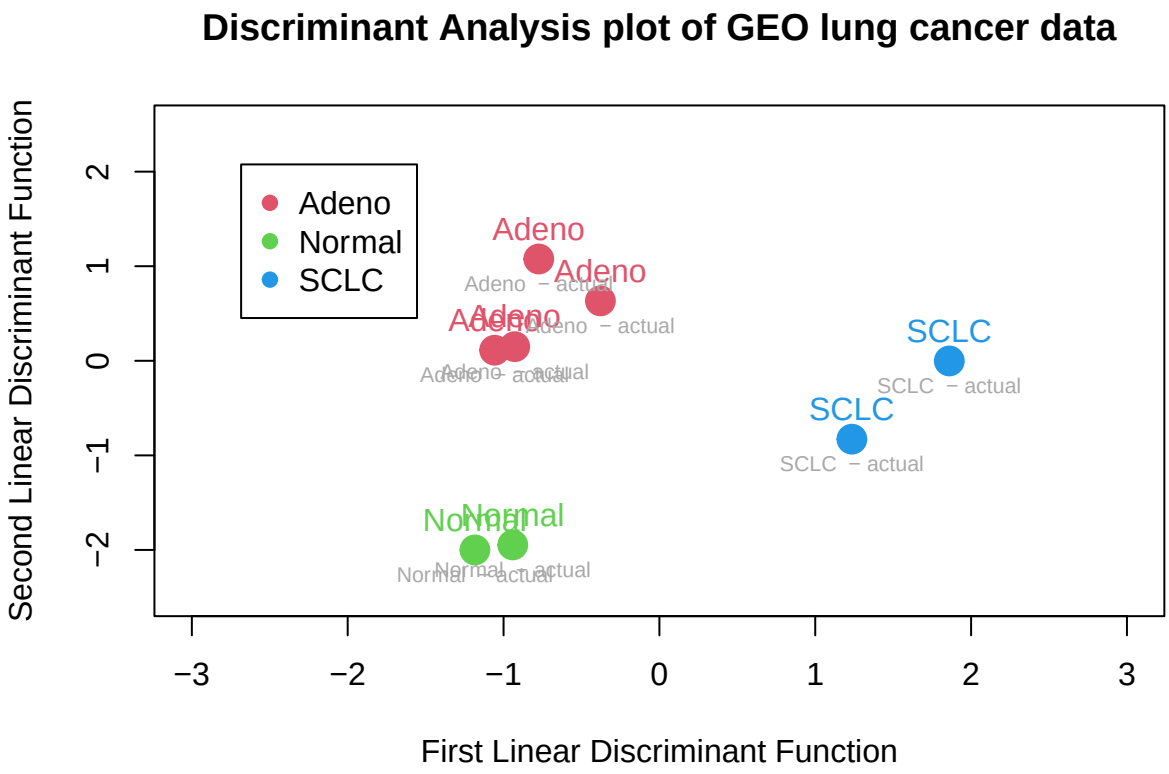
```
#confusion matrix
dat.pred <- lda.prediction$class
table(dat.pred, sample.class)
```

```
##         sample.class
## dat.pred Adeno Normal SCLC
##   Adeno      4      0    0
##   Normal     0      2    0
##   SCLC       0      0    2
```

No samples are misclassified

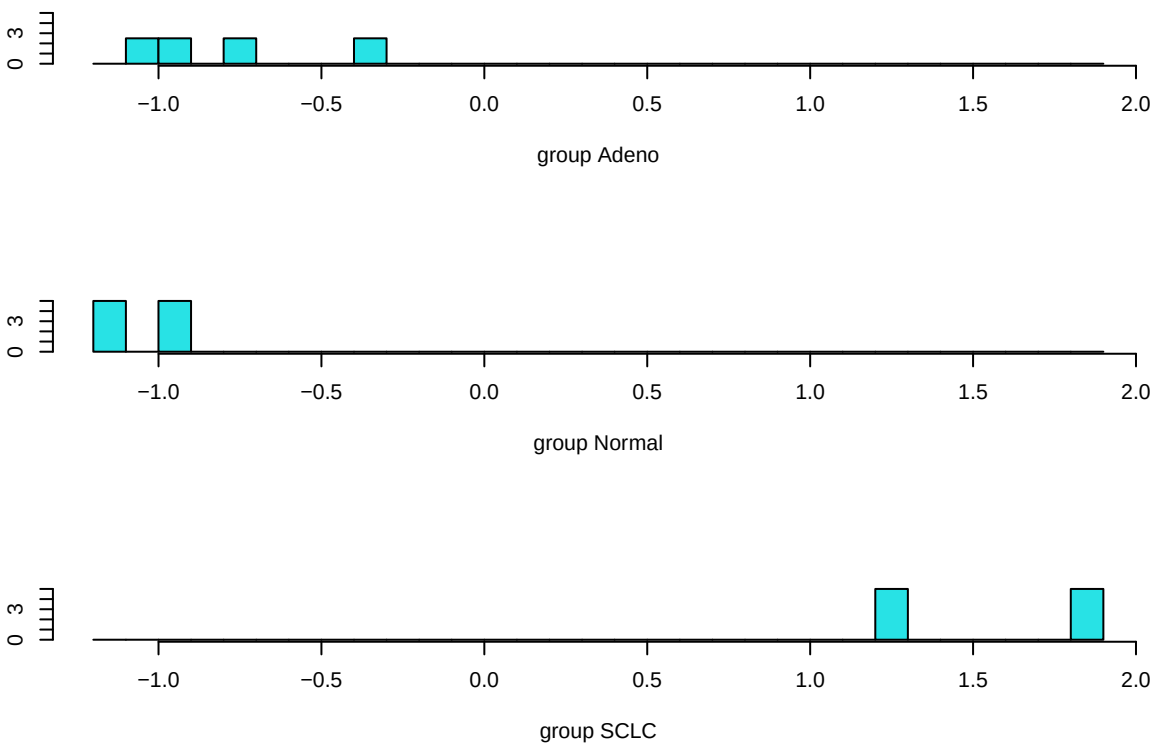
Plot the first two discriminant functions in the lda.prediction$x versus each other (scatterplot)

```
col <- as.numeric(lda.prediction$class) +1
plot(lda.prediction$x, col = col,
     xlab='First Linear Discriminant Function',
     ylab='Second Linear Discriminant Function',
     main='Discriminant Analysis plot of GEO lung cancer data',
     cex=2, pch =19, xlim=c(-3,3), ylim = c(-2.5, 2.5))
legend(min(range(lda.prediction$x[,1]))-1.5, max(range(lda.prediction$x[,2]) +1 ),
       levels(lda.prediction$class),col=sort(unique(col)),pch=19,cex=1)
text(lda.prediction$x,col= col,cex=1, labels= lda.prediction$class, pos=3)
text(lda.prediction$x,col= 'dark gray',cex=0.7, labels= paste(sample.class, ' - actual'), pos=1)
```

