Some Basic Techniques in Data Mining

Distances and similarities

- The concept of **distance** is basic to human experience. In everyday life it usually means some degree of closeness of two physical objects or ideas, while the term metric is often used as a standard for a measurement.
- The mathematical meaning of distance is an abstraction of measurement.
- Distances and similarities are the most important concept that underlies upon many statistical procedures.
- We can consider distances between observations or distances between quantitative or qualitative variables.
- Basically, it is subjective to choose a similarity measure between observations or variables, because it depends of the scales of observations and variables.

Definition of Distance:

Let it be two vectors $\mathbf{x}_i$, $\mathbf{x}_j$ in $\mathbb{R}^k$, a distance is a function d with the following properties:

1. $d : \mathbb{R}^k \times \mathbb{R}^k \to \mathbb{R}^+$, namely $d(\mathbf{x}_i, \mathbf{x}_j) \geq 0$
2. $d(\mathbf{x}_i, \mathbf{x}_i) = 0 \quad \forall i$
3. $d(\mathbf{x}_i, \mathbf{x}_j) = d(\mathbf{x}_j, \mathbf{x}_i)$, distance is symmetrical.
4. $d(\mathbf{x}_i, \mathbf{x}_j) \leq d(\mathbf{x}_i, \mathbf{x}_p) + d(\mathbf{x}_p, \mathbf{x}_j)$, namely the triangular property.

These properties, which are true for distances in the familiar Euclidean geometry, are taken as the definitional characteristics (*axioms*) of the notion of distance.

One can check whether any given function that assigns a numerical value to pairs of points (or to any pair of objects) possesses these three properties.

Euclidean Distance

In Statistics, the typical approach to measure distances among observations is to use the Euclidean distance. It generates a *Metric Space*.

- Given two objects I_1 and I_2 where we consider two quantitative variables x_1 and x_2, the Euclidean distance is defined as:

$$d_{I_1 I_2} = \sqrt{(x_{11} - x_{21})^2 + (x_{12} - x_{22})^2}.$$

- If we consider more than two variables, then

$$d_{I_1 I_2} = \sqrt{\sum_{k=1}^{p} (x_{1k} - x_{2k})^2}$$

By using vectorial notation,

$$d^2_{I_i I_j} = (x_i - x_j)^t (x_i - x_j).$$

- The whole distance for n objects $i, j \in \{1, \ldots, n\}$ is

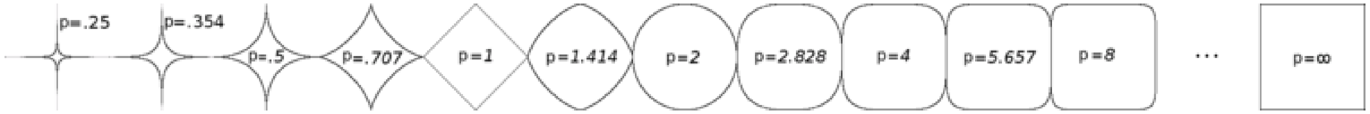
$$\mathbf{d} = \sum_{i=1}^{n} \sum_{j=1}^{n} \left(\sum_{k=1}^{p} (x_{ik} - x_{jk})^2 \right)^{1/2}.$$

- **Minkowski Distance**

$$d_{I_iI_j} = \left[\sum_k |x_{ik} - x_{jk}|^p\right]^{1/p}$$

where $p \in \mathbb{N}$.

If $p = 1$, we have the absolute value distance. If $p \to \infty$, the *Chebyshev* distance.



- **Mahalanobis Distance**

$$d^2_{I_iI_j} = (x_i - x_j)^t W^{-1} (x_i - x_j)$$

where W is the covariance matrix between the variables.

- In this way, the variables are weighted in terms of the relationship that exits between them (in terms of the *covariation*). If the covariance is 0 and the variables are standardized it is obtained the Euclidean distance.

Matching types

- We consider dichotomic variables with 0 or 1 values, *presence – absence*.
- **Example**: Consider two observations where there are 5 dichotomic variables (*yes* / *no*).

  Let Yes = 1 and No = 0

items or variables	x_1	x_2	x_3	x_4	x_5
A	1	1	0	0	1
B	0	1	0	1	0

- One possible similarity coefficient is m/N where $m =$ number of common variables in the two elements and M is the total number of variables. In this example it is $2/5$.

We can define

X_{Aj} = value of observation A in the variable *j*-th $\in \{1, 0\}$.

X_{Bj} = value of observation B in the variable *j*-th $\in \{1, 0\}$.

