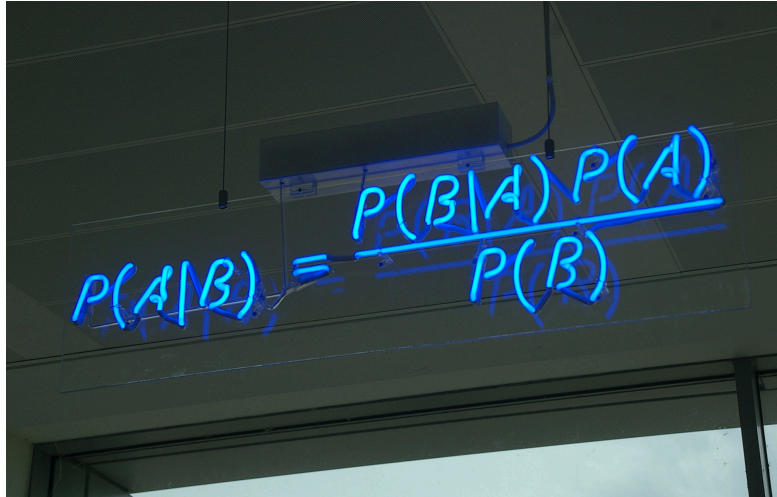

0) LDA, or Using Bayes' theorem for classification


$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

In our case:

$$\Pr(Y = k|X = x) = \frac{\pi_k f_k(x)}{\sum_{l=1}^K \pi_l f_l(x)}$$

where

$$f_k(X) \equiv \Pr(X = x|Y = k)$$

We know from the theory that Bayes' classifier has the *lowest possible error rate*. This, of course, is true only if the terms in the previous equations are all correctly specified.

What we do in LDA is (1) doing an assumption on the nature of f_k and (2) estimating the parameters that we need from the data.

(1) Let us first work on the 1D case (one-dimensional observations). Assume that f_k is a *Gaussian*:

$$f_k(x) = \frac{1}{\sqrt{2\pi}\sigma_k} \exp\left(-\frac{1}{2\sigma_k^2}(x - \mu_k)^2\right)$$

then $P(Y=k|X=x)$ is

$$p_k(x) = \frac{\pi_k \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2\sigma^2}(x - \mu_k)^2\right)}{\sum_{l=1}^K \pi_l \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2\sigma^2}(x - \mu_l)^2\right)}$$

(some terminology: π_k = PRIORS, p_k = POSTERIORs)

Classifying x means assigning it the class k for which $p_k(x)$ (or, equivalently, the *discriminant function* $\delta_k(x)$, is largest:

$$\hat{\delta}_k(x) = x \cdot \frac{\hat{\mu}_k}{\hat{\sigma}^2} - \frac{\hat{\mu}_k^2}{2\hat{\sigma}^2} + \log(\hat{\pi}_k)$$

(you can derive the discriminant function from the equation above, by applying $\log(\cdot)$ and considering that the denominator always remains the same).

(2) The parameters for the discriminant function can be derived as follows:

$$\hat{\mu}_k = \frac{1}{n_k} \sum_{i:y_i=k} x_i$$

$$\hat{\sigma}^2 = \frac{1}{n - K} \sum_{k=1}^K \sum_{i:y_i=k} (x_i - \hat{\mu}_k)^2$$

$$\hat{\pi}_k = n_k/n.$$

Note that we also assume all the distributions have the same variance! We can approximate a common variance from the data by using pooled variance (see https://en.wikipedia.org/wiki/Pooled_variance). Pooled variance can be estimated by the weighted average of the sample variances.

1) 1D LDA example

Here is an example with two 1-D classes (akin to the one shown in Figure 4.4, page 140 ISLR):

```
# generate data
X1size = 100
X2size = 100
totsize = X1size + X2size
X1 = rnorm(X1size, -1.5, 1)
X2 = rnorm(X2size, 1.5, 1)
X = rbind(as.matrix(X1), as.matrix(X2))
Y = rbind(as.matrix(rep(1, X1size)), as.matrix(rep(2, X2size)))

# plot the histograms of the two distributions
hist(X1, col=rgb(1,0,0,0.5), xlim=c(min(c(X1,X2)), max(c(X1,X2))))
hist(X2, col=rgb(0,0,1,0.5), add=T)

# estimate parameters
mu1 = mean(X1)
mu2 = mean(X2)
sigmasq = (sum((X1-mu1)^2)+sum((X2-mu2)^2))/(totsize-2)
pi1 = X1size/totsize
pi2 = X2size/totsize
```

```

# all the discriminant functions are linear in x (ax + b)
a1 = mu1/sigmasq
a2 = mu2/sigmasq
b1 = log(pi1) - mu1^2/(2*sigmasq)
b2 = log(pi2) - mu2^2/(2*sigmasq)

# x belongs to class 1 if x <
(mu1^2-mu2^2)/(2*(mu1-mu2))
# equivalent to
(mu1 + mu2)/2

# also check:
(a1 * sort(X) + b1) > (a2 * sort(X) + b2)