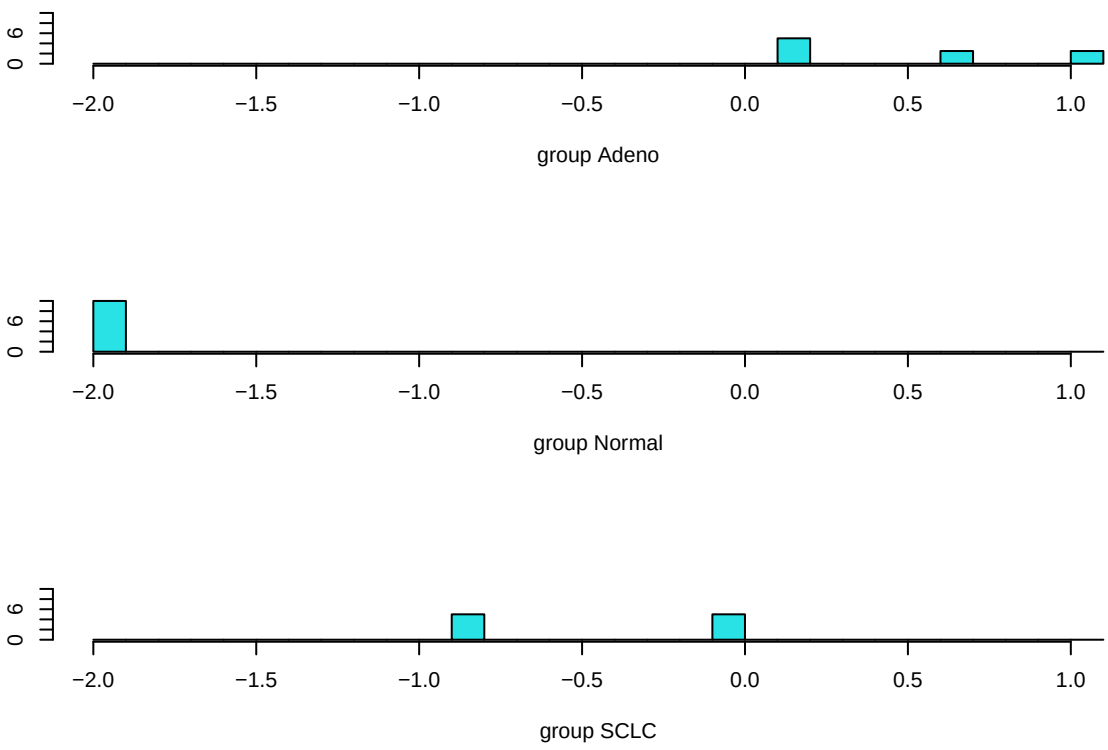


Stacked histogram of the LDA scores of the discriminant functions for each group

```
ldahist(data = lda.prediction$x[,1], g=lda.prediction$class)
```




```
ldahist(data = lda.prediction$x[,2], g=lda.prediction$class)
```



Support Vector Machines (SVM)

Supervised discriminant classifier focused on modeling of the optimal decision boundary between two classes

Discriminates by finding the maximal margin separating the classes (maximum separation parallel hyperplanes for each class)

Linear SVM classifier partitions data into classes in a high-dimensional feature space that is nonlinearly related to the input space

Nonlinear SVM partitioning projects the data into a feature space using a kernel function

```
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")

BiocManager::install("colonCA")
```

Load Alon colon cancer dataset

```
library(colonCA)
data(colonCA)
d <- exprs(colonCA)
clas <- factor(colonCA$class)
print('Total Tumor samples')
```

```
## [1] "Total Tumor samples"
```

```
sum(clas == 't')
```

```
## [1] 40
```

```
print('Total Normal samples')
```

```
## [1] "Total Normal samples"
```

```
sum(clas == 'n')
```

```
## [1] 22
```

```
#clas <- data.frame(as.numeric(colonCA$class))
head(clas, 5)
```

```
## [1] t n t n t
## Levels: n t
```

PCA on colon cancer data and use second and third principal components for classification analysis

```
dat <- scale(t(d),center=T,scale=T)
dat.pca <- prcomp(dat)
dat.loadings <- dat.pca$x[,2:3]
dat.loadings
```

```
##            PC2         PC3
## 1    6.6216927  -6.2756949
## 2   17.5046007  -1.0361004
## 3   12.1753047  -3.6317148
## 4   18.9517262   1.6922475
## 5   -6.5546870  -3.6499939
## 6   -7.4490650  -0.8266418
## 7    2.0499766 -12.7825746
## 8   -3.5233418  -0.8891102
## 9    0.9417999  -9.4859025
## 10   0.5816125  -6.7793192
## 11 -28.6596144 -44.7405547
## 12  14.8488151  -2.0308971
## 13  -1.7741966  -3.1066770
## 14   0.9362762   0.9473657
## 15   0.7852562   5.0252215
## 16   2.4709977   1.6639086
## 17  -9.2186547  -4.8303981
## 18  -8.3584202  -2.5160532
## 19  -7.0767509  -2.8309932
## 20 -10.9735694  -4.4298402
## 21   8.0048617 -10.4294867
## 22  18.2947254  -3.8693846
## 23 -15.5598536   0.5028694
## 24  -5.8783080  -2.8417244
```

```
## 25  -4.3098296   2.6779664
## 26   2.8035811  -9.4114776
## 27   3.0568376  -7.4593840
## 28  14.1835183  -5.9738094
## 29  30.9007018 -28.4370424
## 30   6.5611924 -22.1560878
## 31  25.7617050   3.7616074
## 32 -13.5121574  -7.5163519
## 33 -14.7172343  -7.4413440
## 34   2.7784779  -0.8213737
## 35   0.2661368  -3.8637284
## 36 -18.1432758   1.6404118
## 37  23.1860874 -24.4287309
## 38 -17.2814867  -0.6229471
## 39   5.4344021  13.9132785
## 40  -1.7418234   0.3789496
## 41   8.4887979   6.7067368
## 42  -3.6844197  13.9593111
## 43  -1.0839855  19.1252662
## 44 -34.3769568  -5.9964280
## 45  27.2832350  14.5215793
## 46 -35.7290477   7.4357525
## 47 -26.4449907  14.7117734
## 48  14.6265396  10.8312664
## 49   3.8609730   1.8341399
## 50   4.0877949  17.5423495
## 51  -1.8911714   4.3197824
## 52 -24.2281964  18.2731793
## 53   2.5026999   9.3435411
## 54  11.9292106  24.7311252
## 55  -3.2391063   2.1035992
## 56  -6.3633162  11.5358750
## 57  15.3066321  15.7365722
## 58   3.0475261  -0.7263339
## 59  -5.1530011   8.0311351
## 60  12.8375135   5.5596087
## 61  -4.3450327   1.5150908
## 62  -1.7997152  11.8165900
```

