ST 437/537: Applied Multivariate and Longitudinal Data Analysis

Discriminant Analysis and Classification

Arnab Maity
NCSU Department of Statistics
SAS Hall 5240 919-515-1937 amaity[at]ncsu.edu

Introduction

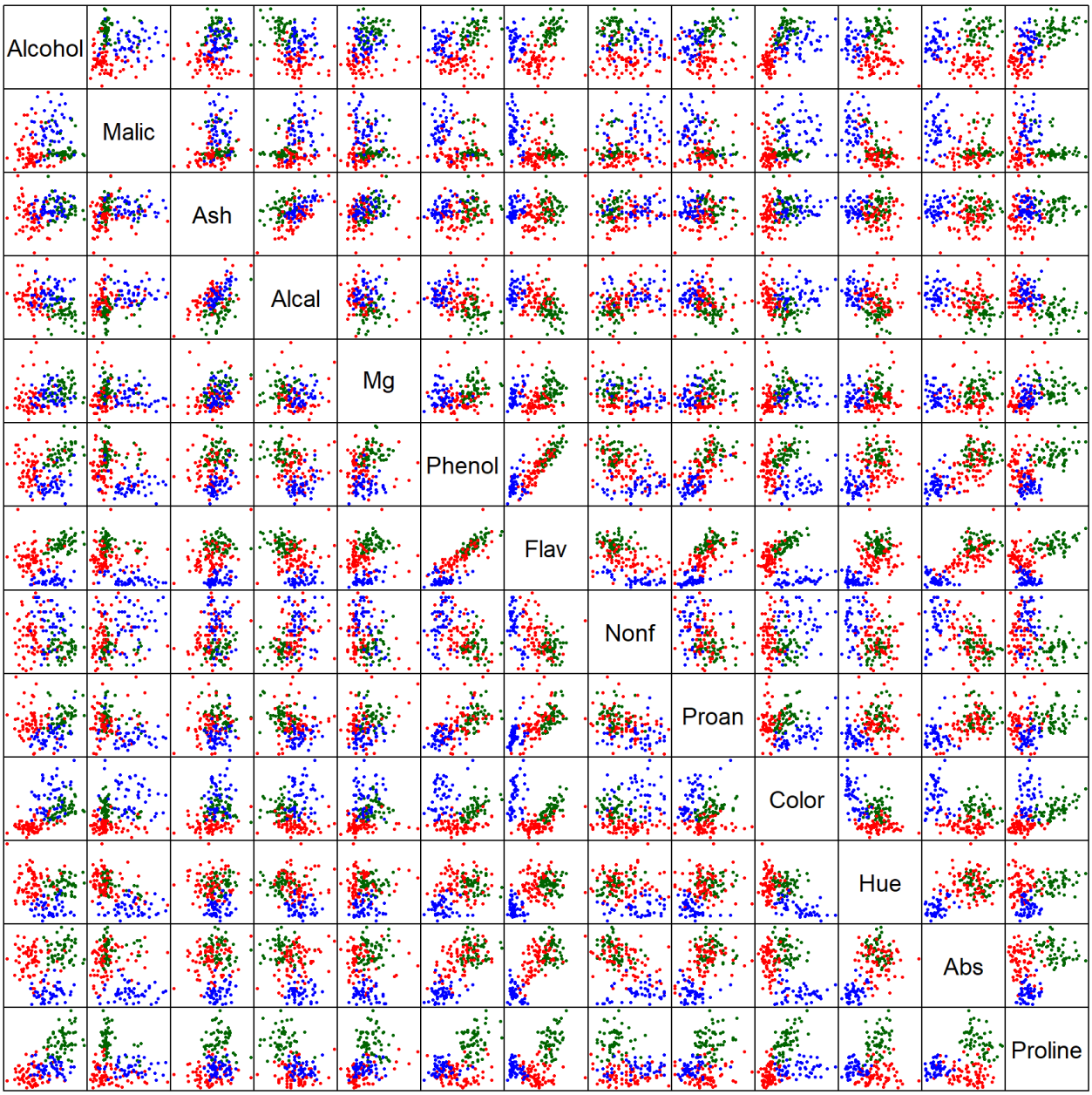
The problem of separating two or more groups is sometimes called discrimination or ``supervised" classification.

- **Discrimination:** finding the features that *separate* known groups in a multivariate sample.
- **Classification:** developing a rule to *allocate* a new object into one of a number of known groups.

A classification rule is based on the features that separate the groups, so the two goals overlap. Making mistakes is inevitable; our goal is to quantify the cost of misclassification and try to make as few mistakes as possible.

Consider the `wines` data set available at [**https://archive.ics.uci.edu/ml/datasets/wine (https://archive.ics.uci.edu/ml/datasets/wine)**]. The data set is also available with the textbook Applied Multivariate Statistics with R by Zelterman, given **[here] (data/Wines.txt)**.

```
# Read the data
wines <- read.table("data/wines.txt", header = TRUE)
colors <- c("darkgreen", "red", "blue")[wines$Class]

# pairs plot
pairs(wines[, -1], pch = 16, cex = .5, gap = 0, col = colors, xaxt = "n", yaxt = "n")
```



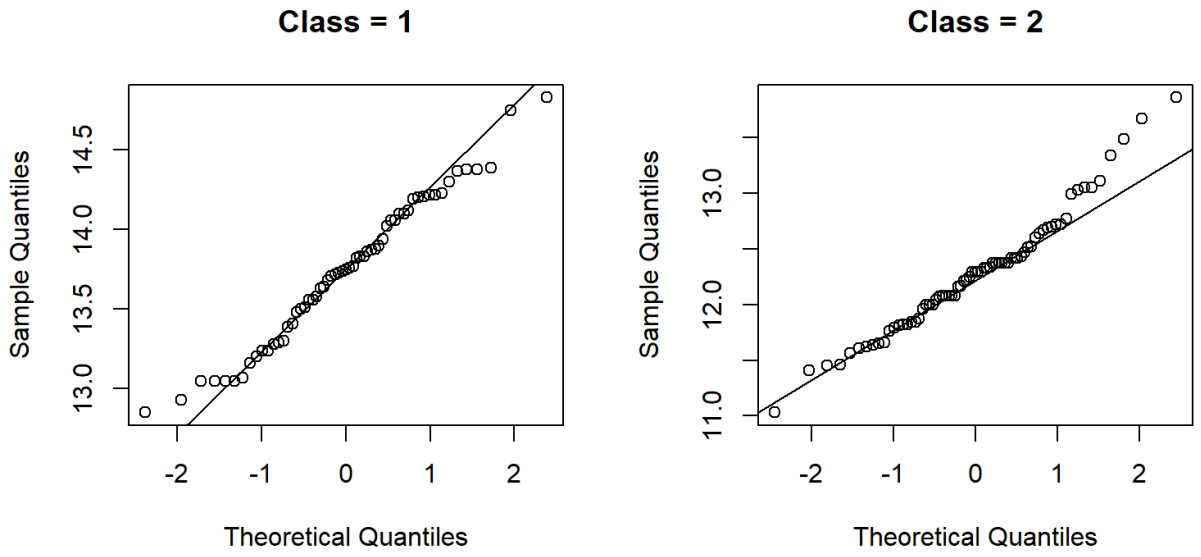

```
# classes of wine
class <- wines$Class
tabulate(class)
```

```
## [1] 59 71 48
```

Two Groups

Consider the `wine` data with only two classes (1 and 2) and with only the `Alcohol` variable.

```
# Alcohol for classes 1 and 2
alc <- wines$Alcohol[wines$Class == 1 | wines$Class == 2]
newclass <- wines$Class[wines$Class == 1 | wines$Class == 2]

# Normal Q-Q plots
par(mfrow=c(1,2))
qqnorm(alc[newclass==1], main = "Class = 1")
qqline(alc[newclass==1])
qqnorm(alc[newclass==2], main = "Class = 2")
qqline(alc[newclass==2])
```



Thw Q-Q plots show fairly linear pattern (except may be only a few points). Let us assume that the data from both classes follow normal distributions. We plot the distributions below.

```
# Mean and variance of the two groups
xbar.1 <- mean(alc[newclass==1])
var.1 <- var(alc[newclass==1])

xbar.2 <- mean(alc[newclass==2])
var.2 <- var(alc[newclass==2])

# Means of the two groups
c(xbar.1, xbar.2)
```

```
## [1] 13.74475 12.27873
```

```
# SD of the two groups
c(var.1, var.2)
```

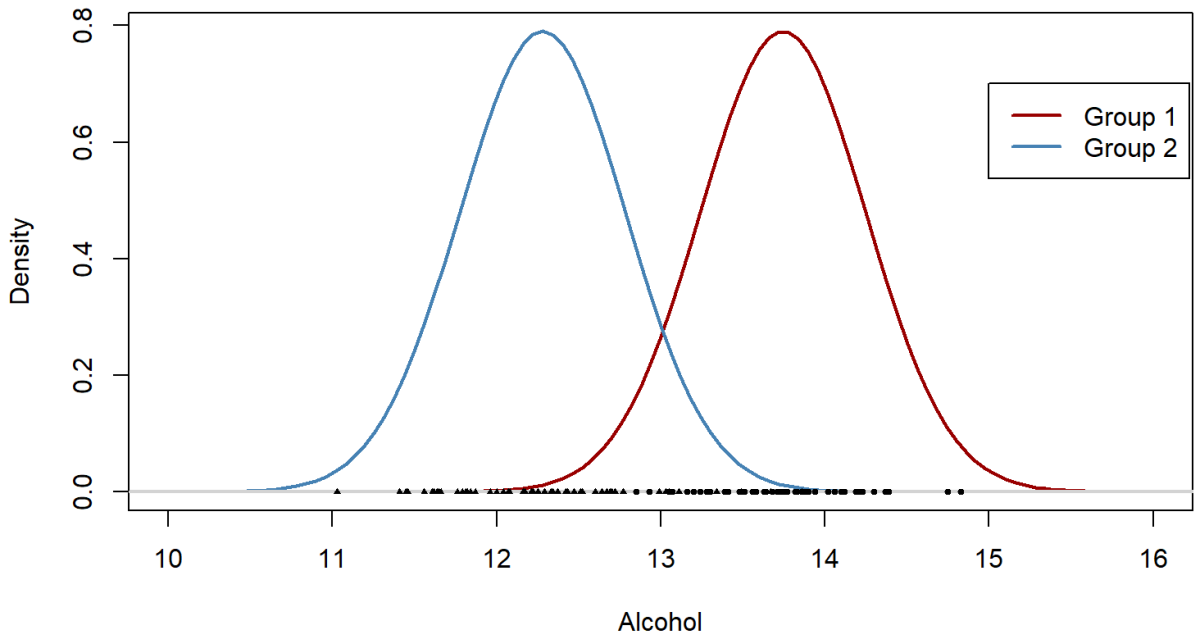
```
## [1] 0.2135598 0.2894055
```

Since the variances of the two groups are close, let us assume that the the two groups have the same variance; we will see later that this assumption can be relaxed. The common variance can be estimated by a “pooled” estimator.

```
n1 <- sum(newclass==1)
n2 <- sum(newclass==2)
var.p <- ( (n1-1)*var.1 + (n2-1)*var.2 )/(n1+n2-2)
sd.p <- sqrt(var.p)
```