$V = \sum_j X_{A_j} \left(1 - X_{B_j}\right)$ **number** of attributes where A is 1 and B is 0

$R = \sum_j X_{A_j} X_{B_j}$ **number** of attributes where A and B are 1

$S = \sum_j \left(1 - X_{A_j}\right)\left(1 - X_{B_j}\right)$ **number** of attributes where A and B are 0

$T = \sum_j \left(1 - X_{A_j}\right) X_{B_j}$ **number** of attributes where A is 0 and B is 1

$U = R + S + T + V$ total **number** of attributes

Example of Matching types

- In the example,

$$
\begin{aligned}
V &= 1(1-0)+1(1-1)+0(1-0)+0(1-1)+1(1-0)=2 \\
R &= 1 \\
S &= 1 \\
T &= 1 \\
U &= 5
\end{aligned}
$$

- In this way we can define different indexes of similarity:

- **Index of Russel-Rao**

$$C = \frac{R}{U}$$

In the example: $1/5$.

- **Index of Kendall**

$$C = 1 - \frac{V+T}{U}$$

In the example: $2/5$.

- **Index of Jaccard**

$$C = \frac{R}{R+T+V}$$

  In the example: $1/4$.

- **Index of Dice–Sorensen**

$$C = \frac{2R}{2R+T+V}$$

  In the example: $2/5$.

- The most common indexes are **Jaccard** and **Dice–Sorensen**.

Gower’s Similarity Coefficient

- It is applied for mixed data types, namely, databases with continuous, ordinal or categorical variables at the same time.
- Gower’s General Similarity Coefficient S_{ij} compares two cases i and j and is defined as follows

$$S_{ij} = \frac{\sum_{k}^{n} w_{ijk} S_{ijk}}{\sum_{k}^{n} w_{ijk}}$$

- where:

  S_{ijk} denotes the contribution provided by the $k-$th variable, and

  w_{ijk} is usually 1 or 0 depending if the comparison is valid for the $k-$th variable.

Ordinal and Continuous Variables

Gower similarity defines the value of S_{ijk} for ordinal and continuous variables as follows:

$$S_{ijk} = 1 - \frac{|x_{ik} - x_{jk}|}{r_k}$$

where r_k is the **range** of values for the $k-$th variable.

Nominal Variables

The value of S_{ijk} for nominal variables is 1 if $x_{ik} = x_{jk}$ or 0 if $x_{ik} \neq x_{jk}$. Thus $S_{ijk} = 1$ if cases i and j have the same *state* for attribute k, or 0 if they have different *states*, and $w_{ijk} = 1$ if both cases have observed states for attribute k.

Binary Variables

For a binary variable (or dichotomous character), the Gower similarity defines the components of similarity and the weight according to the table,

	Value of attribute k			
Case *i*	$+$	$+$	$-$	$-$
Case *j*	$+$	$-$	$+$	$-$
S_{ijk}	1	0	0	0
w_{ijk}	1	1	1	0

where **+** denotes that attribute k is *present* and **-** denotes that attribute k is *absent*.

If all variables are binary, then Gower's similarity coefficient is equivalent to the Jaccard's similarity coefficient.

Distances between variables

- **Correlation Coefficient of Pearson**

  It is defined as

$$r = \frac{S_{xy}}{S_x S_y}$$

  where S_{xy} is the sample covariance between x and y, S_x and S_y are the standard deviations of x and y.

- **Correlation Coefficient of Kendall**

- With the Kendall tau coefficient (τ) two different rankings are compared and it has the following properties:

- If the agreement between the two rankings is perfect (i.e., the two rankings are the same) the coefficient has value 1.

- If the disagreement between the two rankings is perfect (i.e., one ranking is the reverse of the other) the coefficient has value -1.
- For all other arrangements the value lies between -1 and 1, and increasing values imply increasing agreement between the rankings. If the rankings are completely independent, the coefficient has value 0 on average.
- The coefficient can be defined as,

$$\tau = \frac{n_c - n_d}{\frac{n(n-1)}{2}}$$

- where n_c is the number of concordant pairs, and n_d is the number of discordant pairs in the data set. The denominator in the definition of τ can be interpreted as the total number of pairs of items. So, a high value in the numerator means that most pairs are concordant, indicating that the two rankings are consistent.

Correlation Coefficient of Spearman

- The Spearman correlation coefficient is often thought of as being the Pearson correlation coefficient between the ranked variables.