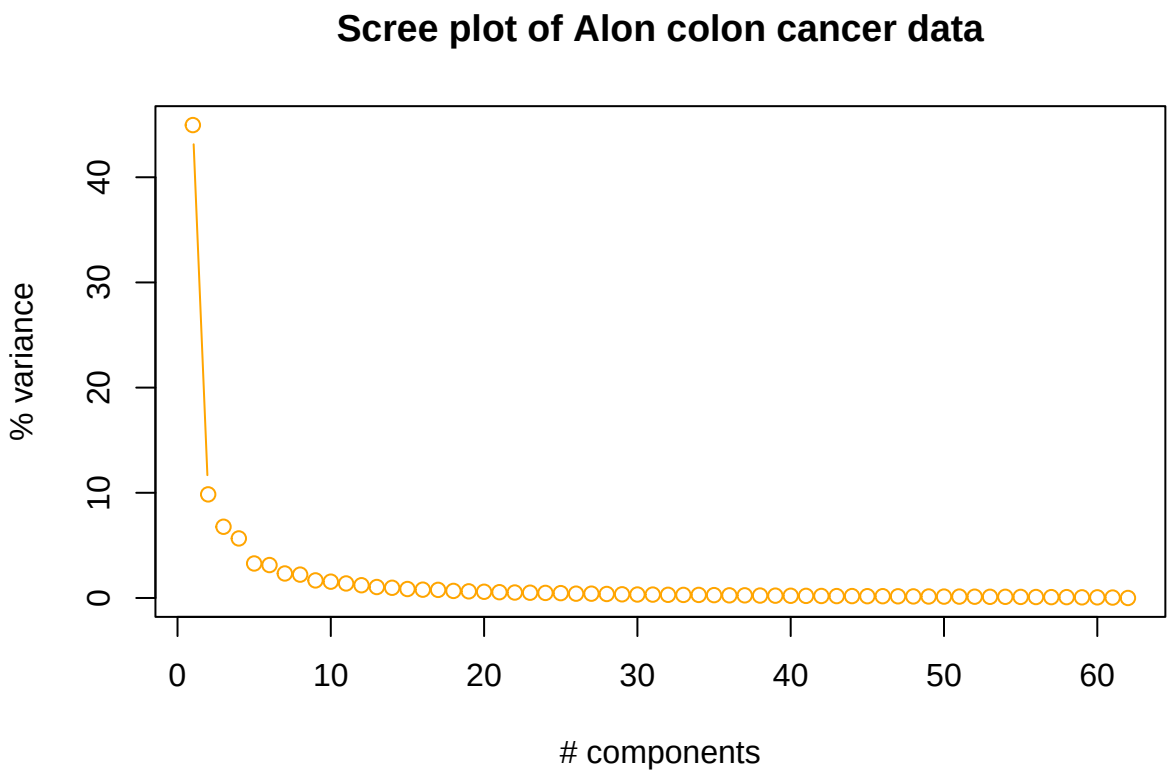
```
# percent variability of the principal components
print("Percent variability of the principal components")
```

```
## [1] "Percent variability of the principal components"
```

```
dat.pca.var <- round(dat.pca$sdev^2 / sum(dat.pca$sdev^2)*100,2)
dat.pca.var
```

```
##  [1] 44.96  9.85  6.77  5.66  3.28  3.13  2.33  2.22  1.67  1.55  1.38  1.21
## [13]  1.04  0.97  0.85  0.79  0.77  0.67  0.63  0.59  0.55  0.52  0.50  0.48
## [25]  0.46  0.41  0.41  0.38  0.35  0.33  0.32  0.30  0.29  0.29  0.27  0.25
## [37]  0.25  0.23  0.22  0.21  0.20  0.19  0.18  0.18  0.17  0.17  0.16  0.15
## [49]  0.14  0.13  0.13  0.12  0.11  0.11  0.10  0.09  0.08  0.08  0.06  0.06
## [61]  0.04  0.00
```

```
# scree plot for the PCA
plot(c(1:length(dat.pca.var)),dat.pca.var,type='b',xlab='# components',ylab='% variance',
     main='Scree plot of Alon colon cancer data', col='orange')
```



Fit a kernel SVM of type C classification

C - tunable regularization parameter for the cost of constraints violation (penalty for misclassification)

kernel - Gaussian Radial Basis Function (RBF) kernel function used in training and predicting (exponetial function of the negative of the ratio between the squared Euclidean distance between two feature vectors and 2*sigma^2)

sigma - tunable regularization hyperparameter representing the inverse width of the Gaussian RBF kernel

gamma - inverse of 2*sigma^2

Increasing gamma, C or both, decreases regularization of an SVM classifier trained with Gaussian RBF that underfits (too much regularization) the (training) data Reciprocally, when the model overfits (not enough regularization) increase regularization by decreasing gamma, C or both.

```
#install.packages('kernlab')

library(kernlab)

svp <- ksvm(dat.loadings,clas,type="C-svc", kernel='rbfdot', kpar='automatic', C=1, fit=T)
svp
```

```
## Support Vector Machine object of class "ksvm"
```

```
##
## SV type: C-svc  (classification)
##  parameter : cost C = 1
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  8.11286035734384
##
## Number of Support Vectors : 57
##
## Objective Function Value : -31.7605
## Training error : 0.145161
```

Get fitted values

```
fit <- fitted(svp)
fit
```

```
##  [1] t n t n t n t t t t t n t t t t t t t t t n t t t t t t t t t t t t t t
## [39] n t t n n t t t t n t n t t t n t t t t t n t n
## Levels: n t
```

```
clas
```

```
##  [1] t n t n t n t n t n t n t n t n t n t n t n t n t t t t t t t t t t t t t t
## [39] n t t n n t t t t t n t n n t t n n t t t t t n t n
## Levels: n t
```

Error rates

```
er1 <- sum(fit[clas=='n']=='t')     # number of incorrect normal classifications
er1
```

```
## [1] 9
```

```
er2 <- sum(fit[clas=='t']=='n')   # number of incorrect tumor classifications
er2
```

```
## [1] 0
```

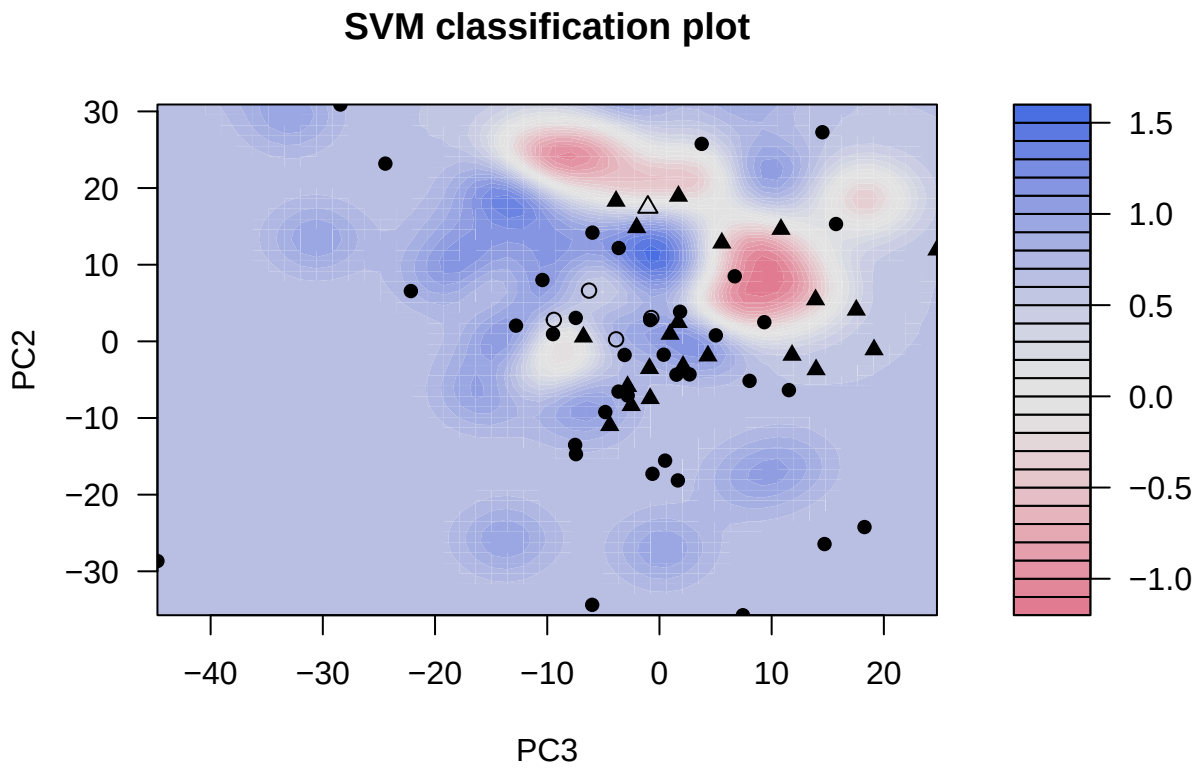
```
table(clas, fit)
```

```
##     fit
## clas  n  t
##    n 13  9
##    t  0 40
```