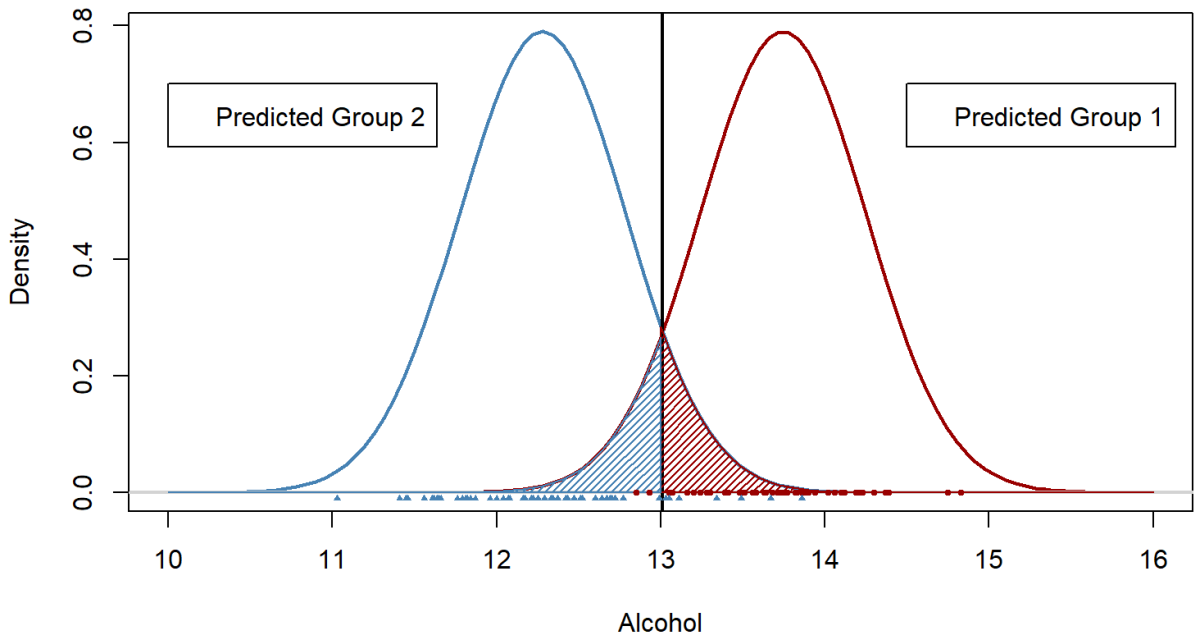
We plot the distributions below.

```
# set up the grid over which to plot
grid <- seq(10, 16, length.out = 101)

# Normal pdfs of two groups
f1 <- dnorm(grid, mean = xbar.1, sd = sd.p)
f2 <- dnorm(grid, mean = xbar.2, sd = sd.p)

# Plot the pdfs
matplot(grid, cbind(f1, f2), lwd=2, type = "l", lty=1, col = c("#990000", "steelblue"), ylab = "Density",
 xlab = "Alcohol")
abline(h=0, lwd=2, col="lightgray")
points(alc[newclass==1], 0*alc[newclass==1], pch=19, cex=0.5)
points(alc[newclass==2], 0*alc[newclass==2], pch=17, cex=0.5)
legend(15, 0.7, legend = c("Group 1", "Group 2"), col = c("#990000", "steelblue"), lwd=2, lty=1)
```



Can we tell which points come from which density? Can we find a “rule” or a “feature” that will allow us seperate these two groups? This the core problem in discrimination. One such rule is

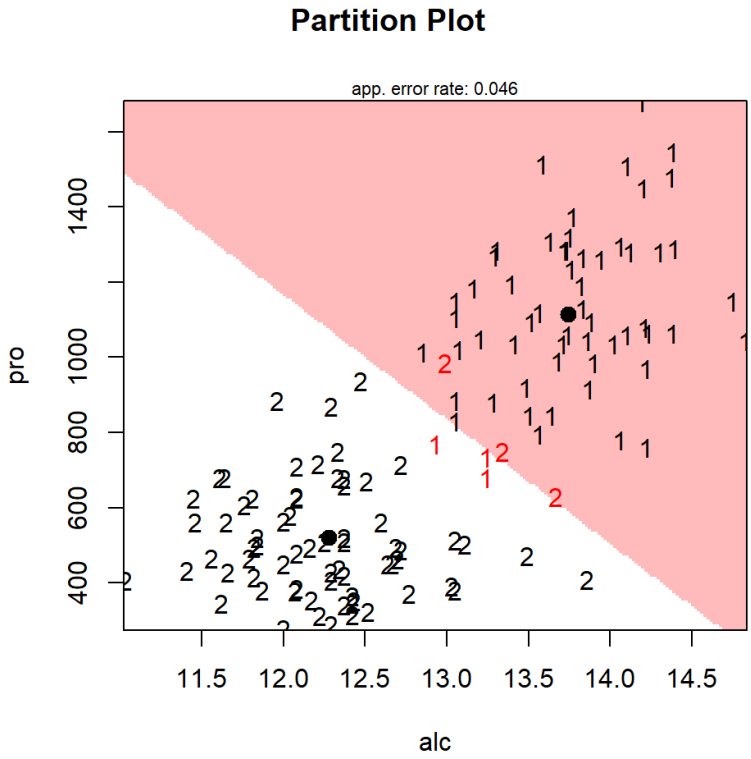
> *An item belongs to group 1 if alcohol > 13, group 2 otherwise.*



Note the following features of the plot above:

- There are clearly two type of **misclassification errors**
  - we can assign a item to group 1 when in reality it belongs to group 2 (this region is shown in red). There are “blue” points in the red region; these are misclassified.
  - we can assign a item to group 2 when in reality it belongs to group 2 (this region is shown in blue). There are “red” points in the “blue” region; these are also misclassified.
- The rule presented above is based on whether `alcohol > 13` or not. Is this rule optimal? How to define optimality?
- We also need to understand what are the cost of misclassification error. Do the types errors have same or different cost?

These ideas can be extended to two dimensions. Let us consider two classes (class 1 and 2) but with two variables `alcohol` and `proline`.



In this case we would like to find a "direction" (the boundary between the red and white regions) that separates the two groups.

In general, we have the follwong setup:

- For each item, we observe data vector $\boldsymbol{X}$, and a group indicator $G = 1/2$.
- The density of $\boldsymbol{X}$ given $G = g$ is $f_g(\cdot)$, $g = 1, 2$.
- $\mathrm{P}(G = 1) = p_1$, $\mathrm{P}(G = 2) = p_2 = 1 - p_1$; p_1 and p_2 are the *prior probabilities* of the groups.

Classification rule: Rule must give an prediction of group membership for any $\boldsymbol{X}$, so it defines two **regions**, R_1 (predicted group 1) and R_2 (predicted group 2)

Without going into technicalities, **Expected Cost of Misclassification** (ECM) is defined as

$$\mathrm{ECM} = \begin{matrix}\text{expected cost of wrongly} \\ \text{classifying an grp 1 as a grp 2}\end{matrix} + \begin{matrix}\text{expected cost of wrongly} \\ \text{classifying a grp 2 as an grp 1}\end{matrix}$$

The **optimal classification rule** is the one that minimizes the ECM. The rule/regions regions depend on the *ratios* of:

- the densities: $f_2(\boldsymbol{x})/f_1(\boldsymbol{x})$
- the costs of misclassification
- the prior probabilities p_1/p_2

Remarks:

- When the prior probabilities are unknown, they are taken to be equal.

- When the misclassification costs are unknown, they are taken to be equal. The optimal classification rule becomes

> *Classify an item (with covariate $\boldsymbol{x}$) in group 2 if $f_2(\boldsymbol{x})/f_1(\boldsymbol{x}) \geq p_1/p_2$; group 1 otherwise.*

- When both the prior probabilities and misclassification costs are unknown, prior probablities are taken to be the same, the misclassification costs are assumed to be equal. In this case, the classification rule become simple:

> *Classify an item (with covariate $\boldsymbol{x}$) in group 2 if $f_2(\boldsymbol{x}) \geq f_1(\boldsymbol{x})$; group 1 otherwise.*

How to evaluate a classifier

We can use the following creteria to evaluate a classification rule.

- Accuracy of the classifier: $\frac{\text{Total correct classification}}{\text{Total number of points}}$
- Apparent error (APER): $\frac{\text{Total incorrect classification}}{\text{Total number of points}}$.

Confusion matrix/error matrix: a matrix (table) containing information about predicted and actual classifications obtained by a classification rule. An example of such a matrix is shown below.

```
##        predicted
## true     1  2 -SUM-
##   1     57  2     2
##   2      7 64     7
##   -SUM-  7  2     9
```

In this case, APER = (2+7)/130 = 0.0692308.

APER is easy to calculate but tends to underestimate the actual error. One could use a hold-out method to estimate the actual error

Holdout method:

- Omit one observation (`holdout") from the data set and develop a classification rule based on the remaining observations
- Based of that classification rule, predict the class of the holdout observation
- Repeat the above two steps for each of the observation of the data set and compute the confusion matrix. Then, the estimated error rate is (Total incorrect classification)/(Total number of points)

Training and test sets: If the data set is large enough,

- we can randomly select a portion of the data set (the `training set`), for example, say 80% of the data, and build the classification rule
- predict the classes of the remaining 20% data (the `test set`).
- compute the APER and accuracy based on the test set predictions.