```

2) 2D (and 1D!) LDA example

Consider the following dataset with three classes and the Linear Discriminant Analysis model (LDA hereafter)

X1	X2	class
1	1	A
2	2	A
2	3	A
3	1	A
4	1	A
1	4	B
2	5	B
3	4	B
4	5	B
4	3	C
6	1	C
6	2	C
6	3	C

Let us first try to solve it as a 1D problem, considering only observations from X2 as input (this is something you should be able to calculate manually, as it has been part of exam exercises in the past!)

```

# prepare data
X = c(1,2,3,1,1,4,5,4,5,3,1,2,3)
Y = c(1,1,1,1,1,1,2,2,2,2,3,3,3,3)

# split to create the different classes
x1 = X[Y==1]
x2 = X[Y==2]
x3 = X[Y==3]
n = length(X)
n1 = length(x1)
n2 = length(x2)
n3 = length(x3)

# try to plot these data
hist(x1, col=rgb(1,0,0,0.5), xlim=c(1,5))
hist(x2, col=rgb(0,1,0,0.5), xlim=c(1,5), add=T)

```

```

hist(x3, col=rgb(0,0,1,0.5), xlim=c(1,5), add=T)

# p values are the priors (the probability that one element belongs
to a given class k)
p = c(n1/n, n2/n, n3/n)

# get means
mu1 = mean(x1)
mu2 = mean(x2)
mu3 = mean(x3)

# get variances
s1 = sum((x1-mu1)^2)/(n1-1)
s2 = sum((x2-mu2)^2)/(n2-1)
s3 = sum((x3-mu3)^2)/(n3-1)

# calculate pooled variance
S = ((n1-1)*s1 + (n2-1)*s2 + (n3-1)*s3) / (n-3)

# calculate deltas, applied to all data
deltas = cbind(X * mu1 / S - mu1^2/(2*S) + log(p[1]), X * mu2 / S -
mu2^2/(2*S) + log(p[2]), X * mu3 / S - mu3^2/(2*S) + log(p[3]))

# get the classes (values for which deltas are highest)
max.col(deltas)

```

This, instead, is the 2D example (probably too complex to calculate manually...)

```

# prepare data
X1 = c(1,2,2,3,4,1,2,3,4,4,6,6,6)
X2 = c(1,2,3,1,1,4,5,4,5,3,1,2,3)
Y = c(1,1,1,1,1,2,2,2,2,3,3,3,3)
m = data.frame(cbind(X1,X2,Y))

# split to create the different classes
m1 = m[m$Y==1,1:2]
m2 = m[m$Y==2,1:2]
m3 = m[m$Y==3,1:2]
n = dim(m)[1]
n1 = dim(m1)[1]
n2 = dim(m2)[1]
n3 = dim(m3)[1]

# p values are the priors (the probability that one element belongs
to a given class k)
p = as.matrix(c(n1/n, n2/n, n3/n))

# get means
mu1 = as.matrix(colMeans(m1))
mu2 = as.matrix(colMeans(m2))

```

```

mu3 = as.matrix(colMeans(m3))

# get covariances
s1 = cov(m1)
s2 = cov(m2)
s3 = cov(m3)

# calculate pooled covariance and its inverse
S = ((n1-1)*s1+(n2-1)*s2+(n3-1)*s3)/(n-3)
library(pracma)
Si = inv(S)

# calculate deltas
deltas = cbind(as.matrix(m[,1:2]) %*% Si %*% mu1 - as.double(1/2 *
t(mu1) %*% Si %*% mu1 + log(p[1])),
as.matrix(m[,1:2]) %*% Si %*% mu2 - as.double(1/2 * t(mu2) %*% Si
%*% mu2 + log(p[2])),
as.matrix(m[,1:2]) %*% Si %*% mu3 - as.double(1/2 * t(mu3) %*% Si
%*% mu3 + log(p[3])))

max.col(deltas)

```

3) Now compare logistic regression, LDA, QDA, and KNN on the same synthetic dataset