Plot the kernel SVM binary classification

The plot function for `ksvm` objects displays a contour plot of the decision values with the corresponding support vectors highlighted (filled shapes)

```
plot(svp, data=dat.loadings)
```



Find optimal cost parameter and gamma hyperparameter for classification using `tune` function

`tune` - generic function that tunes hyperparameters of statistical methods using a grid search over supplied parameter ranges

```
## tune `svm' for classification with RBF-kernel (default in svm),
## using default 10-fold cross validation ( 10 partitions for cross-validation)

#install.packages('e1071')

library(e1071)

# show best model
grid.search.tune.svm <- tune(svm, dat.loadings, clas, nrepeat=1,
                             ranges = list(gamma = c(0.1, 0.5, 1, 2, 3, 4),
                                           cost = c(0.1, 0.5, 1, 5, 10)),
                             tunecontrol = tune.control(sampling = "cross",cross=10))
grid.search.tune.svm
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
```

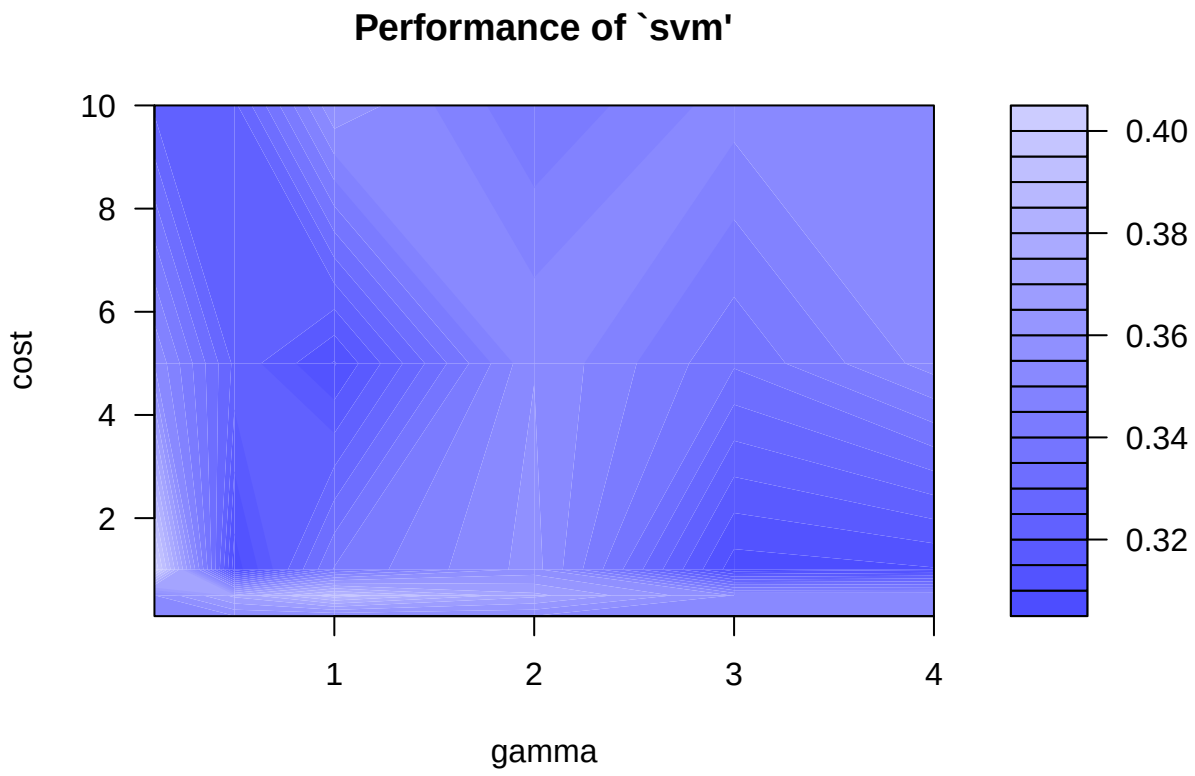
```
## - best parameters:
##  gamma cost
##    0.5    1
##
## - best performance: 0.3071429
```

```
# show best model
grid.search.tune.svm$best.model
```

```
##
## Call:
## best.tune(method = svm, train.x = dat.loadings, train.y = clas, ranges = list(gamma = c(0.1,
##     0.5, 1, 2, 3, 4), cost = c(0.1, 0.5, 1, 5, 10)), tunecontrol = tune.control(sampling = "cross",
##     cross = 10), nrepeat = 1)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  49
```

```
#alternatively tune.svm function can be used
```

```
#summary(grid.search.tune.svm)
plot(grid.search.tune.svm)
```



Classification and Regression Trees (CART)

A method of using independent variables to predict either categorical (classification) or continuous (regression) dependent variables

Using a series of binary decisions (if-then commands), the predictor variables are utilized to partition a classification for each dependent variable at various levels of a hierarchy

The nodes on each level of the tree give the criteria used for each decision

The consistency of classification at a particular node (the accuracy of splitting tree) is measured by the Gini Impurity Index which:

- calculates the amount of probability of a specific feature (predictor variable) classified incorrectly when selected randomly
- ranges from 0 to 1, with 0 - pure classification (one class exists); 1 - random distribution across classes

Binary decision tree classification

Use `tree` function to fit a classification tree

The tree is grown by binary partitioning using the response in the specified formula and choosing splits from the terms of the right hand side The split which maximizes the reduction in impurity is chosen in each iteration

Use first 10 principal components from PCA on Alon colon cancer dataset as predictor variables