Linear Discriminant Analysis (LDA)

Model assumption: normality + equal covariance

$$X|G=1 \sim N(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}), \quad X|G=2 \sim N(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}).$$

When the prior probabilities and misclassification costs are unknown, the optimal rule is

> *An item (with covariate $\boldsymbol{x}$) is classified in group 2 if $\boldsymbol{a}^T\boldsymbol{x} \geq \boldsymbol{b}$, where $\boldsymbol{a} = \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0)$, and $\boldsymbol{b} = \frac{1}{2}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0)^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_1 + \boldsymbol{\mu}_0)$.*

In practice, the true values of $\boldsymbol{\mu}_1$, $\boldsymbol{\mu}_2$ and $\boldsymbol{\Sigma}$ are unknown, we estimate these parameters as

$$\hat{\boldsymbol{\mu}}_1 = \text{ sample mean of group 1 }, \quad \hat{\boldsymbol{\mu}}_2 = \text{ sample mean of group 2 }, \quad \hat{\boldsymbol{\Sigma}} = \text{ pooled sample covariance.}$$

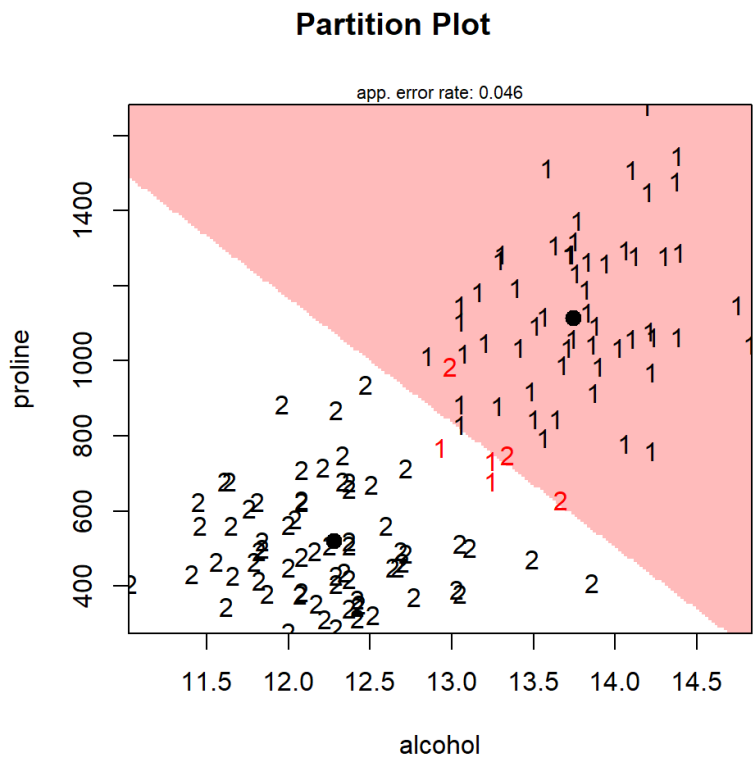
Note: the pooled sample covariance is defined as

$$\hat{\boldsymbol{\Sigma}} = \frac{(n_1-1)\boldsymbol{S}_1 + (n_2-1)\boldsymbol{S}_2}{n_1+n_2-2},$$

where $\boldsymbol{S}_1$ and $\boldsymbol{S}_2$ are sample covariance matrices from group 1 and 2, respectively.

The R function `lda` in the `MASS` package performs LDA. The function `partimat` helps to visualize the classification rule. Let us consider two variables `alcohol` and `proline`, and only two classes in the wine data.

```
library(klaR)
# Extract proline
pro <- wines$Proline[wines$Class == 1 | wines$Class == 2]

# create the data matrix
data <- data.frame(proline = pro, alcohol = alc, group = factor(newclass))
partimat(group ~ proline + alcohol, data = data, image.colors = c("#FFBBBB", "white"), prec=200)
```



To compute the APER for the LDA classifier, we use the training/test set method.

```
# set seed for reproduction
set.seed(1001)

# total number of items
n <- nrow(data)

# test set size
n.test <- round(n * 0.2)

# obtain the test and training sets
ind <- sample(1:n, size = n.test, replace = F)
test <- data[ind, ]
train <- data[-ind, ]

# train the classifier
x <- lda(group ~ proline + alcohol, data = train)

# test the classifier
y <- predict(x, test)

# Confusion matrix
emat <- errormatrix(test$group, y$class)
emat
```

```
##        predicted
## true     1  2 -SUM-
##   1     12  0     0
##   2      1 13     1
##   -SUM-  1  0     1
```

```
# APER
aper <- emat[3,3]/n.test
aper
```

```
## [1] 0.03846154
```

Typically, one would repeat this process a few times and compute average APER for a more stable estimate. We repear the above process 30 times; the average APER is below.

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.00000 0.00000 0.03846 0.03590 0.03846 0.11538
```

Quadratic Discriminant Analysis (QDA)

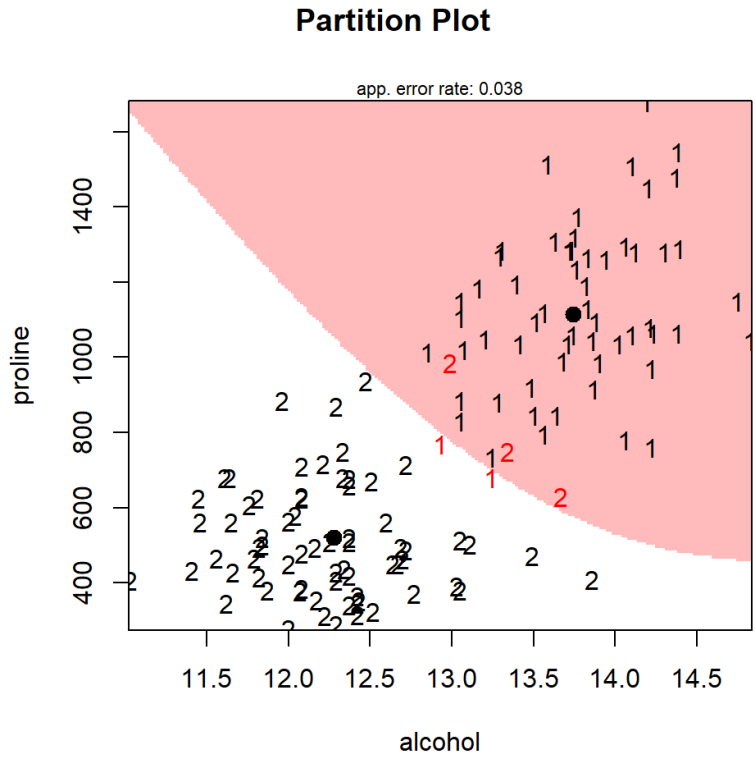
Model assumption: normality + unequal covariance

$$\boldsymbol{X}|G=1 \sim N(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1), \quad \boldsymbol{X}|G=2 \sim N(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2).$$

When the prior probabilities and misclassification costs are unknown, the optimal rule is quadratic, that is, the classification boundary is quadratic.

We can use the `qda` function in R to do so.

```
partimat(group ~ proline + alcohol, data = data, method = "qda",
         image.colors = c("#FFBBBB", "white"), prec=200)
```



Logistic Regression

The logistic regression model arises from the desire to model the posterior probabilities of each of the two classes as functions of the data.

Suppose we have two classes (0 and 1), and suppose for an item, we have covariate $\boldsymbol{x} = (x_1, \ldots, x_p)^T$, we assume

$$G \sim Bernoulli(p(\boldsymbol{x})),$$

that is

$$P(G = 1) = p(\boldsymbol{x}) \text{ and } P(G = 0) = 1 - p(\boldsymbol{x}).$$

We model $p(\boldsymbol{x})$ as

$$p(\boldsymbol{x}) = \frac{\exp\left(\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p\right)}{1 + \exp\left(\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p\right)}.$$

The paramaters $\beta_0, \beta_1, \ldots, \beta_p$ quantifies the impact of the covariates to the classifier.

The model parameters can be estimated directly by maximum likelihood, solution is obtained numerically by teratively reweighted least squares. It follows that $\mathrm{P}(G = 1)$ can be estimated by

$$\frac{\exp\left(\hat{\beta}_0 + \hat{\beta}_1 x_1 + \cdots + \hat{\beta}_p x_p\right)}{1 + \exp\left(\hat{\beta}_0 + \hat{\beta}_1 x_1 + \cdots + \hat{\beta}_p x_p\right)}$$

We can predict the class for a item with covariate $\boldsymbol{x}$ using the estimated probability that $G = 1$ as follows:

> *The item is classified in group* 1 *if* $\widehat{P}(G = 1|\boldsymbol{x}) \geq 0.5$*, otherwise in group* 0*.*

Logistic regression can be performed using the `glm` function in base R.

```
####### Logistic regression
fit.glm =  glm(group ~ pro + alc, family = binomial(), data = data)
```

The first part `group ~ pro + alc` is specifying `group` as response and `pro` and `alc` as covariates. The statement `family = binomial()` is used to perform logistic regression (performs linear regression without this statement).

```
# testing each beta coefficient
summary(fit.glm)
```

```
##
## Call:
## glm(formula = group ~ pro + alc, family = binomial(), data = data)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.57716  -0.02786   0.00558   0.03196   2.19220
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) 69.027638  22.757675   3.033  0.00242 **
## pro         -0.013695   0.004362  -3.140  0.00169 **
## alc         -4.453109   1.592916  -2.796  0.00518 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 179.11  on 129  degrees of freedom
## Residual deviance:  19.72  on 127  degrees of freedom
## AIC: 25.72
##
## Number of Fisher Scoring iterations: 9
```

The summary of the fit produces z-tests for coefficient of each covariate; it seems both `alc` and `pro` are associated with the group indicator.