- The n raw scores $(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)$ are converted to ranks $(r_{x_1}, r_{y_1}), (r_{x_2}, r_{y_2}), \cdots, (r_{x_n}, r_{y_n})$, and the differences $d_i = (r_{x_i} - r_{y_i})$, between the ranks of each observation on the two variables are calculated.

$$\begin{array}{cc|cc} x_1 & y_1 & r_{x_1} & r_{y_1} \\ x_2 & y_2 & r_{x_2} & r_{y_2} \\ \vdots & \vdots & \vdots & \vdots \\ x_n & y_n & r_{x_n} & r_{y_n} \end{array}$$

- Then, the coefficient is defined as

$$r_s = 1 - \frac{6 \sum_{i=1}^{n} d_i^2}{n(n^2 - 1)}.$$

Tree Models

- They represent a compromise between *linear models* and a completely *nonparametric* approach. The methodology has roots in both the *statistics* and *computer science* literature.
- A precursor to the current methodology was `CHAID` developed by Morgan and Sonquist (1963), and Breiman et al. (1984) introduced the main ideas to statistics.
- Tree-based methods, or *decision tree methods* are especially popular in data mining, and they may be used for problem-classification and regression.
- They may be appropriate when there are extensive data, and there is uncertainty about the form in which explanatory variables ought to enter into the model.
- The methodology makes weak assumptions about the form of the regression model.

- In small data sets, it is unlikely to reveal data structure. Its strength is that, in large data sets, it has the potential to reflect relatively complex forms of structure, of a kind that may be hard to detect with conventional regression modeling.

- It is well-suited for using with big data sets.

- PROBLEMS FOR WHICH TREE-BASED REGRESSION MAY BE USED

  1. Regression with a continuous outcome variable.

  2. Binary regression.

  3. Ordered classification problems where there are more than two outcomes,

  4. Unordered classification problems.

STRENGTHS of tree-based regression:

1. Results are invariant to a *monotone re-expression* of explanatory variables.
2. The methodology is readily adapted to handle *missing values*, without omission of complete observations.
3. Tree-based regression is adapted at capturing non-additive behaviour. *Interactions* are automatically included.
4. It handles regression and, in addition, with unordered and ordered classification.
5. Results are in an immediately useful form for classification or diagnosis.

WEAKNESSES:

1. The overall tree may not be optimal. The methodology assures only that each split will be **optimal**.
2. Continuous predictor variables are treated, inefficiently, as discrete categories.
3. Low order interaction effects do not take precedence over higher order interactions.
4. Limited notions of what to look for may result in failure to find useful structure.
5. It may obscure insights that are obvious from parametric modeling.
6. Large trees make poor intuitive sense: their predictions must be used as black boxes.

Recursive partitioning regression algorithm

1. Consider the partition of an independent variable by choosing a point along the range of that variable to make the split. For each partition, we take the mean of the observations in that partition (for the corresponding elements of the leaves) and compute the residual sum of squares (*RSS*):

   $$RSS(\mathrm{partition}) = RSS(\mathrm{part1}) + RSS(\mathrm{part2})$$

   We then choose the partition that minimizes the residual sum of squares (*RSS*).

2. We now *subpartition* the partitions in a recursive manner. We only allow partitions within existing partitions and not *across* them. Partitioning can be represented using a tree and there is no restriction about splitting the same variables consecutively.

For categorical predictors, it is possible to split on the levels of the factor. For an ordered factor with L levels, there are only $L - 1$ possible splits.

EXAMPLE:

4601 email items, of which 1813 items were identified as spam (see `DAAG` library of `R`).

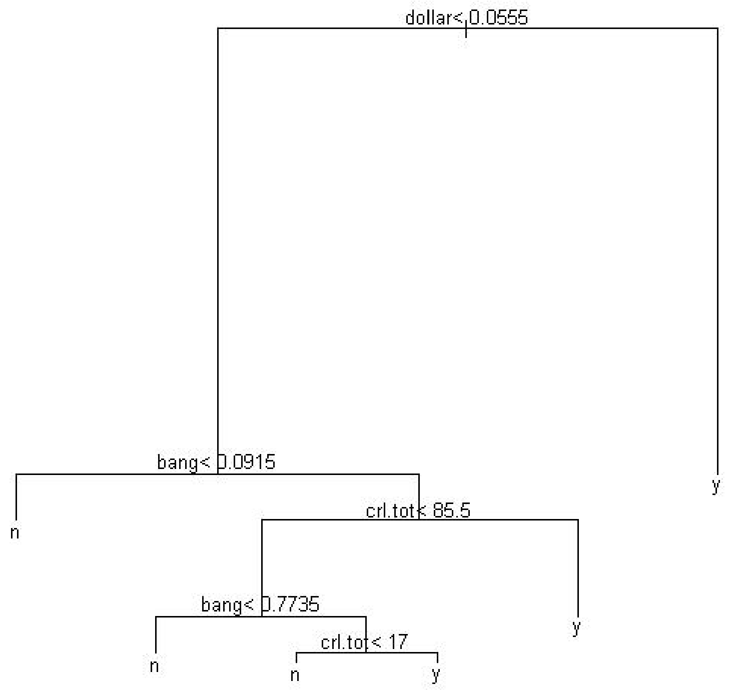
The explanatory variables are

- `crl.tot`, total length of words that are in capitals,
- `dollar`, the frequency of the `$` symbol, as a percentage of all characters,
- `bang`, the frequency of the `!` symbol, as a percentage of all characters,
- `money`, frequency of the word *money*, as a percentage of all words,
- `n000`, frequency of the text string *000*, as a percentage of all words,
- `make`, frequency of the word *make*, as a percentage of all words.