Let us have a more realistic approach, with disjoint training and test sets:

```

set.seed(20)
# per class observations for training and test sets
trainObs = 100
testObs = 50
trainSize = 2*trainObs
testSize = 2*testObs

# Both training and test are normal distributions, with the same
variance, and slightly overlapped.
# Play with mean and variance and see what happens to
classification results when sets overlap more/less
mean_1 = 5
mean_2 = 6
var_1 = .8
var_2 = .8

train.x = rbind(cbind(rnorm(trainObs,mean_1,var_1),
rnorm(trainObs,mean_1,var_1)),
               cbind(rnorm(trainObs,mean_2,var_2),
rnorm(trainObs,mean_2,var_2)))
test.x = rbind(cbind(rnorm(testObs,mean_1,var_1),
rnorm(testObs,mean_1,var_1)),
               cbind(rnorm(testObs,mean_2,var_2),

```

```

rnorm(testObs,mean_2,var_2)))

# look at training and test sets
x_range =
c(min(min(train.x[,1]),min(test.x[,1])),max(max(train.x[,1]),max(test.x[,1])))
y_range =
c(min(min(train.x[,2]),min(test.x[,2])),max(max(train.x[,2]),max(test.x[,2])))
plot(train.x, xlim=x_range, ylim=y_range)
points(train.x[(trainObs+1):trainSize,], col='red')
plot(test.x, xlim=x_range, ylim=y_range)
points(test.x[(testObs+1):testSize,], col='red')

# build ground truth classes
train.y = rep(0,trainSize)
train.y[(trainObs+1):trainSize] = 1
test.y = rep(0,testSize)
test.y[(testObs+1):testSize] = 1

# build data frames (to be used by the fit and pred functions)
train.df = data.frame(train.x)
test.df = data.frame(test.x)

# run logistic regression
logr.fit = glm(train.y ~ X1 + X2, data = train.df, family =
binomial)
summary(logr.fit)

logr.probs = predict(logr.fit, test.df, type="response")

logr.pred = rep(0,testSize)
logr.pred[logr.probs>.5] = 1

table(logr.pred,test.y)
mean(logr.pred==test.y)

# extend to LDA
library(MASS)
?lda
?qda

lda.fit = lda(train.y ~ X1 + X2, data = train.df)
lda.fit
lda.pred = predict(lda.fit, test.df)
lda.class = lda.pred$class
table(lda.class,test.y)
mean(lda.class == test.y)

#extend to qda
qda.fit = qda(train.y ~ X1 + X2, data = train.df)

```

```

qda.fit
qda.pred = predict(qda.fit, test.df)
qda.class = qda.pred$class
table(qda.class, test.y)
mean(qda.class == test.y)

#extend to KNN
library(class)
knn.pred = knn(train.df, test.df, train.y, k = 1)
table(knn.pred, test.y)
mean(knn.pred == test.y)

```

For a geometric interpretation of the covariance matrix, check:

<http://www.visiondumy.com/2014/04/geometric-interpretation-covariance-matrix/>

4) Perform an extensive test to compare the different approaches on different datasets

```

logr.errs = NULL
lda.errs = NULL
qda.errs = NULL
knn1.errs = NULL
knn3.errs = NULL

set.seed(20)

for (i in 1:10 ) {
# per class observations for training and test sets
trainObs = 100
testObs = 100
trainSize = 2*trainObs
testSize = 2*testObs

train.x = rbind(cbind(rnorm(trainObs,5,.8), rnorm(trainObs,5,.8)),
                    cbind(rnorm(trainObs,6,.8), rnorm(trainObs,6,.8)))
test.x = rbind(cbind(rnorm(testObs,5,.8), rnorm(testObs,5,.8)),
                    cbind(rnorm(testObs,6,.8), rnorm(testObs,6,.8)))

plot(test.x)
points(test.x[(testObs+1):testSize,], col='red')

train.y = rep(0, trainSize)
train.y[(trainObs+1):trainSize] = 1
test.y = rep(0, testSize)
test.y[(testObs+1):testSize] = 1

train.df = data.frame(train.x)
test.df = data.frame(test.x)

logr.fit = glm(train.y ~ X1 + X2, data = train.df, family =

```

```

binomial)
lda.fit = lda(train.y ~ X1 + X2, data = train.df)
qda.fit = qda(train.y ~ X1 + X2, data = train.df)

logr.probs = predict(logr.fit, test.df, type="response")
logr.pred = rep(0, testSize)
logr.pred[logr.probs > .5] = 1
logr.errs[i] = mean(logr.pred != test.y)

lda.pred = predict(lda.fit, test.df)
lda.class = lda.pred$class
lda.errs[i] = mean(lda.class != test.y)

qda.pred = predict(qda.fit, test.df)
qda.class = qda.pred$class
qda.errs[i] = mean(qda.class != test.y)

knn1.pred = knn(train.df, test.df, train.y, k = 1)
knn1.errs[i] = mean(knn1.pred != test.y)

knn3.pred = knn(train.df, test.df, train.y, k = 3)
knn3.errs[i] = mean(knn3.pred != test.y)

}

errs = data.frame(cbind(logr.errs, lda.errs, qda.errs, knn1.errs,
knn3.errs))
library(psych)
error.bars(errs)