```
# Prediction of P(G = 1)
post.prob = fit.glm$fitted

# Predicted groups
Group.hat = ifelse(post.prob>0.5, 2, 1)

# Confusion matrix
errormatrix(predicted = Group.hat, true = data$group)
```

```
##        predicted
## true     1  2 -SUM-
##   1     56  3     3
##   2      3 68     3
##   -SUM-  3  3     6
```

Comparison between LDA and logistic regression:

- Logistic regression models $\mathrm{P}(G = g|\boldsymbol{x})$ directly.
- When the groups are well separated, the parameters can be estimated well.
- Conversely, when the groups are not well separated, the mixture estimation problem is difficult.
- Linear discriminant analysis is more efficient when the normality assumptions on $\boldsymbol{X}$ are satisfied.
- Logistic regression does not depend on any assumptions about the distribution of $\boldsymbol{X}$.
- Logistic regression can include an arbitrary mix of squared and product terms, or other polynomials of $\boldsymbol{X}$.
- Recommendation: use logistic regression unless you're really sure about the normality and equality of covariances.

Some other methods for discriminant analysis are

- Regularized Discriminant Analysis (`rda` function in `klaR` library): using regularized group covariance matrices that are robust against multicollinearity in the data
- Naive Bayes Classifier (`NaiveBayes` in `klaR` library): uses estimated density for each group (insead of assuming normality)
- Flexible discriminant analysis (`fda` function in `mda` library): regerssion based classifier, captures nonlinear features of the covariates.
- Mixture discriminant analysis (`mda` function in `mda`):

Classification with more than two groups

All the methods presented for two groups can be extended for more than two groups.

Linear Discriminant Analysis:

We assume $\boldsymbol{X}|G = 1 \sim N_p(\boldsymbol{\mu}_1, \boldsymbol{\Sigma})$, $\boldsymbol{X}|G = 2 \sim N_p(\boldsymbol{\mu}_2, \boldsymbol{\Sigma})$, and $\boldsymbol{X}|G = 3 \sim N_p(\boldsymbol{\mu}_3, \boldsymbol{\Sigma})$. Linear discrimination rule (assuming all the prior probabilities are same):

$$\text{Assign a data point } \boldsymbol{x} \text{ to group 1 } (G = 1) \text{ if } \quad \frac{f_1(\boldsymbol{x})}{f_2(\boldsymbol{x})} > 1 \text{ and } \frac{f_1(\boldsymbol{x})}{f_3(\boldsymbol{x})} > 1$$

$$\text{Assign a data point } \boldsymbol{x} \text{ to group 2 } (G = 2) \text{ if } \quad \frac{f_2(\boldsymbol{x})}{f_1(\boldsymbol{x})} > 1 \text{ and } \frac{f_2(\boldsymbol{x})}{f_3(\boldsymbol{x})} > 1$$

$$\text{Assign a data point } \boldsymbol{x} \text{ to group 3 } (G = 3) \text{ otherwise}$$

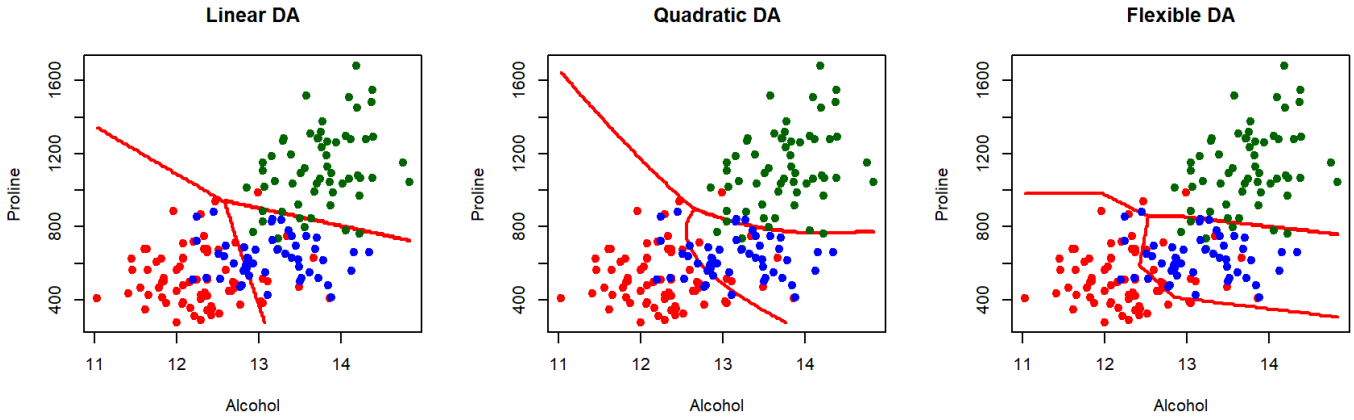
The boundaries between the regions are straight lines.

Quadratic Discriminant Analysis:

We assume $\boldsymbol{X}|G = 1 \sim N_p(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$, $\boldsymbol{X}|G = 2 \sim N_p(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$, and $\boldsymbol{X}|G = 3 \sim N_p(\boldsymbol{\mu}_3, \boldsymbol{\Sigma}_3)$.

- Same rule as in LDA but the ratios of the densities are now quadratic functions of $\boldsymbol{x}$
- The boundaries between the regions are now quadratic

We show the regions of a few classifier below.



Let us classify the `wine` data using all the covariates and using LDA.

```
#LDA
ldout <- lda(Class ~ ., data = wines)
ldout$scaling
```

```
##                  LD1           LD2
## Alcohol -0.403274956  0.8718833272
## Malic    0.165185223  0.3051811048
## Ash     -0.368792093  2.3459219420
## Alcal    0.154783909 -0.1463931519
## Mg      -0.002162757 -0.0004611477
## Phenol   0.617931702 -0.0324979420
## Flav    -1.661172871 -0.4916834144
## Nonf    -1.495756932 -1.6303752589
## Proan    0.134093115 -0.3070371492
## Color    0.355006846  0.2530559406
## Hue     -0.819785218 -1.5182643908
## Abs     -1.157612096  0.0512054337
## Proline -0.002690475  0.0028540202
```

The `scaling` field in `ldout` computes two loading vectors (linear combination of the covariates) that best classifies the data. The corresponding linear combinations are called scores (recall PCA).

```
# Scores
scores <- as.matrix(wines[ , -1]) %*% ldout$scaling
```

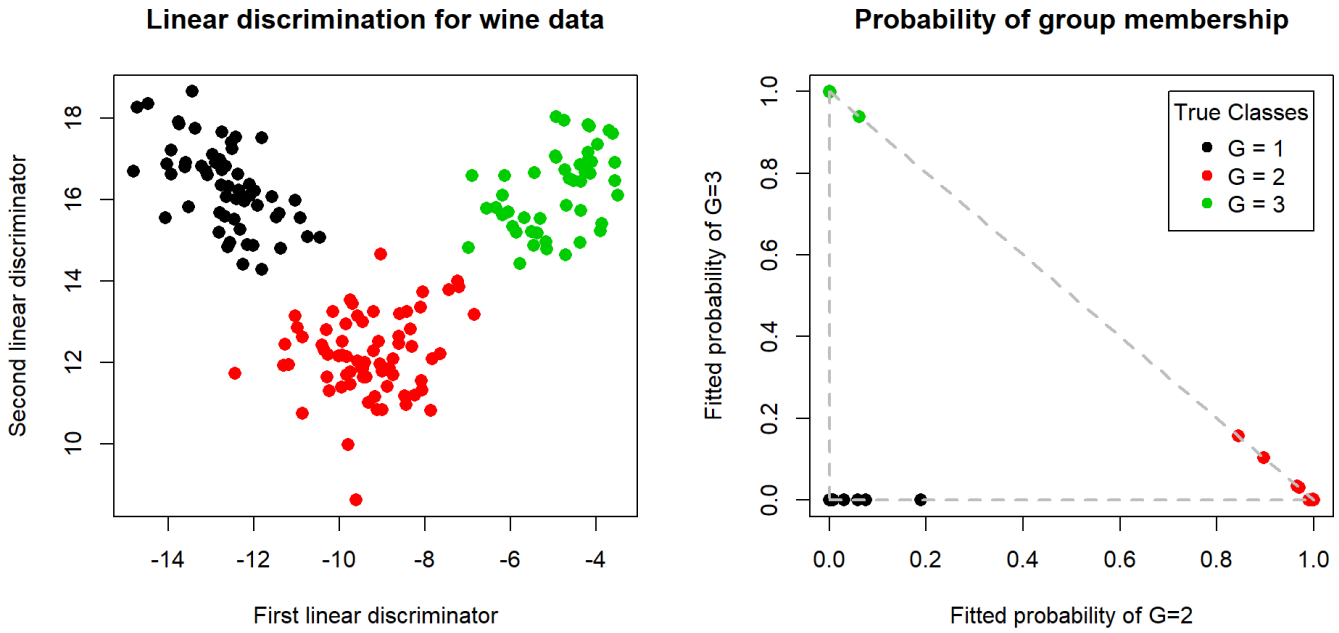
The fitted (posterior) estimated probabilities of group membership of each wine can be obtained as `predict(ldout)$posterior`.

```
probs <- predict(ldout)$posterior
head(probs)
```

```
##           1            2            3
## 1 1.0000000 3.260193e-09 3.636780e-18
## 2 0.9999996 3.582035e-07 8.720572e-17
## 3 0.9999977 2.319796e-06 7.817130e-14
## 4 1.0000000 3.720763e-12 1.335308e-16
## 5 0.9251266 7.487317e-02 2.167098e-07
## 6 1.0000000 3.506181e-11 1.291637e-17
```

Each row shows the estimated probability of group membership for this wine sample. For example, the first wine (first row), has a 100% probability of being in group 1, and thus it will be classified in group 1, and so on.

```
par(mfrow=c(1,2))
# Plot the two scores
plot(scores, col = wines[ , 1],
     pch = 16, cex = 1.25,
     xlab = "First linear discriminator",
     ylab = "Second linear discriminator",
     main = "Linear discrimination for wine data")

# Plot the group probability
plot(probs[, 2], probs[, 3], col = wines[ , 1],
     pch = 16, cex = 1.25,
     xlab = "Fitted probability of G=2",
     ylab = "Fitted probability of G=3",
     main = "Probability of group membership")
legend(0.7, 1, legend = c("G = 1", "G = 2", "G = 3"), col = 1:3, pch=19, cex=1, title = "True Classes")
lines(c(0,0), c(1,0), lwd=2, col="grey", lty=2)
lines(c(0,1), c(0,0), lwd=2, col="grey", lty=2)
lines(c(0,1), c(1,0), lwd=2, col="grey", lty=2)
```



Logistic Regression and Classification

We can extend logistic regression presented for two classes to the case of multiple classes; the regression method is called `Multinomial Logistic Regression`. We can estimate the probability of an item belonging to each class.

An item with covariate $\boldsymbol{x}$ is

> *predicted to be in class* 1 *if the estimated probability* $P(G = 1|\boldsymbol{x})$ *is larger than both* $P(G = 2|\boldsymbol{x})$ *and* $P(G = 3|\boldsymbol{x})$

> *predicted to be in class* 2 *if the estimated probability* $P(G = 2|\boldsymbol{x})$ *is larger than both* $P(G = 1|\boldsymbol{x})$ *and* $P(G = 3|\boldsymbol{x})$

predicted to be in class 3 otherwise.