The outcome is the variable `yesno`, which is **n** for *non-spam* and **y** for *spam*.

We can consider a tree-based regression, using the 6 variables as predictors.

```
library(DAAG)

library(rpart)

data(spam7)

attach(spam7)

head(spam7)


spam.tree <- rpart(formula = yesno ~ crl.tot + dollar + bang + money +

n000 + make, method="class", data=spam7)


plot(spam.tree)

text(spam.tree)

options(digits=5)

printcp(spam.tree)
```


dollar< 0.0555
bang< 0.0915
n
crl.tot< 85.5
bang< 0.7735
n
crl.tot< 17
n
y
y
y

- Reading the tree is done as follows: If the **condition** that is specified at a node is **satisfied**, then we take the branch to the **left**. Thus, dollar (\$) $<$ 0.0555 and bang (!) $<$ 0.0915 leads to the prediction that the email is **not** *spam*.

- In *classical* regression, the inclusion of too many explanatory variables may lead to a loss of predictive power, relative to a more *parsimonious* model.

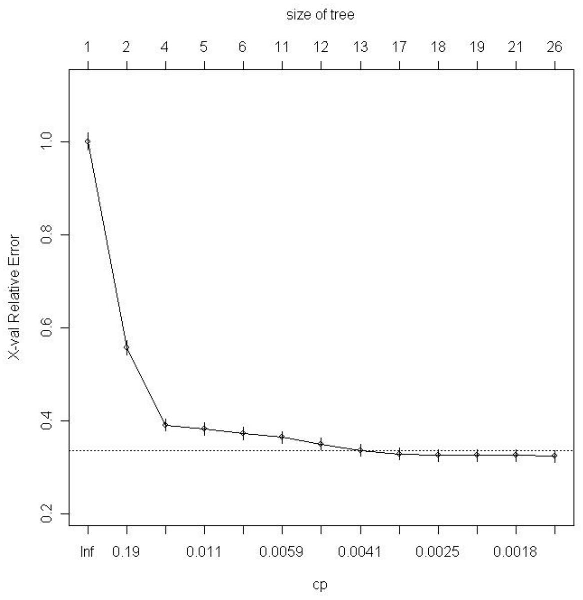
- With tree-based regression, the more important issue is the **number of splits**, rather than the number of explanatory variables.

- Key concerns are to find an *optimum tree size,* and to get an unbiased assessment of predictive accuracy.

- It is necessary to define a measure that balances the lack of fit against the tree complexity.

- In the library `rpart`, it is defined a *complexity parameter* `cp`. It plays a similar role to a *smoothing* parameter. A high value for `cp` leads to a small tree. The choice of `cp` is thus a proxy for the number of splits.
- It is taken the tree to be optimal for the given value of `cp`. The optimal tree size increases as `cp` decreases.
- Each choice of `cp` thus determines an optimal tree size. Splitting continues until the tree size is reached that is optimal for this value of `cp`.
- Given any tree, we can thus identify a sequence of prunings from the constructed tree back to the root, such that
  - at each pruning the complexity reduces,
  - each tree has the smallest number of nodes possible for that complexity, given the previous tree.

```
spam7a.tree <- rpart(formula = yesno ~ crl.tot + dollar + bang +
money + n000 + make, method="class", data=spam7, cp=0.001)


plotcp(spam7a.tree)
printcp(spam7a.tree)
```



- We can select the size of the tree by minimizing the value of the *cross-validated* error (`xerror`) and selecting the corresponding value of `cp`