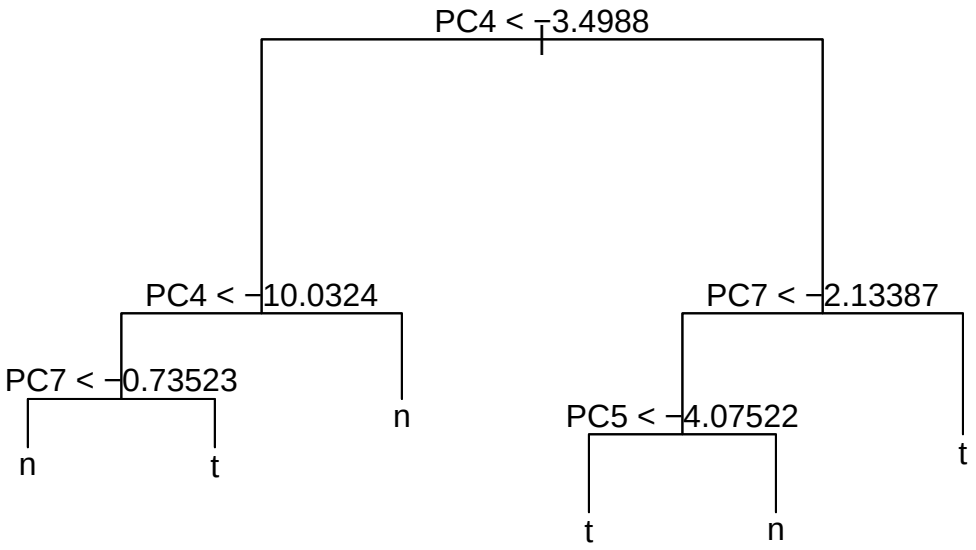
```r
#install.packages("tree")

library(tree)

dat.loadings <- dat.pca$x[,1:10]
dat <- as.data.frame(dat.loadings)
dat.c <- data.frame(clas,dat)   # bind class labels to dataframe
head(dat.c, 5)
```

```
##   clas       PC1       PC2       PC3       PC4       PC5       PC6       PC7
## 1    t -13.18720  6.621693 -6.275695  2.578185 -6.346293 7.0423444 -4.944281
## 2    n   5.72884 17.504601 -1.036100 -4.966108 -8.829636 4.6346695 -5.610497
## 3    t -29.81798 12.175305 -3.631715 -1.622651  1.674495 6.2227084  7.611522
## 4    n -19.13399 18.951726  1.692248  5.332339 -3.916954 7.4698325 -2.464240
## 5    t -24.01947 -6.554687 -3.649994 -1.281030  9.179632 0.3175728  1.012784
##         PC8        PC9       PC10
## 1  1.188046 -1.3367609  6.7352776
## 2  8.613667 -2.8933694 -0.6481336
## 3 -2.620587  5.8061542 12.7191135
## 4  5.098892  3.5912608  8.2766827
## 5 -1.482653 -0.7186981 -1.6959527
```

```r
dat.train.tree<-tree(clas~.,data=dat.c)
summary(dat.train.tree) # classification results
```

```
##
## Classification tree:
## tree(formula = clas ~ ., data = dat.c)
## Variables actually used in tree construction:
## [1] "PC4" "PC7" "PC5"
## Number of terminal nodes:  6
## Residual mean deviance:  0.3749 = 21 / 56
## Misclassification error rate: 0.08065 = 5 / 62
```

```r
plot(dat.train.tree)
text(dat.train.tree)
```




```
# PC4 is the primary bifurcation at values < and > -3.4988
# PC4 and PC7 are the secondary bifurcations at values < and > -10.0324 and -2.13387, respectively
# and so on
```

```
dat.train.tree
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 62 80.650 t ( 0.3548 0.6452 )
##    2) PC4 < -3.4988 22 23.580 n ( 0.7727 0.2273 )
##      4) PC4 < -10.0324 11 15.160 n ( 0.5455 0.4545 )
##        8) PC7 < -0.73523 6  5.407 n ( 0.8333 0.1667 ) *
##        9) PC7 > -0.73523 5  5.004 t ( 0.2000 0.8000 ) *
##      5) PC4 > -10.0324 11  0.000 n ( 1.0000 0.0000 ) *
##    3) PC4 > -3.4988 40 30.140 t ( 0.1250 0.8750 )
##      6) PC7 < -2.13387 14 18.250 t ( 0.3571 0.6429 )
##       12) PC5 < -4.07522 6  0.000 t ( 0.0000 1.0000 ) *
##       13) PC5 > -4.07522 8 10.590 n ( 0.6250 0.3750 ) *
##      7) PC7 > -2.13387 26  0.000 t ( 0.0000 1.0000 ) *
```

k-nearest neighbors (KNN)

Classifies a point (gene/sample) by calculating the distances between all points and assigns the point to the class that is most common among its k-nearest neighbors

Spatial relationships between points in n-dimensional space determine class membership

The larger k that is specified, the larger the error rate

Unsupervised classification method - Class membership is utilized only to count votes from neighboring points, not to optimize the classifier

```
dat.k <- exprs(colonCA)      # matrix of expression values
clas <- as.character(colonCA$class)
```

Run KNN classifier with increasing number of k

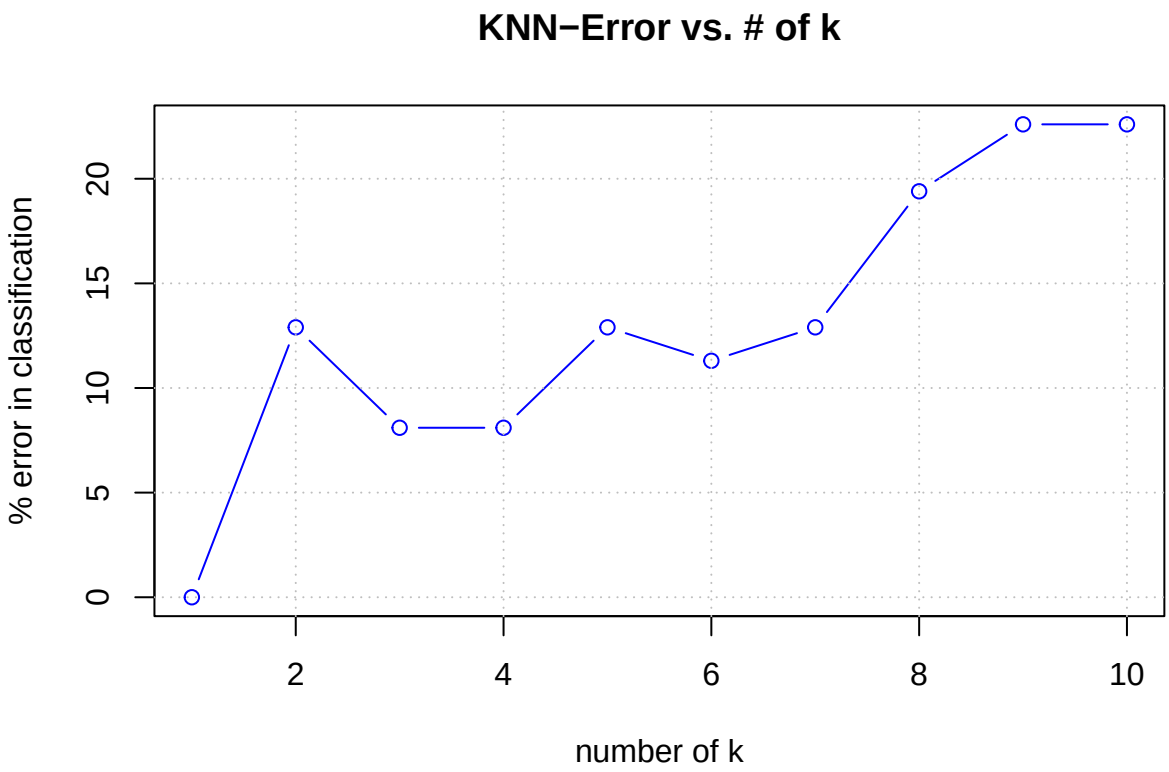
knn(train, test, cl, k = 1, l = 0, prob = FALSE, use.all = TRUE)

train, test - matrices or data frames of the train set, test set cases cl - factor of true classifications of training set k - number of neighbors considered prob - if TRUE, the proportion of the votes for the winning class are returned as `prob` attribute

```
library(class)

error.list <- NULL

for (i in 1:10) {
    dat.knn <- knn(t(dat.k),t(dat.k),clas,k=i,prob=T)

    # error rates
    er1 <- sum(dat.knn[clas=="n"]=="t")      # number of incorrect normal classifications
    er2 <- sum(dat.knn[clas=="t"]=="n")      # number of incorrect tumor classifications
    er.total <- sum(er1,er2)/ncol(dat.k)
    er.total <- round(er.total*100,1)

    # store in list
    error.list <- c(error.list,er.total)
}
```

Plot classification error vs. k in KNN classifier

```
plot(c(1:10),error.list,type="b",col="blue",xlab="number of k",ylab="% error in classification",
     main="KNN-Error vs. # of k")
grid(col="grey")
```



Receiver operator curves (ROC)

Used to characterize the sensitivity/specificity tradeoffs for a binary classifier

Run KNN classifier with increasing number of k with 10 genes for ROC curve

```
tp.list <- NULL
fp.list <- NULL
k.list <- c(1:35)

for (i in 2:length(k.list)) {
    dat.knn <- knn(t(dat.k[1:10,]), t(dat.k[1:10,]), clas, k=k.list[i], prob=T)

    # error rates
    # true positive rate (TPR/sensitivity)
    er1 <- sum(dat.knn[clas=="t"]=="t")/sum(clas=="t")*100
    # false positive rate (FPR/1-specificity)
    # specificity is true negative rate (TNR)
    er2 <- sum(dat.knn[clas=="n"]=="t")/sum(clas=="n")*100
    tp.list <- c(tp.list,er1)
    fp.list <- c(fp.list,er2)
}
```