```
library(nnet)

alc <- wines$Alcohol
pro <- wines$Proline
group <- as.factor(wines$Class)
data <- cbind(pro, alc)

df <- data.frame(alc = alc, pro = pro, group = group)

multilogit <- multinom(group ~ alc + pro, data = df, maxit = 200)
```

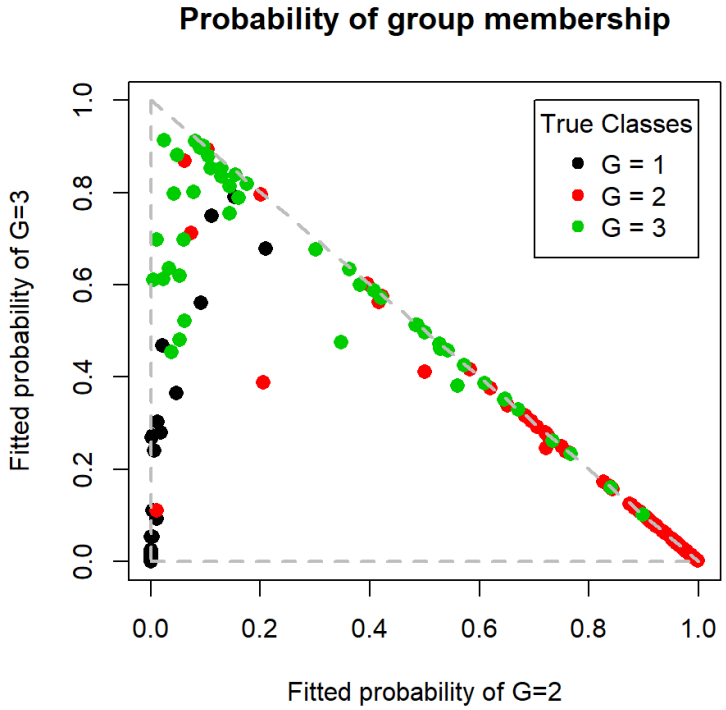
```
## # weights:  12 (6 variable)
## initial  value 195.552987
## iter  10 value 84.965497
## iter  20 value 75.874917
## iter  30 value 68.668034
## iter  40 value 67.873091
## iter  50 value 67.730849
## iter  60 value 67.688355
## final  value 67.636013
## converged
```

```
probs <- summary(multilogit)$fitted.values

head(probs)
```

```
##           1            2            3
## 1 0.9981894 3.218223e-06 1.807398e-03
## 2 0.9746092 9.157438e-04 2.447510e-02
## 3 0.9973053 5.827977e-05 2.636433e-03
## 4 0.9999990 1.613827e-10 9.763351e-07
## 5 0.1382244 1.117572e-01 7.500184e-01
## 6 0.9999976 7.567173e-10 2.429840e-06
```

Each row shows the estimated probability of group membership for this wine sample. For example, the first wine (first row), has a 99.82% probability of being in group 1, and thus it will be classified i group 1.

```
# Plot the group probability
plot(probs[, 2], probs[, 3], col = wines[ , 1],
     ylim = c(0,1), xlim = c(0,1),
     pch = 16, cex = 1.25,
     xlab = "Fitted probability of G=2",
     ylab = "Fitted probability of G=3",
     main = "Probability of group membership")
legend(0.7, 1, legend = c("G = 1", "G = 2", "G = 3"), col = 1:3, pch=19, cex=1, title = "True Classes")
lines(c(0,0), c(1,0), lwd=2, col="grey", lty=2)
lines(c(0,1), c(0,0), lwd=2, col="grey", lty=2)
lines(c(0,1), c(1,0), lwd=2, col="grey", lty=2)
```



k-Nearest-Neighbor Classifier

Basic idea: Given a observation x, find k training observations that are closest to x, and then classify using **majority vote** among these k neighbors.

- "Closest" observations are determined by some "distance" measure
- It can be applied to any objects, as long as we define a distance measure
- No probability model is assumed (completely nonparametric).
- This seemingly simple classifier works pretty well in a lot of real applications.
- The number of neighbors k is often chosen by cross-validation

```
library(class)
alc <- wines$Alcohol[wines$Class == 1 | wines$Class == 2]
pro <- wines$Proline[wines$Class == 1 | wines$Class == 2]
newclass <- wines$Class[wines$Class == 1 | wines$Class == 2]
data <- data.frame(proline = pro,
                   alcohol = alc, group = factor(newclass))

knn <- knn(train = data[,-3], test = data[,-3], cl = data$group, k=5)
errormatrix(data$group, knn)
```

```
##        predicted
## true     1  2 -SUM-
##    1    57  2     2
##    2     6 65     6
##    -SUM-  6  2     8
```

Above, we used $k = 5$ in the knn algorithm. Ideally, the number k should be chosen using cross-validation or training/test sets.

Support Vector Machine

A large number of tutorials can be found at **[http://www.svms.org/tutorials/] (http://www.svms.org/tutorials/)**

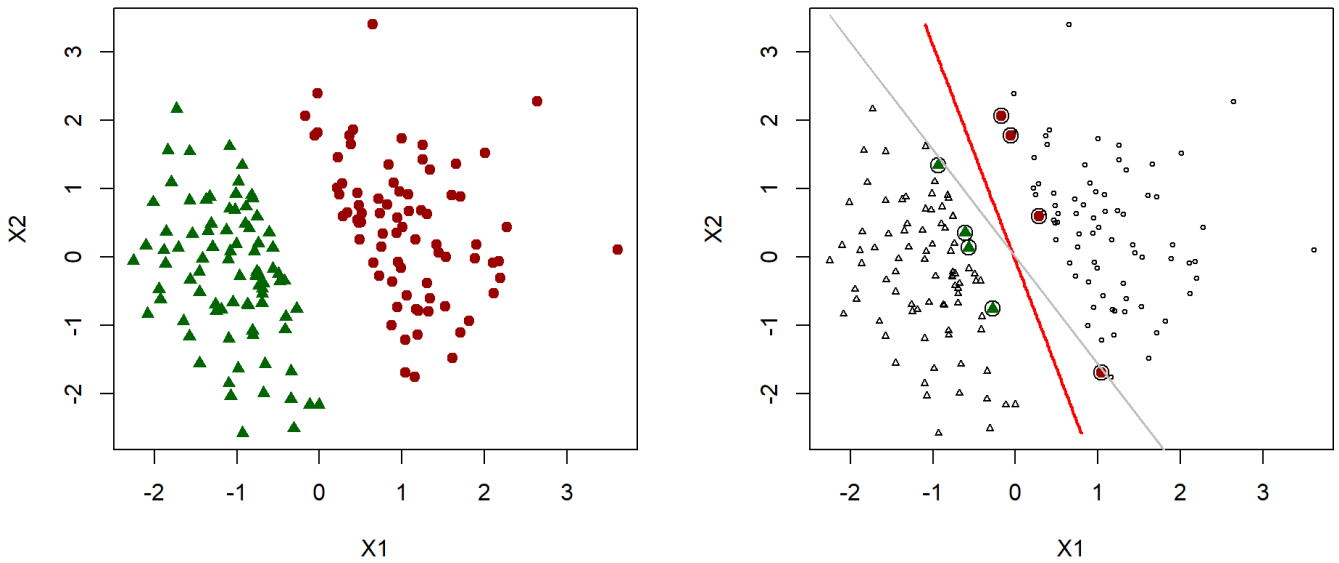
The support vector machine (SVM) is a family of classification rules that contain both parametric (e.g., linear) and nonparametric (e.g., kernel based) methods.

For a given dataset, the **support vectors** are the points that

- are closest to the "boundary" of the two classes,
- are the hardest to classify,
- have direct influence on the classification rule.

Let us take a look at the following (artificially) generated dataset. The daatset contains two predictors and observations come from two classes (red and green).

```
##  Setting default kernel parameters
```



The plot in the left panel shows the raw data. Clearly, the two classes are well separated, and a straight line can be used for classification. This situation is called "linearly separable". However, it is evident that there are many such separating lines. The plot in the right panel, shown the "support vectors" in green and red, and the optimal classification rule (red line) that *maximizes the margin* around the separating line.

Formally, for a given separating line, define

$$C_1 = \text{distance between the line and the closest point in group 1},$$

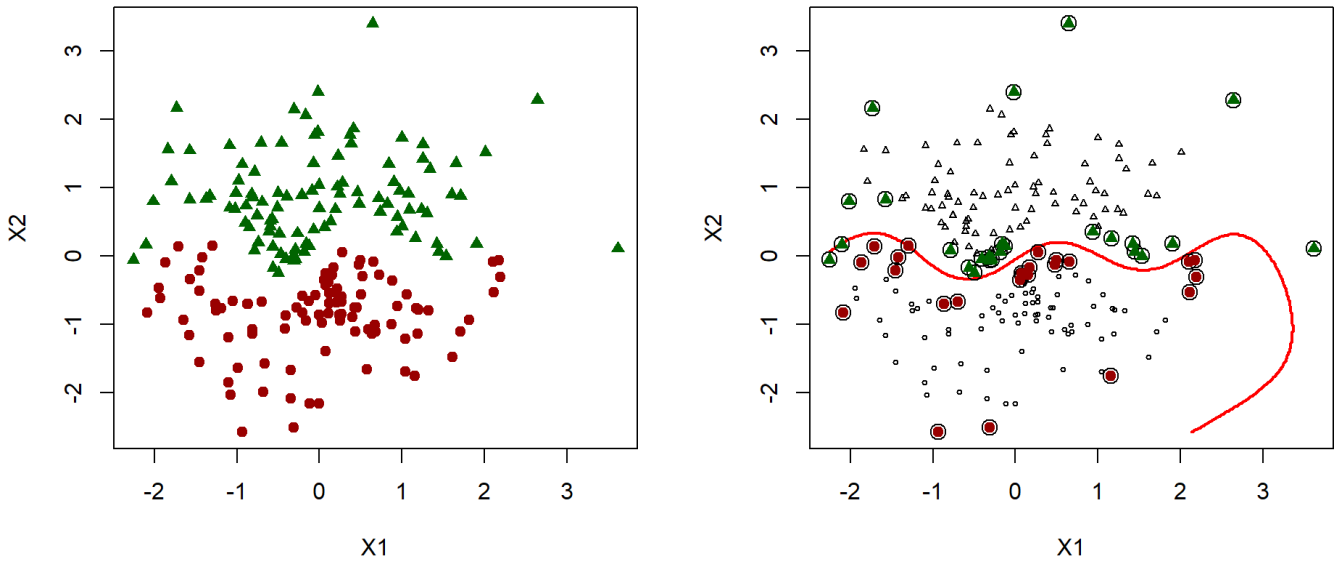
$$C_2 = \text{distance between the line and the closest point in group 2},$$

The `margin` is then defined as $C_1 + C_2$.

The optimal classification rule (separating line) maximizes the margin. It can be shown the optimal classification rule can be fully defined by only the support vectors (and hence the name of the procedure).

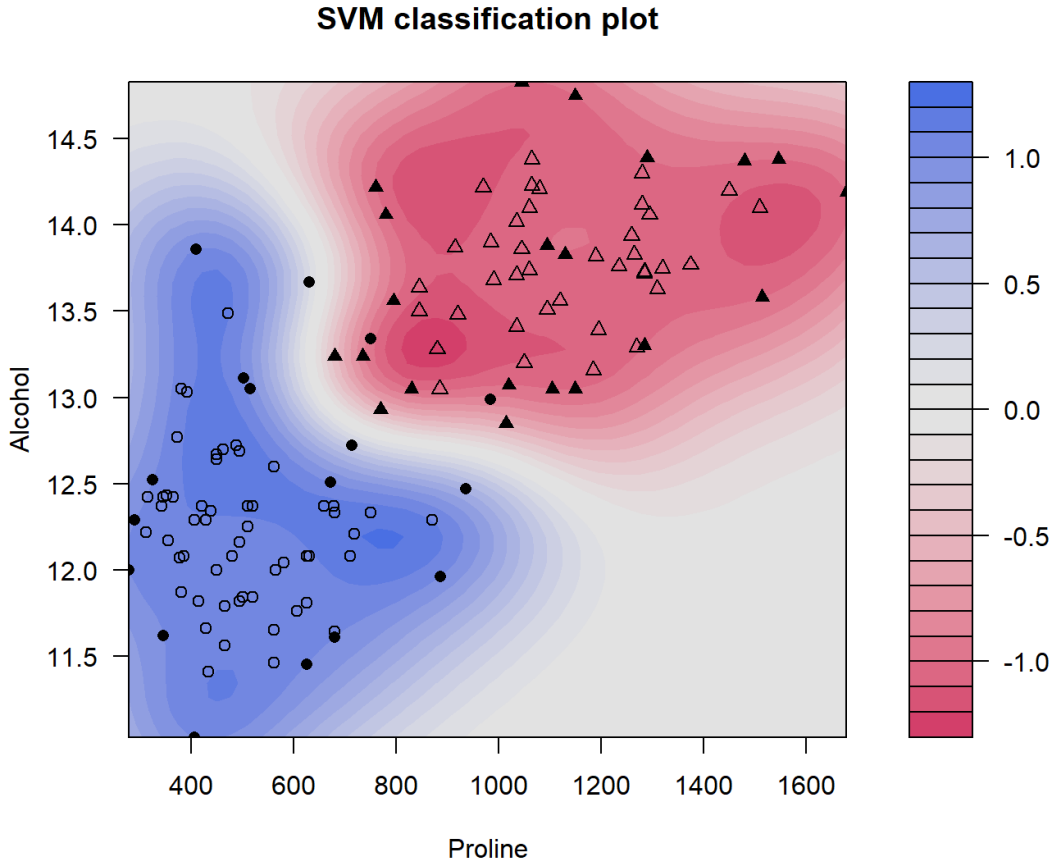
When the two classes are not linearly separable, SVM finds transformation of the data that separates the two classes well. Such transforms (often called `features`) are incorporated using `kernel` functions (mathematics omitted).

We see an (artificially generated) example of a nonlinear SVM below. The left panel shows the raw data. The right panel shows the support vectors for the two classes and the optimal classificatio rule as found by SVM.



Let us apply the SVM to our `wine` data. For better visualization we will use only two classes (1/2) and two variables (`Alcohol` and `Proline`); however, this method can be applied to any number of classes and any number of variables. The function `ksvm` in the `kernlab` library performs SVM.