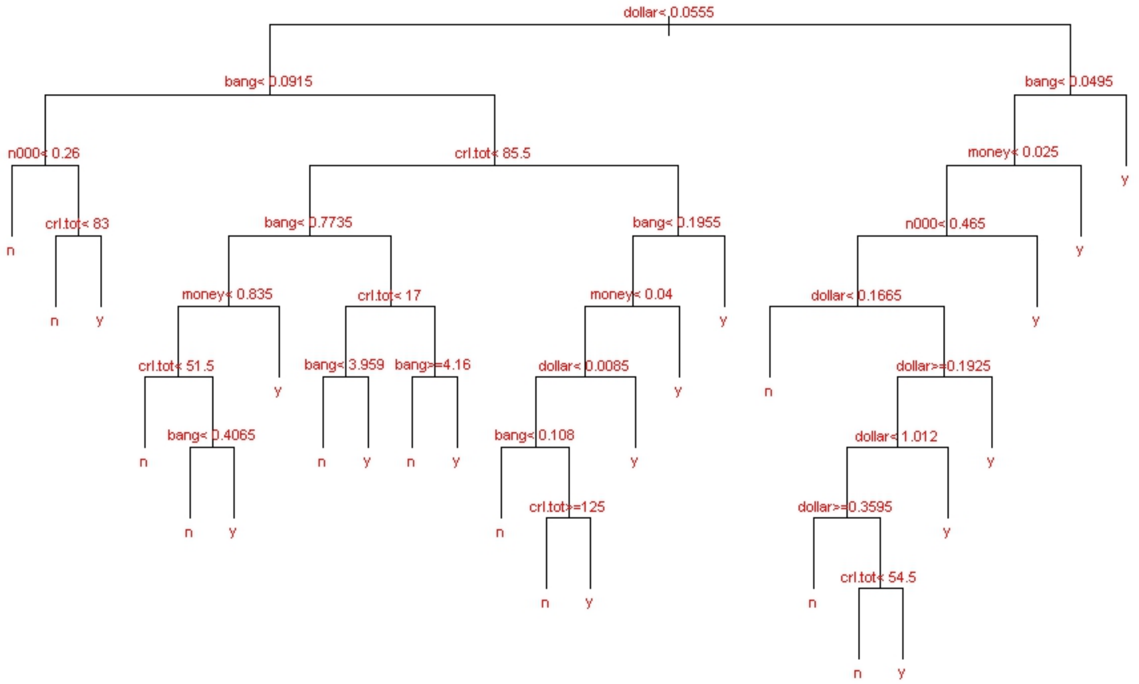
  (see `help(xpred.rpart)`).

- The process of *cross-validation* is based in this scheme:

  1. Decide on the number of folds you want (k).
  2. Subdivide your dataset into k folds.
  3. Use $k - 1$ folds for a training set to build a tree, and 1 to test the tree.
  4. Use the testing set to estimate statistics about the error in your tree.
  5. Save your results for later.
  6. Repeat steps 3-5 for k times leaving out a different fold for your test set.
  7. Average the errors across your iterations to predict the overall error.

- The cross-validated error rate estimates the expected error rate for use of the prediction tree with new data that are sampled in the same way as the original.
- We consider the *prune* of the optimal tree. Examination of the cross-validated error rate suggests that five splits may be marginally better than four splits.

```
spam7b.tree <- prune(spam7a.tree,
cp=spam7a.tree$cptable[which.min(spam7a.tree$cptable[,"xerror"]), "CP"])
plot(spam7b.tree, uniform=TRUE)
text(spam7b.tree, cex=0.7, col="red")
```


dollar< 0.0555
bang< 0.0915
bang< 0.0495
n000< 0.26
crl.tot< 85.5
money< 0.025
y
n
crl.tot< 83
bang< 0.7735
bang< 0.1955
n000< 0.465
y
n
y
money< 0.835
crl.tot< 17
money< 0.04
y
dollar< 0.1665
y
crl.tot< 51.5
y
bang< 3.959
bang>=4.16
dollar< 0.0085
y
n
dollar>=0.1925
n
bang< 0.4065
n
y
n
y
bang< 0.108
y
dollar< 1.012
y
n
y
n
crl.tot>=125
dollar>=0.3595
y
n
y
n
crl.tot< 54.5
n
y

ADDENDA. See

http://maya.cs.depaul.edu/~classes/ect584/WEKA/

http://maya.cs.depaul.edu/~classes/ect584/WEKA/classify.html

```
# Trees with WEKA (by means of R)

library(RWeka)

tree <- make_Weka_classifier("weka/classifiers/trees/J48", c("bar",  "Weka_tree"))

print(tree)

WOW(tree)


fm <- tree(yesno ~ crl.tot + dollar + bang + money + n000 + make, data=spam7,

control=Weka_control(M=150))

fm

table(observed=spam7$yesno, predicted=fitted(fm))

summary(fm)

plot(fm)
```

K means clustering

- The *k-means* clustering technique seeks to partition a set of data into a specified number of groups, k, by minimizing some numerical criterion.
- The most commonly used approach, for example, is to try to find the partition of the n individuals into k groups, which minimizes the *within-group* sum of squares over all variables.
- The problem is to consider every possible partition of the n individuals into k groups, and select the one with the lowest *within-group* sum of squares.
- The problem in practice is not so straightforward. The numbers involved are so vast that complete enumeration of every possible partition remains impossible even with the fastest computer. To illustrate the scale of the problem:

n	k	**Number of possible partitions**
15	3	2375101
20	4	45232115901
25	8	690223721118368580
100	5	10^{68}

- The impracticability of examining every possible partition has led to the development of algorithms designed to search for the minimum values of the clustering criterion by rearranging existing partitions and keeping the new one only if it provides an improvement.