```

5) Further extend to a non-linear case

```

logr.errs = NULL
lda.errs = NULL
qda.errs = NULL
knn1.errs = NULL
knn3.errs = NULL

set.seed(20)

for (i in 1:10 ) {

trainObs = 100
testObs = 100
trainSize = 2*trainObs
testSize = 2*testObs

X1 = rnorm(trainObs, mean=10, sd=.9)
X2 = X1 - 20 + rnorm(trainObs, mean=8, sd=.9)

```



```

Y1 = rnorm(trainObs,mean=10,sd=.9)
Y2 = - 2*Y1 + 10 + rnorm(trainObs, mean=11, sd=1.1)

train.x = rbind(cbind(X1,X2), cbind(Y1,Y2))
plot(train.x)
points(train.x[(trainObs+1):trainSize,], col='red')

X1 = rnorm(testObs,mean=10,sd=.9)
X2 = X1 - 20 + rnorm(testObs, mean=8, sd=.9)
Y1 = rnorm(testObs,mean=10,sd=.9)
Y2 = - 2*Y1 + 10 + rnorm(testObs, mean=11, sd=1.1)

test.x = rbind(cbind(X1,X2), cbind(Y1,Y2))
plot(test.x)
points(test.x[(testObs+1):testSize,], col='red')

# build ground truth classes
train.y = rep(0,trainSize)
train.y[(trainObs+1):trainSize] = 1
test.y = rep(0,testSize)
test.y[(testObs+1):testSize] = 1

# build data frames (to be used by the fit and pred functions)
train.df = data.frame(train.x)
test.df = data.frame(test.x)

logr.fit = glm(train.y ~ X1 + X2, data = train.df, family =
binomial)
lda.fit = lda(train.y ~ X1 + X2, data = train.df)
qda.fit = qda(train.y ~ X1 + X2, data = train.df)

logr.probs = predict(logr.fit, test.df, type="response")
logr.pred = rep(0,testSize)
logr.pred[logr.probs>.5] = 1
logr.errs[i] = mean(logr.pred!=test.y)

lda.pred = predict(lda.fit, test.df)
lda.class = lda.pred$class
lda.errs[i] = mean(lda.class != test.y)

qda.pred = predict(qda.fit, test.df)
qda.class = qda.pred$class
qda.errs[i] = mean(qda.class != test.y)

knn1.pred = knn(train.df, test.df, train.y, k = 1)
knn1.errs[i] = mean(knn1.pred!=test.y)

knn3.pred = knn(train.df, test.df, train.y, k = 3)
knn3.errs[i] = mean(knn3.pred!=test.y)

}

```

```
errs = data.frame(cbind(logr.errs, lda.errs, qda.errs, knn1.errs,
knn3.errs))
library(psych)
error.bars(errs)
```

6) Testing with the "heart" dataset

```
heart =
read.table("~/Downloads/SAheart.data", sep="," ,head=T,row.names=1)
attach(heart)
```

```
trainSize = 300
train = rep (FALSE,dim(heart)[1])
train[sample.int(dim(heart)[1],trainSize)]=TRUE
heart.test = heart[!train,]
chd.test = chd[!train]
```

```
logr.fit = glm(chd ~ ., data=heart, family = binomial, subset =
train)
lda.fit = lda(chd ~ ., data=heart, subset = train)
qda.fit = qda(chd ~ ., data=heart, subset = train)
```

```
logr.probs = predict(logr.fit, heart.test, type="response")
logr.pred = rep(0,dim(heart.test)[1])
logr.pred[logr.probs>.5] = 1
table(chd.test, logr.pred)
mean(logr.pred != chd.test)
```

```
lda.pred = predict(lda.fit, heart.test)
lda.class = lda.pred$class
table(chd.test, lda.class)
mean(lda.class != chd.test)
```

```
qda.pred = predict(qda.fit, heart.test)
qda.class = qda.pred$class
table(chd.test, qda.class)
mean(qda.class != chd.test)
```

7) Check out the curse of dimensionality exercise (4.4) and look for the online tutorial for a better explanation:

<http://www.visiondummys.com/2014/04/curse-dimensionality-affect-classification/>