```
# Two class data
data <- data.frame(Alcohol = wines$Alcohol[wines$Class ==1 | wines$Class ==2],
                   Proline = wines$Proline[wines$Class ==1 | wines$Class ==2],
                   Class = factor(wines$Class[wines$Class ==1 | wines$Class ==2]))

# SVM
library(kernlab)
sv <- ksvm(Class ~ ., data = data,
           kernel = "rbfdot", type = "C-svc")
plot(sv, data = data, xlim  = range(data$Proline))
```




```
# Confusion matrix
pred = predict(sv, data) # predicted classes
errormatrix(pred, data$Class)
```

```
##         predicted
## true      1  2 -SUM-
##    1     59  2     2
##    2      0 69     0
##    -SUM-  0  2     2
```

The argument kernel = “rbfdot” specifies that the radial basis function (for data transformation) be used to capture nonlinear features; see `?ksvm` and `?kernels` for more kernel choices. The argumemt type = “C-svc” specifies that we are solving a classification problem, as described above.

Regression Trees

Basic idea:

- Initially all objects are considered as a single group.
- The group is (binary) split into two subgroups using $X_j \geq c$ for one group and $X_j < c$ for the other group, $j = \{1, 2, \ldots, p\}$.
- Each subgroup is then (binary) split further similarly
- The splitting process stops until some stopping criterion is met.

Remark:

- Easy to interpret.
- Can handle missing data or categorical data effectively.

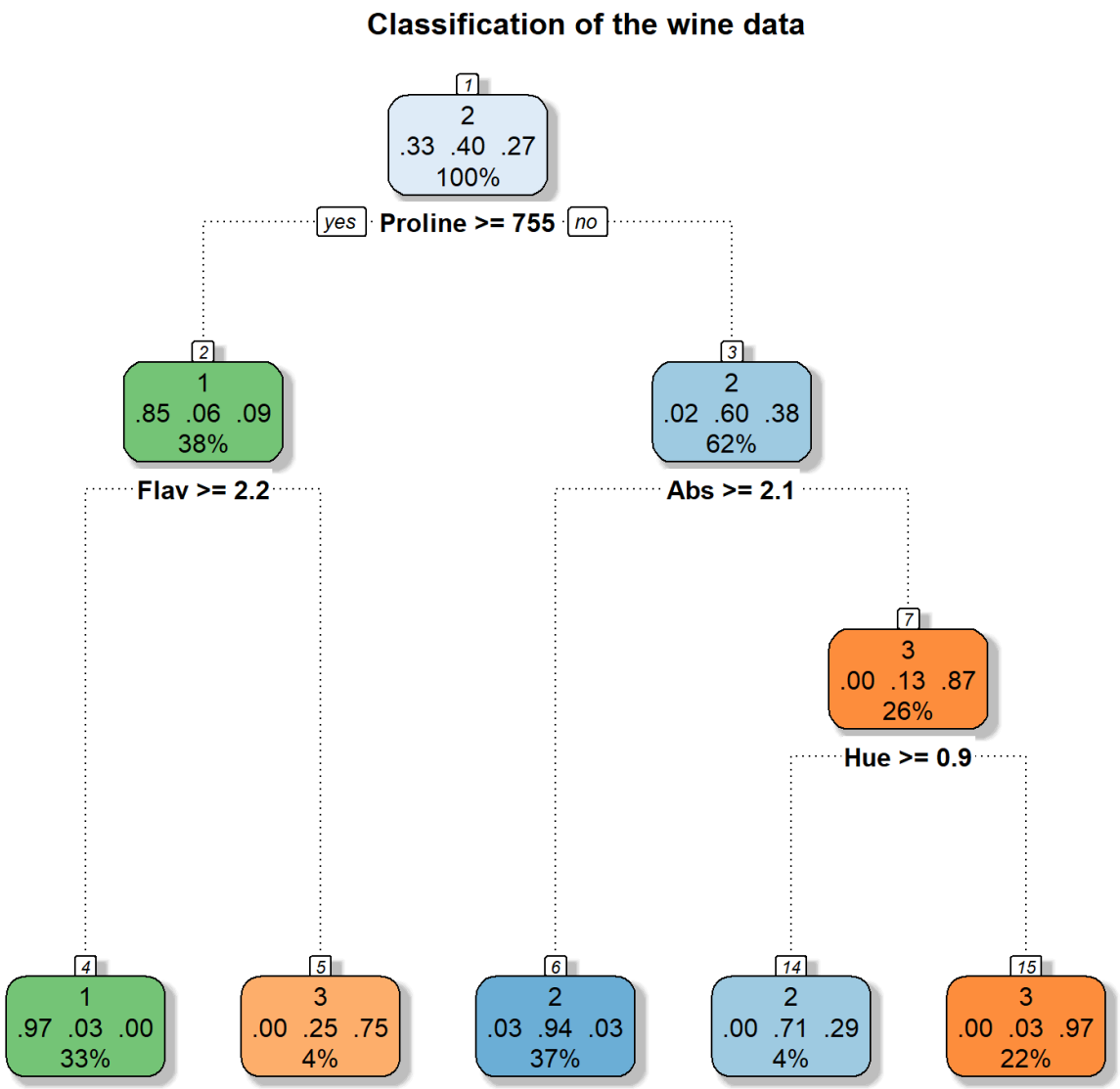
Let us consider the full wine data (with all three classes and thirteen variables). The function `rpart` function in the package with the same name can be used to fit a classification tree.

```
library(rpart)
tree <- rpart(factor(Class) ~ ., data = wines)

# Draw the tree
library(rattle)
```

```
## Warning: package 'rattle' was built under R version 3.5.2
```

```
## Rattle: A free graphical interface for data science with R.
## Version 5.2.0 Copyright (c) 2006-2018 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

```
fancyRpartPlot(tree, sub = "", main = "Classification of the wine data")
```




```
# Prediction of the class probabilities
pred <- predict(tree, wines)
head(pred)
```

```
##            1          2          3
## 1 0.96610169 0.03389831 0.00000000
## 2 0.96610169 0.03389831 0.00000000
## 3 0.96610169 0.03389831 0.00000000
## 4 0.96610169 0.03389831 0.00000000
## 5 0.03076923 0.93846154 0.03076923
## 6 0.96610169 0.03389831 0.00000000
```

```
# Prediction of class memberships
predclass <- predict(tree, wines, type = "class")
head(predclass)
```

```
## 1 2 3 4 5 6
## 1 1 1 1 2 1
## Levels: 1 2 3
```

```
# Confusion matrix
errormatrix(wines$Class, predclass)
```

```
##        predicted
## true     1  2  3 -SUM-
##   1     57  2  0     2
##   2      2 66  3     5
##   3      0  4 44     4
##   -SUM-  2  6  3    11
```

The `caret` package

The `caret` package in R contains a large number of classifiers and regression models (to date there are more than 230 models from various other packages). I provide a brief introduction to the functionality of the `caret` package.

The materials presented below and much more are available at **[https://topepo.github.io/caret/index.html] (https://topepo.github.io/caret/index.html)**.