- Such algorithms **do not guarantee** finding the global minimum of the criterion.

- The essential steps in these algorithms are as follows:

  1. Find some initial partition of the individuals into the required number of groups.
  2. Calculate the change in the clustering criterion produced by *moving* each individual from its own to another cluster.
  3. Make the change that leads to the greatest improvement in the value of the clustering criterion.
  4. Repeat steps (2) and (3) until no move of an individual causes the clustering criterion to improve.

- The k-means approach can be used to partition the states into a pre specified number of clusters set by the investigator.

- In practice, solutions for a range of values for number of groups are found, but the question remains as to the *optimal* number of clusters for the data.

Consider the following data: the chemical composition of 48 objects of *Romano-British* pottery, determined by atomic absorption spectrophotometry, for nine oxides.

	AL2O3	*FE2O3*	*MGO*	*CAO*	*NA2O*	*K2O*	*TIO2*	*MNO*	*BAO*
1	1.76	1.11	0.30	0.46	0.50	1.02	1.29	0.48	1.07
2	1.58	0.85	0.25	0.49	0.50	0.97	1.27	0.41	1.29
3	1.70	0.89	0.27	0.45	0.50	0.98	1.26	0.54	1.00
4	1.58	0.85	0.23	0.44	0.50	0.97	1.28	0.39	1.36
5	1.66	0.84	0.27	0.53	0.54	0.99	1.19	0.38	1.36
6	1.76	0.87	0.31	0.51	0.31	1.04	1.26	0.44	1.21
7	1.54	0.82	0.27	1.01	0.41	1.02	1.22	0.41	1.36
8	1.68	0.86	0.31	0.58	0.35	1.07	1.23	0.44	1.21
9	1.48	0.83	0.24	0.41	0.48	1.04	1.19	0.38	1.21
10	1.36	0.80	0.25	0.44	0.41	0.97	1.17	0.34	0.86
11	1.28	0.68	0.22	0.38	0.16	0.72	0.96	0.21	0.86
12	1.36	0.79	0.24	0.86	0.25	0.96	1.12	0.34	1.14
13	1.38	0.82	0.24	0.84	0.30	0.96	1.10	0.49	1.14
14	1.60	0.91	0.30	0.48	0.58	1.00	1.19	0.56	1.43
15	1.57	0.91	0.28	0.49	0.58	0.93	1.21	0.58	1.43
16	1.48	0.89	0.29	0.47	1.04	1.06	1.23	0.69	1.36
17	1.74	0.91	0.35	0.51	0.48	1.01	1.26	0.50	1.29
18	1.58	0.92	0.27	0.76	0.66	0.98	1.22	0.57	1.64
19	1.77	0.88	0.31	0.48	0.16	1.05	1.26	0.44	1.07
20	1.68	0.87	0.29	0.40	0.15	1.00	1.19	0.22	1.21
...	...	...	...	...	...	...	...	...	...

- The variables are on very different scales they will need to be standardized before applying k-means clustering.
- we divide each variable’s values by the range of the variable (max – min).

```
# pots <- read.table("pots.txt",header=T)
rge <- apply(pots,2,max)-apply(pots,2,min)
pots <- sweep(pots,2,rge,FUN="/")
```

- The k-means approach can be used to partition the states into a pre specified number of clusters set by the investigator.
- In practice, solutions for a range of values for number of groups are found.
- The question remains as to the *optimal* number of clusters for the data.

- A possibility is to examine the value of the within-group sum of squares associated with solutions for a range of values of k, the number of groups.

- As k increases this value will necessarily decrease but some *sharp* change may be indicative of the best solution.

```
# We seek for an optimal number of groups
n <- length(pots[,1])
# Compute the sum of squares within groups
# Compute the sum of squares within groups (1 only group)
scd1 <- (n-1)*sum(apply(pots,2,var))
# Compute the sum of squares within 2 to 6 groups
scd <- numeric(0)
for(i in 2:6) {
               W <- sum(kmeans(pots,i)$withinss)
               scd <- c(scd,W)   }
```

```r
# Append results of squares

scd <- c(scd1,scd)


# Plot squares within groups and number of groups

plot(1:6,scd,type="l",xlab="Number of groups",

ylab="Sum of squares within groups",lwd=2)


# Best result is with 2 or 3 groups

pots.kmedia <- kmeans(pots,3)

pots.kmedia


# Actual non standardized results

lapply(1:3,function(eso){apply(pots[pots.kmedia$cluster==eso,],2,mean)})
```

The means of each of the nine variables for each of the three clusters show that:

- Cluster **three** is characterized by a high aluminium oxide value and low iron oxide and calcium oxide values.
- Cluster **two** has a very high manganese oxide value and a high potassium oxide value.
- Cluster **one** has high calcium oxide value.
- In addition to the chemical composition of the pots, an archaeologist might be interested in assessing whether there is any association between the *site* and the distinct *compositional groups* found by the cluster analysis.