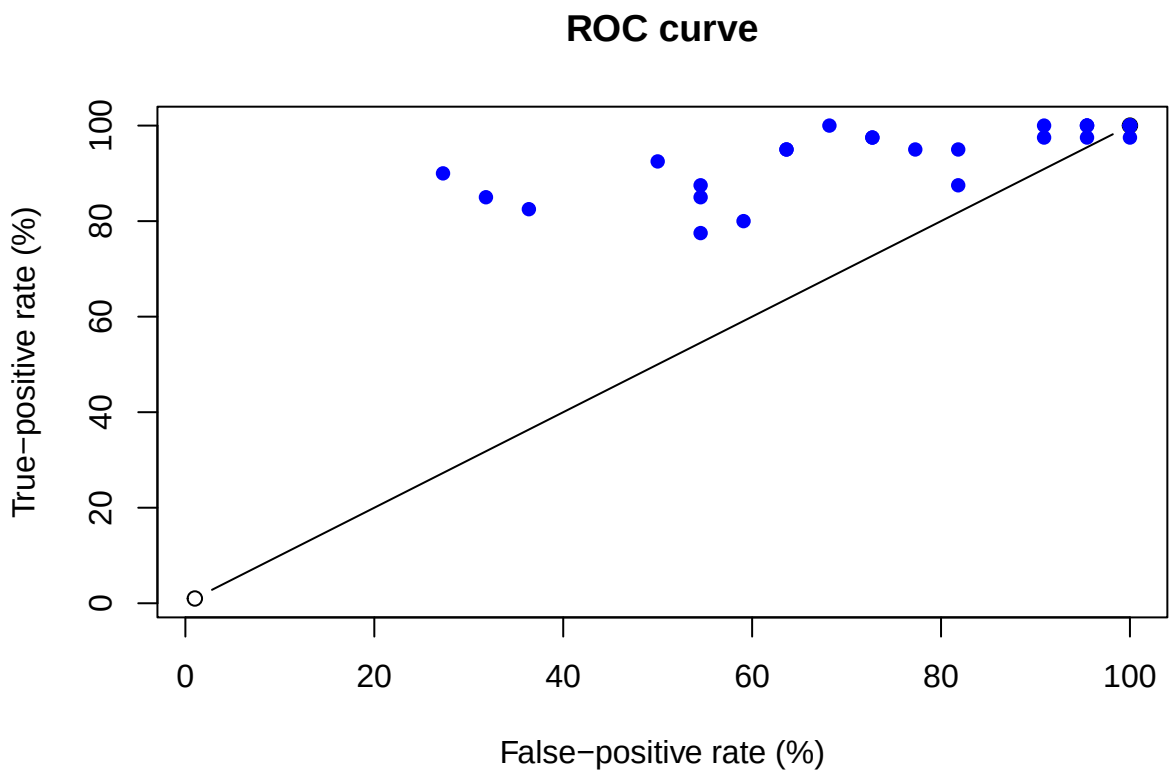
Plot ROC curve of TPR (sensitivity) vs FPR (1-specificity)

Sensitivity: probability of a positive test results, given the presence of the condition (true positive rate)

Specificity: probability of a negative test result given the absence of the condition (true negative rate)

Each point on the plot represents a model accuracy rate

```
plot(c(1,100),c(1,100),type="b",xlab="False-positive rate (%)",ylab="True-positive rate (%)",
     main="ROC curve")
points(fp.list,tp.list,cex=1.0,col="blue",pch=16)
```



Artificial Neural Networks (ANN)

An ANN, initially inspired by neural networks in the brain, consists of layers of interconnected compute units (neurons).

The network receives data in an input layer, the data is then transformed in a non-linear way through a hidden layer, before final outputs are computed in the output layer.

Neurons in the hidden and output layers are connected to all neurons in previous layers, and are called fully connected or dense layers.

Each connection between neurons carries a weight.

Each neuron computes a weighted sum of its inputs and applies a non-linear activation function to calculate its output, using also a bias term (threshold) as a parameter.

The weights between neurons are free parameters that capture the model's representation of the data, are learned from input observations and adjusted during training.

Learning minimizes a loss (cost/error/objective) function that measures the fit between the predictions of the model parameterized by the connection weights and the actual observations.

So, goal of ANN model training (learning) is to find the parameters - the weights - that minimize the loss which measures the fit of the model output to the true label of an observation.