A few visualization functions

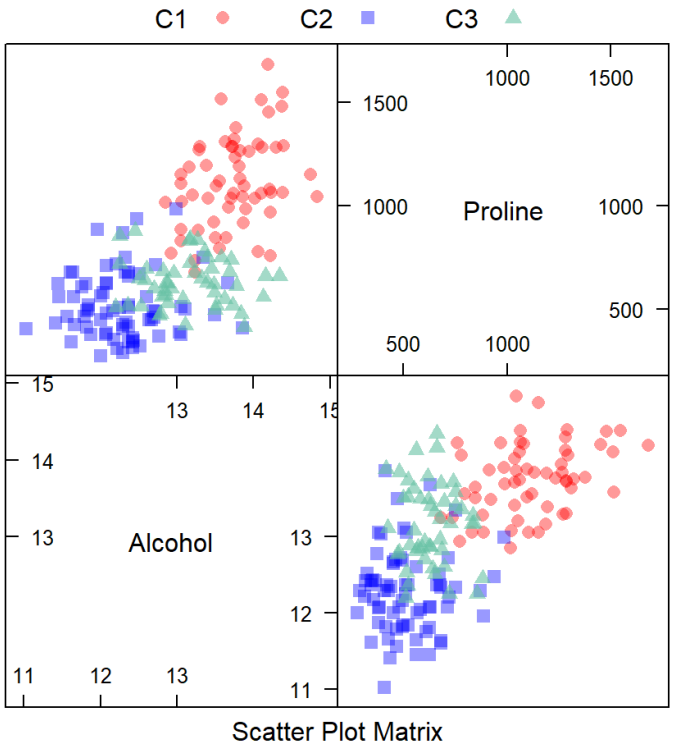
```
# Read the wine data
wines <- read.table("data/wines.txt", header = TRUE)

# Data with just Alcohol and Proline
# Classes are re-labeled as C1 -- C3
data <- data.frame(Alcohol = wines$Alcohol,
                   Proline = wines$Proline,
                   Class = factor(wines$Class, labels = c("C1", "C2", "C3")))

# Not stricly needed. Just for better visual
library(AppliedPredictiveModeling)
```

```
## Warning: package 'AppliedPredictiveModeling' was built under R version
## 3.5.2
```

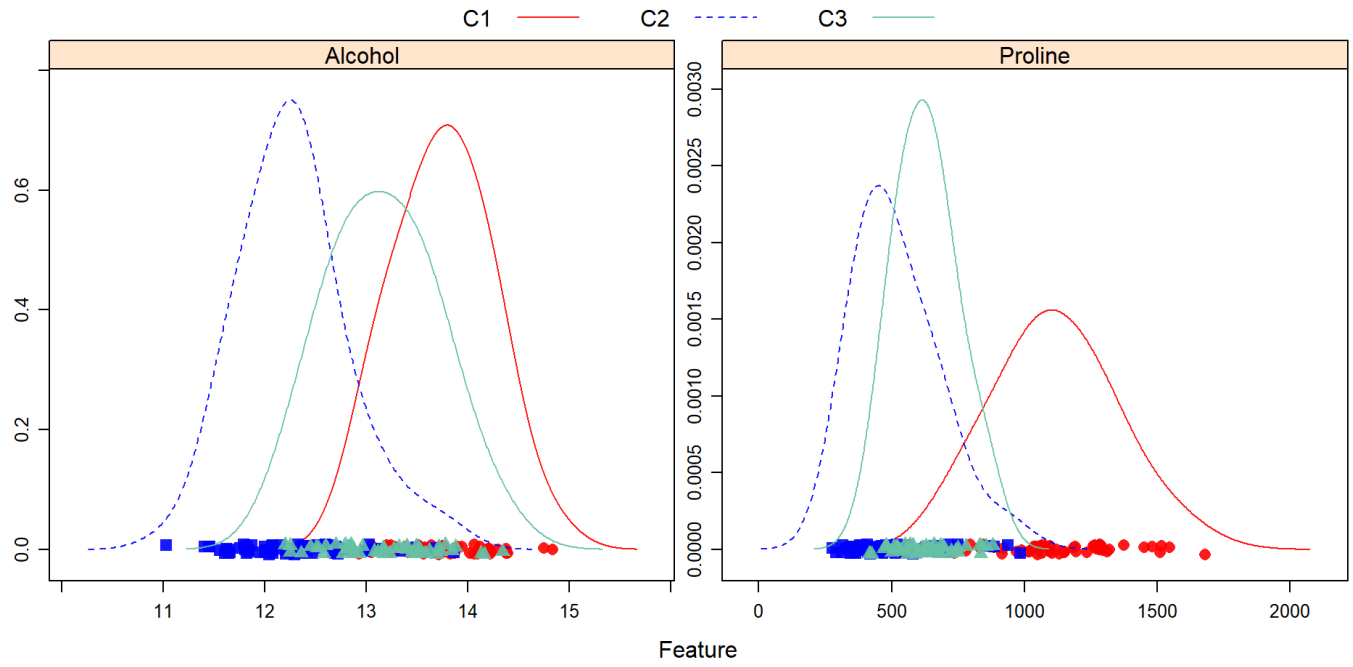
```
transparentTheme(trans = .4)

# Load library caret
library(caret)
```

The `featurePlot()` in `caret` can produce plots of various attributes or features such as simple pairs plot to overlapping densities of the various variables accross classes.

```
#library(gridExtra)

# Pairs plot
p1 <- featurePlot(x = data[, 1:2],
            y = data$Class,
            plot = "pairs",
            ## Add a key at the top
            auto.key = list(columns = 3))
p1
```




```
# Density plots
transparentTheme(trans = .9)
p2 <- featurePlot(x = data[, 1:2],
            y = data$Class,
            plot = "density",
            scales = list(x = list(relation="free"),
                          y = list(relation="free")),
            adjust = 1.5,
            ## Add a key at the top
            layout = c(2, 1),
            auto.key = list(columns = 3))
p2
```




```
# Better visual
#grid.arrange(p1, p2)
```

Splitting the dataset

The following functions can be used to split the dataset into test and training sets.

```
# For reproducibility
set.seed(1)

# indices for the training data
trainIndex <- createDataPartition(y = data$Class,
                                  p = .8,
                                  list = FALSE,
                                  times = 1)
# Training and test sets
Train <- data[ trainIndex,]
Test  <- data[-trainIndex,]
head(trainIndex)
```

```
##      Resample1
## [1,]         1
## [2,]         3
## [3,]         4
## [4,]         5
## [5,]         6
## [6,]         8
```

Notice we put the first argument `y` as a factor. Then the function does resampling from each class so that the overall class distribution of the data is preserved. The `times` argument controls how many times the resample is done.

Model training

The function `trainControl()` can pre-set training parameters. The code below, for example, sets us cross-validation parameters for the training set. We will do a 10-fold cross-validation and repeat it 30 times.

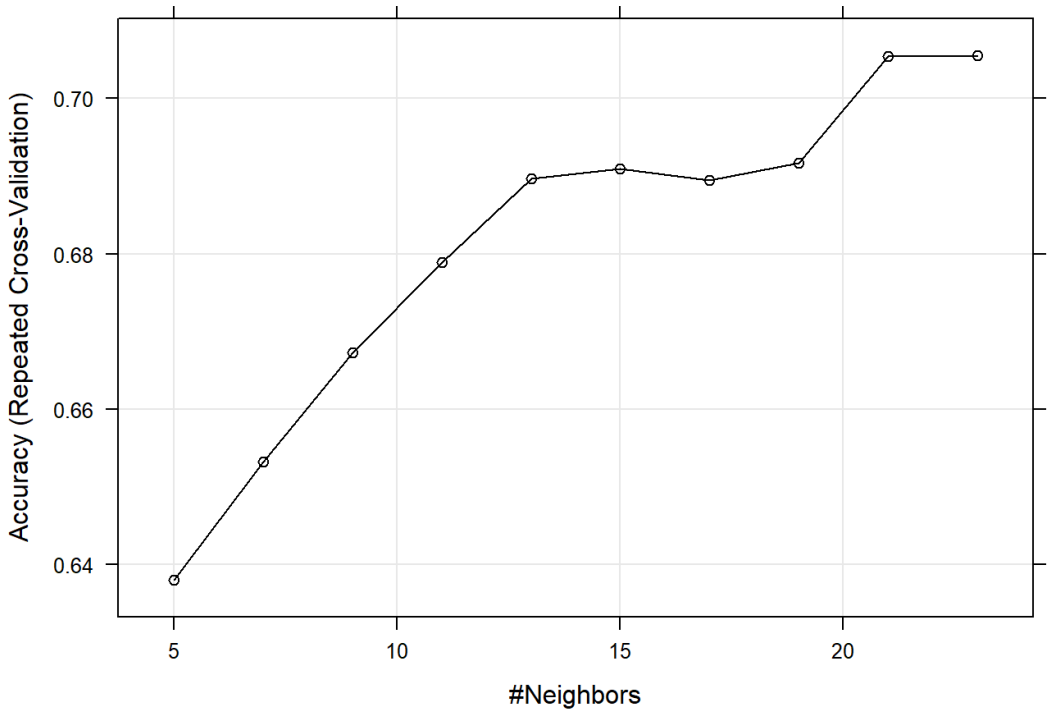
```
TrControl <- trainControl(method = "repeatedcv",
                          number = 10, #10-fold CV
                          repeats = 30 # number of repeats
)
```

The function `train` can be used to fit the classifier (can fit any model available in caret). The code below fits a KNN model. The CV (set up above in `trainControl`) procidure is automatically used to choose the best value of k in KNN using the acuuracy of the classifier.

```
Model.knn <- train(Class ~ ., data = Train,
               method = "knn",
               trControl = TrControl,
               tuneLength = 10)
Model.knn
```

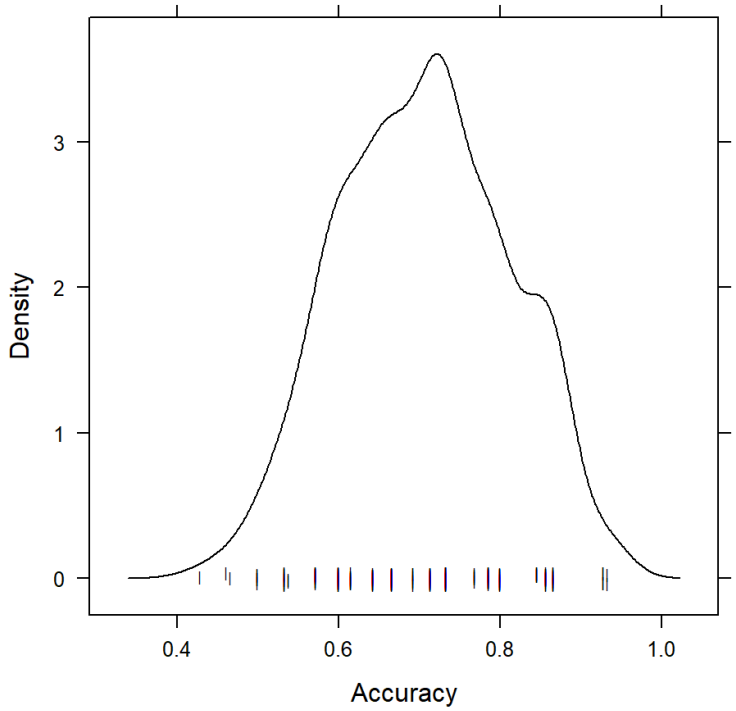
```
## k-Nearest Neighbors
##
## 144 samples
##   2 predictor
##   3 classes: 'C1', 'C2', 'C3'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 30 times)
## Summary of sample sizes: 129, 131, 131, 130, 130, 129, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    5  0.6379347  0.4494836
##    7  0.6532045  0.4757750
##    9  0.6672930  0.4967763
##   11  0.6789170  0.5149173
##   13  0.6897149  0.5326243
##   15  0.6909927  0.5344336
##   17  0.6894335  0.5326896
##   19  0.6917179  0.5363711
##   21  0.7053999  0.5568551
##   23  0.7055800  0.5568464
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 23.
```

```
plot(Model.knn)
```