ADDENDA.

See

```
pots <- read.table("pots.txt",header=T)

rge <- apply(pots,2,max)-apply(pots,2,min)

pots <- sweep(pots,2,rge,FUN="/")


library(RWeka)

pakmean <- make_Weka_clusterer("weka/clusterers/SimpleKMeans")

print(pakmean)

WOW(pakmean)


feoK <- pakmean(pots, control=Weka_control(N=3,V=TRUE))

feoK
```

Overview of Multidimensional Scaling

- **Multidimensional scaling** (*MDS*) is a set of data analysis techniques that display the structure of *distance-like* data as a **geometrical picture**.
- MDS has its origins in *Psychometrics*, where it was proposed to help understand people's judgments of the similarity of members of a set of objects.
- MDS has now become a general data analysis technique used in a wide variety of fields like marketing, sociology, physics, political science and biology.
- MDS pictures the structure of a set of objects from data that approximate the distances between pairs of the objects. The data, which are called similarities, dissimilarities, distances or proximities, must reflect the amount of dissimilarity between pairs of observations.

- Data can be *objective* similarity measures (the driving time between pairs of cities) or an index calculated from multivariate data (the proportion of agreement in the votes cast by pairs of electors).
- Each object or event is represented by a point in a multidimensional space. The points are arranged in this space so that the distances between pairs of points have the strongest possible relation to the similarities among the pairs of objects.
- Two similar objects are represented by two points that are close together, and two dissimilar objects are represented by two points that are far apart. The space is usually a two- or three-dimensional Euclidean space, but may be non-Euclidean and may have more dimensions.

```r
loc <- cmdscale(eurodist)
x <- loc[, 1]
y <- -loc[, 2] # reflect so North is at the top
plot(x, y, type="n", xlab="", ylab="", asp=1, axes=FALSE, main="Cities of Europe")
text(x, y, rownames(loc), cex=0.6)
```

SOM (Kohonen Nets)

- The **self organizing map** (*SOM*) is an algorithm developed by Kohonen (1982-1995). It has a *vague* biological motivation, and it can be seen as a specific type of clustering algorithm.
- In the *k-means* method we saw that an observation was assigned to the cluster whose representative m_j is nearest to it.
- This is precisely what happens in SOM, but the training algorithm attempts to assign some *structure* to the representatives m_j.
- A large number of representatives are chosen, and arranged on a regular grid in one or two dimensions (both square and hexagonal grids are used).
- The idea is that the representatives (called *weights* by Kohonen) are spatially correlated, so that representatives at nearby points on the grid are more similar that those which are widely separated.

- This process is *conceptually similar* to multidimensional scaling that maps similar examples to nearby points in a q-dimensional space.

- If we were to discretize the 2-dimensional space, for example by dividing it into a grid of square bins, we would have a mapping from the space of possible examples into a discrete space that provides a clustering.

- Further, we could average the examples which are mapped to each bin to provide a representative for each non-empty bin, and the representatives in nearby bins would be similar.

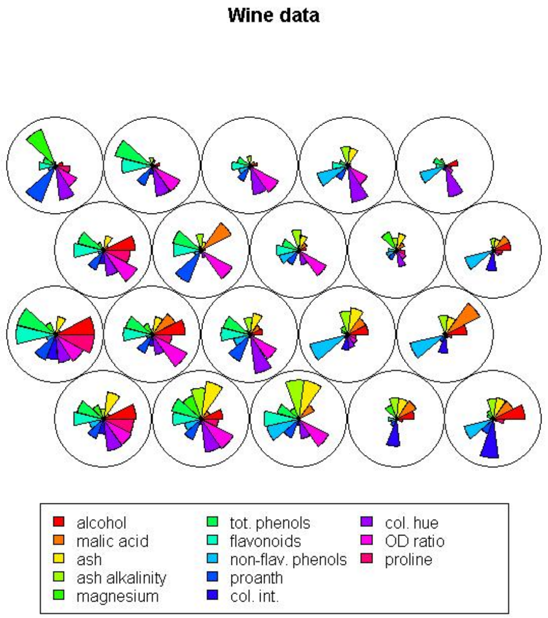
- This is precisely the *spirit* of SOM, and it is often used to provide a crude version of multidimensional scaling.