Most ANN are trained using maximum (conditional) likelihood estimation, so the most common loss function for classification is the negative log-likelihood, that is between the empirical distribution defined by the training set and the probability distribution defined by the model

```
dat <- exprs(colonCA)
clas <- factor(colonCA$class)

# define training samples by random
samp <- sample(c(1:ncol(dat)),30,replace=FALSE)
samp
```

```
##  [1] 16 52 46 53 36  4 25 21 17 56 44 30 58  1  3 40 26  5 48  6 24 10 51 43 27
## [26] 57  8 38 31 14
```

```
dat.n <- data.frame(dat.loadings,clas)
dat.n
```

```
##           PC1         PC2         PC3          PC4         PC5         PC6
## 1  -13.187201   6.6216927  -6.2756949   2.57818474  -6.3462928   7.0423444
## 2    5.728840  17.5046007  -1.0361004  -4.96610829  -8.8296364   4.6346695
## 3  -29.817982  12.1753047  -3.6317148  -1.62265107   1.6744949   6.2227084
## 4  -19.133988  18.9517262   1.6922475   5.33233899  -3.9169541   7.4698325
## 5  -24.019470  -6.5546870  -3.6499939  -1.28103000   9.1796315   0.3175728
## 6  -34.683882  -7.4490650  -0.8266418  -5.76942370   9.7801211   4.2894765
## 7  -17.408072   2.0499766 -12.7825746   0.13875800  -2.6449392   5.5352968
## 8  -18.155241  -3.5233418  -0.8891102  -4.36691245   8.2138300  11.7091410
## 9    7.271389   0.9417999  -9.4859025  28.60559302   1.2862233  20.5677178
## 10   3.096873   0.5816125  -6.7793192 -14.16676307  20.8753506   6.2187817
## 11  63.656944 -28.6596144 -44.7405547 -23.21061867   6.5614097   8.8609186
## 12  17.938588  14.8488151  -2.0308971  -5.37908398  -8.1204862  19.7655453
## 13 -14.947898  -1.7741966  -3.1066770   0.06049827   2.7065614   2.1085697
## 14 -26.523426   0.9362762   0.9473657  -2.21725664   1.7485541   4.9386102
## 15  -3.351608   0.7852562   5.0252215   7.22213221   8.6591130  10.3939057
## 16 -29.195926   2.4709977   1.6639086   9.57339298   1.4748873   0.9880057
## 17 -37.038359  -9.2186547  -4.8303981  -1.60441576  -4.7204188   2.0288658
## 18 -37.539710  -8.3584202  -2.5160532  -6.89074750  -2.3893150   2.7959941
## 19 -26.306095  -7.0767509  -2.8309932   6.24211663   1.0724032   5.4523249
## 20 -23.236586 -10.9735694  -4.4298402  -9.94942971  13.3296504   7.7362678
## 21   1.213897   8.0048617 -10.4294867   5.29367279  -6.6280505   1.8061087
## 22  14.716280  18.2947254  -3.8693846 -11.62313769   2.3092494   8.6559442
## 23 -31.684353 -15.5598536   0.5028694   2.93673598   6.9505760  -1.7550254
## 24 -53.318523  -5.8783080  -2.8417244  -6.70116815   0.9946149  -5.2415582
## 25   3.565699  -4.3098296   2.6779664  13.11288656   6.1134406   1.9086058
## 26 -19.407752   2.8035811  -9.4114776   1.83128069  -5.3726477 -12.2096119
## 27 -20.582157   3.0568376  -7.4593840  -1.83181721  -3.5846125 -10.5292746
## 28  22.367346  14.1835183  -5.9738094  22.77984170  -4.2334775  -6.9020999
## 29  51.070926  30.9007018 -28.4370424   9.13363552  -1.1787100 -12.8290932
## 30  38.898352   6.5611924 -22.1560878   5.35518055  -7.1852654  -5.3256641
## 31  38.112151  25.7617050   3.7616074  13.65072113   0.3147594   5.5509402
## 32 -16.870356 -13.5121574  -7.5163519 -10.70443542  -2.3821894  -5.2912013
```

```
## 33 -18.980133 -14.7172343  -7.4413440  -2.38671785  -0.6191901  -5.3628783
## 34  16.776156   2.7784779  -0.8213737  12.67692655   3.5126160  -3.3259714
## 35 -18.373304   0.2661368  -3.8637284   3.71295525 -10.0315578  -4.5275504
## 36 -19.208822 -18.1432758   1.6404118   6.42127277   0.3802876  -7.1329118
## 37  17.526368  23.1860874 -24.4287309   6.70517794  17.8697716  -3.3075763
## 38 -20.688100 -17.2814867  -0.6229471   7.86852125   3.1957661  -8.7762192
## 39 -16.138008   5.4344021  13.9132785  -3.60784683   5.5527351  -3.9009620
## 40  -2.804771  -1.7418234   0.3789496   4.28336268 -13.0889818  -4.0152791
## 41 -29.651378   8.4887979   6.7067368   7.84829883  -6.6151231  -8.4948092
## 42  14.479882  -3.6844197  13.9593111 -10.45866178   0.3406714   6.8682032
## 43  56.498853  -1.0839855  19.1252662 -22.30903042  -9.7604333   5.5797385
## 44  58.673359 -34.3769568  -5.9964280  -3.38974400 -18.3589208  -2.5338437
## 45  79.282660  27.2832350  14.5215793 -15.78177254   1.7809661  -5.1649576
## 46  66.464327 -35.7290477   7.4357525   4.29842633  -8.9201347   1.9246279
## 47  65.182673 -26.4449907  14.7117734  18.00727047  27.0998749 -18.2165459
## 48  25.774573  14.6265396  10.8312664 -16.73827508  12.1932799 -16.0009557
## 49 -11.046772   3.8609730   1.8341399 -17.48492920  -5.9358262  -5.2941173
## 50   7.013021   4.0877949  17.5423495  -9.07418376  -2.3983738   8.2587268
## 51  -2.397413  -1.8911714   4.3197824   0.22392903  -2.8107528  -2.4851306
## 52  40.155567 -24.2281964  18.2731793  21.07777806  -5.4885789  13.6523205
## 53  -9.096281   2.5026999   9.3435411  10.53400556  -3.8101139  -2.7621426
## 54  12.803734  11.9292106  24.7311252   3.06873520   7.1483404   3.8764732
## 55  -7.790132  -3.2391063   2.1035992  -8.14782165  -3.2037760  -7.5538519
## 56 -13.311461  -6.3633162  11.5358750 -10.07605411   2.3357158  -3.7193250
## 57 -10.869634  15.3066321  15.7365722   3.54453125   6.6228025   1.5155323
## 58 -33.737784   3.0475261  -0.7263339   2.58865807  -8.0783555  -7.7960313
## 59   1.174889  -5.1530011   8.0311351  11.65605113  -6.8508793  -3.2277926
## 60   1.077665  12.8375135   5.5596087 -15.61678912  -6.7358125 -10.1064708
## 61 -15.919196  -4.3450327   1.5150908  -1.01736657  -7.7537497  -7.3490517
## 62  -4.099239  -1.7997152  11.8165900  -9.98867788  -3.2841421   2.4641325
##             PC7         PC8          PC9        PC10 clas
## 1   -4.94428073   1.1880458  -1.33676087   6.7352776    t
## 2   -5.61049711   8.6136670  -2.89336938  -0.6481336    n
## 3    7.61152234  -2.6205868   5.80615424  12.7191135    t
## 4   -2.46423979   5.0988917   3.59126082   8.2766827    n
## 5    1.01278364  -1.4826534  -0.71869811  -1.6959527    t
## 6    0.97043404   1.9042032   0.58829074  -3.0112276    n
## 7   -0.01530982   3.1198179   2.84496572   6.5985945    t
## 8   -1.36489386   1.3481473  10.39396192   3.9655657    n
## 9    7.78266636  20.7327988 -11.78208318  -8.9021724    t
## 10  -4.57513595  -2.1542091   1.60948289  -6.5588319    n
## 11   2.56213602 -15.0238344 -15.69305503   1.4060010    t
## 12   6.15084789   9.8349043  -4.05695086  -3.2571043    n
## 13  -4.71394271  -3.9500963   4.28993897   1.9237316    t
## 14  -7.59565922   0.2759560   6.50384032   0.7167675    n
## 15  -5.10756485  -5.1079466   7.96920165   6.3805198    t
## 16  -8.86697199   4.5530971   2.38111703  -3.5262520    n
## 17   1.52826416  -0.8054771   3.05737067  -2.1678560    t
## 18  -4.01968191   4.7923292   3.36772181  -1.5359174    n
## 19   0.57088168   4.7052970  -1.89130657  -1.0411292    t
## 20  -1.76552360   1.7414860   1.63351171  -4.5951668    n
## 21  -9.64684028   3.6266393   4.05769895   7.0723718    t
## 22 -12.25205297  -3.4175473   6.67076234   7.9908193    n
## 23  -0.13776966  -3.3099308   3.43442357  -7.5968673    t
```

```
## 24  -0.97274455   2.1300370   1.45133314  -1.0069760    n
## 25  -4.63013486  -0.6846195   2.38868559  -1.7094196    t
## 26   0.93261729   1.3232818  -1.15030632   0.3841638    t
## 27  -1.62597539  -0.1852020   1.04744623  -2.3850319    t
## 28  -7.64404544  -1.6641701  -5.79293829  -3.3304857    t
## 29  -1.80350543  -1.6303953   0.04843364  -5.5491143    t
## 30  -2.91536218  -2.7321130  10.36754374  -4.0634554    t
## 31  20.93757819 -17.1067297  17.61007714 -11.8108093    t
## 32   7.30233123  -1.8806199   3.81284435  -7.0029868    t
## 33  -0.45106183  -0.3157333   0.74146239  -4.8585876    t
## 34   3.52242497  -4.0663349  -1.78982180  -0.3690188    t
## 35   0.64697375   2.4309067   1.96268596  -2.7893330    t
## 36   2.70077415   0.4338796   1.28920963  -5.9068403    t
## 37   8.08207180   7.1225742  -6.88282333   6.8438009    t
## 38  -1.47054427   1.0012703   1.26598413  -8.9147932    t
## 39  -3.95942422  -2.0426905  -3.51722851  -4.9064086    n
## 40  -0.26225128  -2.9503093  -5.40120937   2.0404600    t
## 41   2.40793404   1.7532596  -5.40484659   1.6853396    t
## 42  -3.82682171   1.6620698  -7.27183552   5.1777837    n
## 43   6.18380676  15.1760346   0.22837942  -8.1540277    n
## 44   2.43197801  10.4586612  12.51262092   3.7667300    t
## 45 -11.01321497  -0.9928513   3.63609456  -3.1017036    t
## 46   6.94671948   2.7983750   6.25881530   7.5364829    t
## 47  -1.21982851  12.0465809   3.29782577  11.9258169    t
## 48  -0.74866826   7.6365212  -4.12670239  -6.1018451    n
## 49  -0.72179109  -0.4913101  -4.22412809   3.4129312    t
## 50   0.49164939 -10.6491735  -0.16071103  -5.3969222    n
## 51  -9.48967820 -12.9143509  -4.54738278   6.3717268    n
## 52 -10.63062323 -14.4817433 -11.33239315  -2.2778641    t
## 53  -1.56930757  -4.7114079  -5.10142034   0.2883871    t
## 54  -2.87973579  -0.9297110  -2.41074292   1.2066317    n
## 55  15.12158238  -6.9516794  -2.96335013  10.0703375    n
## 56   4.32381524   2.4245795  -3.40746859  -3.2704582    t
## 57  25.29789689  -5.7946635  -5.72497004   8.3110636    t
## 58   1.85660483   1.0681211  -6.12395909   3.2874372    t
## 59  -5.45894670  -9.9912271  -3.01908200  -2.7845330    t
## 60  -3.09543463   2.8527106  -5.20057066   1.4358928    n
## 61   6.24045084  -0.8182745   0.25033009   4.9764751    t
## 62   5.85271924  -1.9965519  -2.44336042  -2.2796801    n
```