We can also see the distribution of the accuracy (or other criterion) accross the cross-validation samples.

```
densityplot(Model.knn, pch = "|")
```



Comparing several models

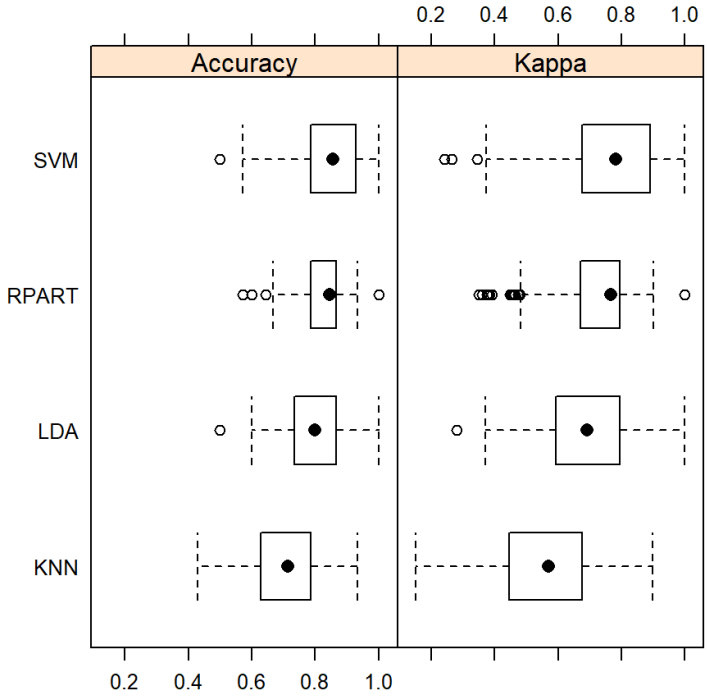
Suppose we want to compare several classifiers. We can do so as follows. Below we compare four classifiers: KNN, SVM, RPART and LDA.

```
# SVM fit
Model.svm <- train(Class ~ ., data = Train,
               method = "svmRadial",
               trControl = TrControl,
               tuneLength = 10)

# LDA fit
Model.lda <- train(Class ~ ., data = Train,
               method = "lda",
               trControl = TrControl,
               tuneLength = 10)

# Rpart fit
Model.rpart <- train(Class ~ ., data = Train,
               method = "rpart",
               trControl = TrControl,
               tuneLength = 10)

# Extract the resamples from all the four models
resamp <- resamples(list(SVM = Model.svm,
                         KNN = Model.knn,
                         LDA = Model.lda,
                         RPART = Model.rpart))
summary(resamp)
```

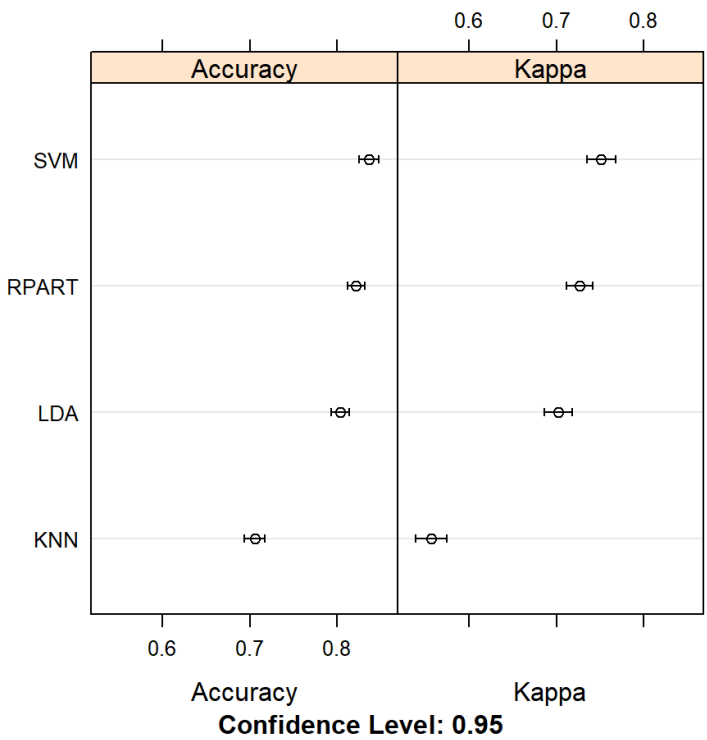
```
##
## Call:
## summary.resamples(object = resamp)
##
## Models: SVM, KNN, LDA, RPART
## Number of resamples: 300
##
## Accuracy
##            Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## SVM   0.5000000 0.7857143 0.8571429 0.8360183 0.9285714 1.0000000    0
## KNN   0.4285714 0.6359890 0.7142857 0.7055800 0.7857143 0.9333333    0
## LDA   0.5000000 0.7333333 0.8000000 0.8034548 0.8666667 1.0000000    0
## RPART 0.5714286 0.7857143 0.8461538 0.8212772 0.8666667 1.0000000    0
##
## Kappa
##            Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## SVM   0.2403101 0.6744186 0.7812500 0.7513779 0.8914729 1.0000000    0
## KNN   0.1515152 0.4444444 0.5708749 0.5568464 0.6769231 0.8993289    0
## LDA   0.2794118 0.5939051 0.6917808 0.7020791 0.7959184 1.0000000    0
## RPART 0.3488372 0.6692913 0.7651515 0.7263288 0.7959184 1.0000000    0
```

Boxplots and confidence intervals of the accuracy of the classifiers (as estimated using cross validation) are shown below.

```
bwplot(resamp)
```



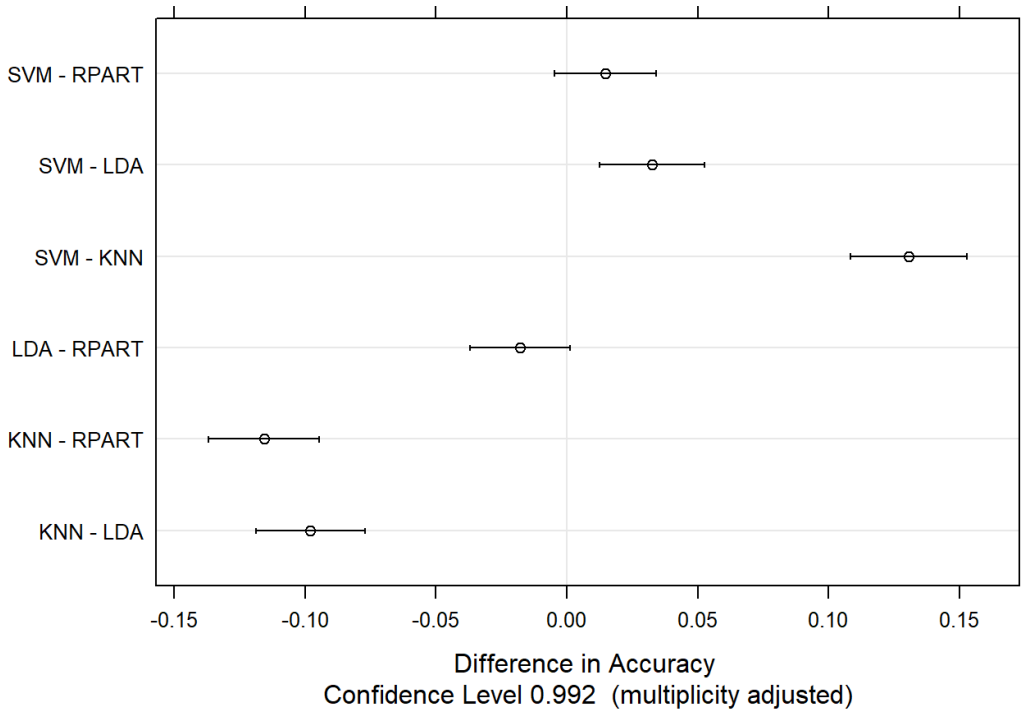

```
dotplot(resamp)
```



For this data set, it seems SVM, LDA and RPART all give about the same accuracy while KNN falls short.

We can estimate the different in accuracy as follows.

```
difValues <- diff(resamp)
dotplot(difValues)
```



The confidence intervals of the accuracy difference shows the same conclussion as before.

Performance evaluation

We can also predict the classes of the test data set, and measure the goodness of fit of any classifiers. The prediction for the SVM is givem below.

```
pred <- predict(Model.svm, Test)
postResample(pred = pred, obs = Test$Class)
```

```
##  Accuracy     Kappa
## 0.8823529 0.8215223
```

```
confusionMatrix(data = pred, reference = Test$Class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction C1 C2 C3
##         C1  9  0  0
##         C2  0 13  1
##         C3  2  1  8
##
## Overall Statistics
##
##                Accuracy : 0.8824
##                  95% CI : (0.7255, 0.967)
##     No Information Rate : 0.4118
##     P-Value [Acc > NIR] : 1.675e-08
##
##                   Kappa : 0.8215
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: C1 Class: C2 Class: C3
## Sensitivity             0.8182    0.9286    0.8889
## Specificity             1.0000    0.9500    0.8800
## Pos Pred Value          1.0000    0.9286    0.7273
## Neg Pred Value          0.9200    0.9500    0.9565
## Prevalence              0.3235    0.4118    0.2647
## Detection Rate          0.2647    0.3824    0.2353
## Detection Prevalence    0.2647    0.4118    0.3235
## Balanced Accuracy       0.9091    0.9393    0.8844
```

Overall, the `caret` package has quite a bit functionality. Please consult their documentation/tutorials [[**https://topepo.github.io/caret/index.html (https://topepo.github.io/caret/index.html)**]] for much more details as well as many other measures of goodness of fit of the classifiers.

Main page: **ST 437/537: Applied Multivariate and Longitudinal Data Analysis (https://maityst537.wordpress.ncsu.edu/)**