- *I just wanted an algorithm that would effectively map similar patterns (pattern vectors close to each other in the input signal space) onto contiguous locations in the output space.* (Kohonen, 1995, p. VI.)

Example

The wine data set contains information on a set of 177 Italian wine samples from three different grape cultivars; thirteen variables (such as concentrations of alcohol and flavonoids, but also color hue) have been measured.

```
library(kohonen)

data(wines)

set.seed(7)

wines.sc <- scale(wines)

wine.som <- som(data=wines.sc, grid=somgrid(5, 4, "hexagonal"))

names(wine.som)

summary(wine.som)

wine.som$unit.classif

wine.som$codes

plot(wine.som, main="Wine data")
```



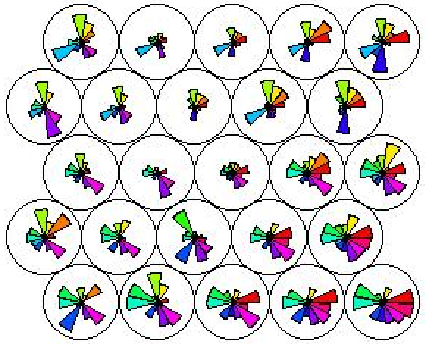
The *codebook* vectors are visualized in a segments plot. High alcohol levels, for example, are associated with wine samples projected in the bottom right corner of the map, while color intensity is largest in the bottom left corner.

Supervised SOM

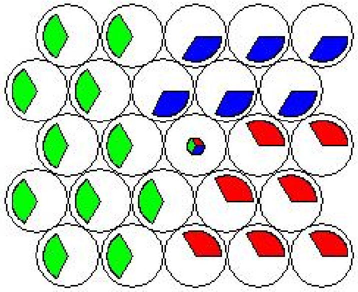
In supervised SOM there is a available dependent variable (*categorical* or *continuous*)

```
library(kohonen)

data(wines)

set.seed(7)

kohmap <- xyf(scale(wines), classvec2classmat(wine.classes),

grid = somgrid(5, 5, "hexagonal"))


par(mfrow=c(2,1))

plot(kohmap, type="codes", main=c("Codes X", "Codes Classes"))

plot(kohmap, type="counts")

plot(kohmap, type="mapping",labels=wine.classes, col=wine.classes+1,

main="Mapping Plot")
```

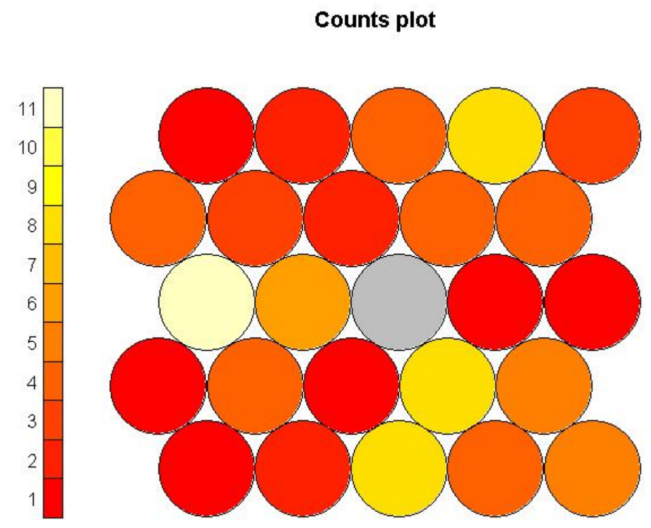
Codes X



Codes Classes

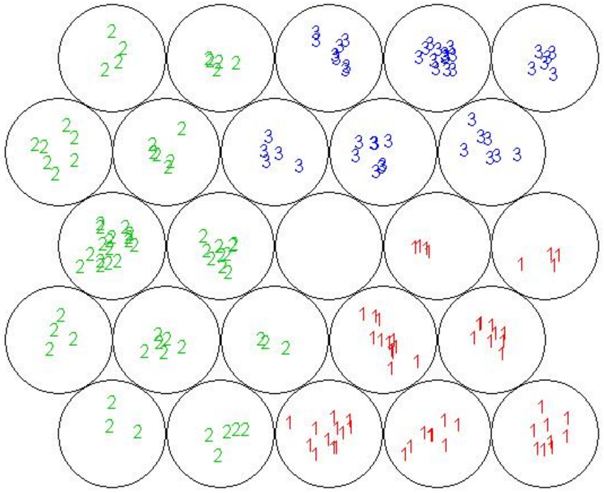


🟥 1 🟩 2 🟦 3



The background color of a unit corresponds to the number of samples mapped to that particular unit; they are reasonably spread out over the map. One of the units is empty (depicted in *gray*): no samples have been mapped to it.

Mapping Plot



The three classes of the dependent variable (the categorical class variable) are shown.