Fit a feed-forward neural network with `nnet` function

```
library(nnet)

dat.nn <- nnet(clas ~ ., data = dat.n, subset = samp, size=4, rang=0.1, decay=5e-4, maxit=200)
```

```
## # weights:  49
## initial  value 21.153393
## iter  10 value 5.410141
## iter  20 value 1.223991
## iter  30 value 0.256605
## iter  40 value 0.225925
```

```
## iter  50 value 0.211922
## iter  60 value 0.195894
## iter  70 value 0.181524
## iter  80 value 0.176247
## iter  90 value 0.174189
## iter 100 value 0.173472
## iter 110 value 0.173047
## iter 120 value 0.171389
## iter 130 value 0.171171
## iter 140 value 0.171122
## iter 150 value 0.171086
## iter 160 value 0.171058
## iter 170 value 0.171046
## iter 180 value 0.171039
## iter 190 value 0.171032
## iter 200 value 0.171029
## final  value 0.171029
## stopped after 200 iterations
```

```
dat.nn
```

```
## a 10-4-1 network with 49 weights
## inputs: PC1 PC2 PC3 PC4 PC5 PC6 PC7 PC8 PC9 PC10
## output(s): clas
## options were - entropy fitting  decay=5e-04
```

In nnet: data -> data frame from which variables specified in formula (clas ~.) are taken and used as predictors 10 variables used in the input layer (10 input neurons) clas column is the response variable to predict subset -> index vector specifying the cases used in the training sample size -> number of neurons in the hidden layer (=4) - free parameter rang -> initial random weights (on [-0.1, 0.1]) activation function -> logistic sigmoid (outputs a value between 0 and 1, the estimated probability that an observation belongs to a particular class) loss function -> entropy (maximum likelihood) optimization of loss -> BFGS algorithm - iterative second-order quasi-Newton method for nonlinear optimization fast converging to the minimum of the loss function rang -> initial random weights (on [-0.1, 0.1]) decay -> parameter for weight decay regularization - added penalty to the loss function that causes the weights to exponentially decay to zero during BFGS optimization maxit -> maximum number of BFGS iterations (free parameter)

The ANN has 10 inputs, 4 hidden neurons, and 1 output neuron, so it has `(10*4) + 4 + (4*1) + 1` = 49 weights

Confusion matrix

Evaluate accuracy of the ANN model on non-training (test) data

```
#cm <- table(dat.n$clas[-samp], predict(dat.nn, dat.n[-samp,], type = "class"))
actual.class <- dat.n$clas[-samp]
predicted.class <- predict(dat.nn, dat.n[-samp,], type = "class")
cm <- table(actual.class, predicted.class)

cat("\nConfusion matrix for the ANN model: \n\n")
```

```
##
## Confusion matrix for the ANN model:
```

```
# each row in the confusion matrix represents an actual class
# each column represents the predicted class
# first row considers the negative class (n - Normal condition)
# second row the positive class (t - Tumor condition)
cm
```

```
##             predicted.class
## actual.class  n  t
##            n 10  2
##            t  4 16
```

```
#cm.df <- as.data.frame(cm)
#cm.df
```

The confusion matrix explained:

The diagonal: top left - true negatives (TN): number of true normals (n - negative condition) classified as normal (n - negative) bottom right - true positives (TP): number of true tumors (t - positive condition) clasiffied as tumor (t - positive) Off-diagonal: top right - false positives (FP): number of true normals misclassified as tumors bottom left - false negatives (FN): number of true tumors misclassified as normal

```
cm.expl <- matrix(c('TN', 'FN', 'FP', 'TP'), nrow=2, ncol=2)
rownames(cm.expl) <- c('n', 't')
colnames(cm.expl) <- c('n', 't')

print.table(cbind(cm.expl, cm))
```

```
##   n  t  n  t
## n TN FP 10 2
## t FN TP 4  16
```

Performance metrics

True positive rate/Recall/Sensitivity - the fraction of positive instances correctly detected by the classifier = **TP/(TP+FN)** (=posivitives correctly classified/total positives)

False positive rate - the fraction of negative instances incorrectly classified = **FP/(FP+TN)** (=negatives incorrectly classified/total negatives)

True negative rate/Specificity = 1-false positive rate = TN/(FP+TN)

Precision - the fraction of positive predictions that are correct; accuracy of positive predictions = **TP/(TP+FP)** (=positives correctly predicted/total predicted positives)

Accuracy - the fraction of the number of correct predictions from the total number of predictions = **(TP+TN)/(TP+TN+FP+FN)** (measure of model's performance across all classes)

F1-measure - harmonic mean of precision and recall = **2/(1/precision + 1/recall) = TP/(TP + (FN+FP)/2)** (high if both precision and recall are high)

```
# True positive rate/Recall/Sensitivity
tpr <- cm[-1,-1]/(cm[-1,-1] + cm[-1,1])
tpr
```

```
## [1] 0.8
```

```
recall <- cm[-1,-1]/sum(cm[-1,])
recall
```

```
## [1] 0.8
```

```
sensitivity <- cm[2,2]/sum(cm[2,])
sensitivity
```

```
## [1] 0.8
```

```
# False positive rate
fpr <-  cm[1,-1]/sum(cm[1,])
fpr
```

```
## [1] 0.1666667
```

```
# True negative rate/Specificity
tnr <- cm[1,1]/sum(cm[1,])
tnr
```

```
## [1] 0.8333333
```

```
specificity <- 1 - fpr       # = tnr
specificity
```

```
## [1] 0.8333333
```

```
# Precision
precision <- cm[-1,-1]/sum(cm[,-1])
precision
```

```
## [1] 0.8888889
```

```
#Accuracy
accuracy <- sum(diag(cm))/sum(cm)
accuracy
```

```
## [1] 0.8125
```

```
# F1-measure
f1.score <- 2/(1/precision + 1/recall)
f1.score
```

```
## [1] 0.8421053
```

```r
metrics <- as.data.frame(
  round(cbind(sensitivity, specificity, accuracy, precision, recall, f1.score),3),
  col.names = c('Sensitivity', 'Specificity', 'Accuracy', 'Precision', 'Recall', 'F1 measure'))
metrics
```

```
##   sensitivity specificity accuracy precision recall f1.score
## 1         0.8       0.833    0.812     0.889    0.8    0.842
```