



# Classification & Clustering

# Outline

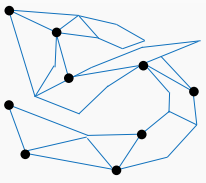
# Classification vs Clustering

# Classification

# Cross Validation

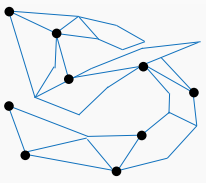
# Clustering





# Classification vs Clustering

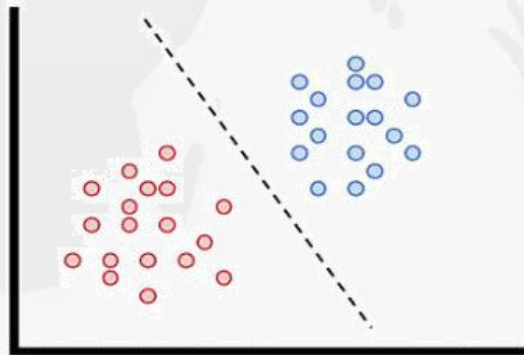
1



# Classification vs. Clustering

## *Classification:*

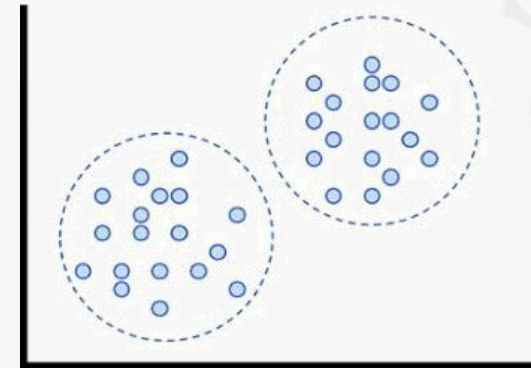
- An instance of **supervised** learning
- Labeled objects (we know to which class they belong)
- The task is to build a classifier based on the labeled objects, a model which is able to predict the class of a new object



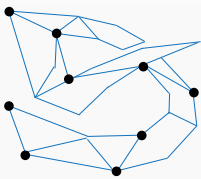
**Classification**

## *Clustering:*

- An instance of **unsupervised** learning
- We do not know to which class (group/cluster) the objects belong
- The task is to group data into categories based on some measure of inherent similarity or distance

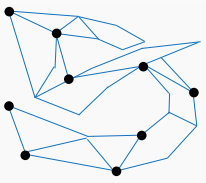


**Clustering**



# Classification

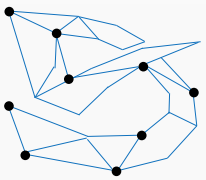
2



# Decision Tree is a General Term

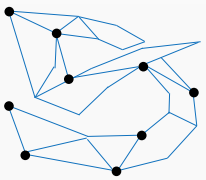
The philosophy of operation of any algorithm based on decision trees is quite simple. In fact, although sometimes containing important differences in the way to do this or that step, any algorithm of this category is based on the strategy of divide and conquer.

In general, this philosophy is based on the successive division of the problem into several subproblems with a smaller number of dimensions, until a solution for each of the simpler problems can be found. Based on this principle, the classifiers based on decision trees try to find ways to divide the universe into successively more subgroups (creating nodes containing the respective tests) until each addressing only one class or until one of the classes shows a clear majority do not justifying further divisions, generating in this situation a leaf containing the class majority. Obviously, the classification is only to follow the path dictated by the successive test placed along the tree until it found a leaf containing the class to assign to the new example.

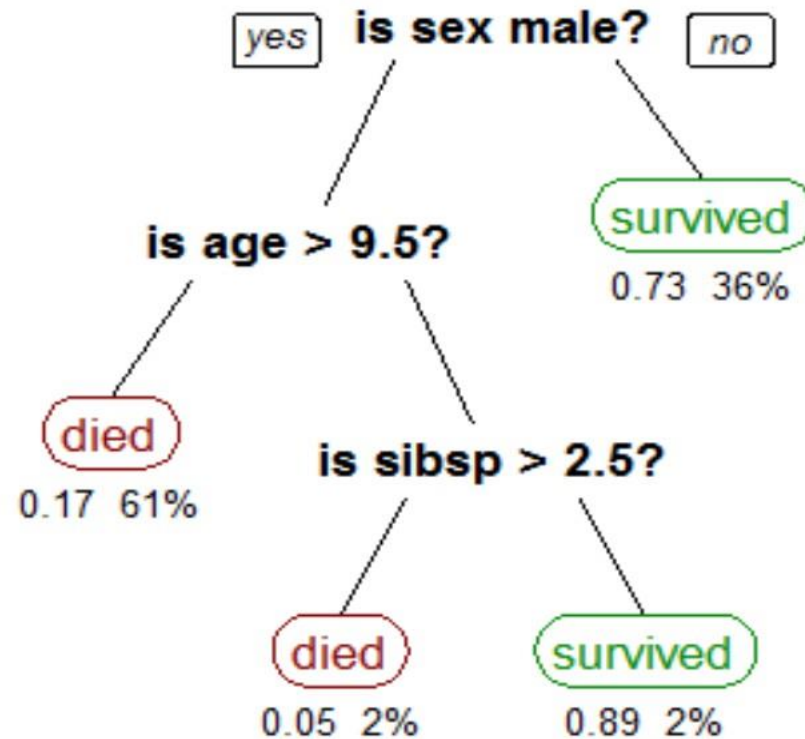


# Advantages of Decision Tree

- Can be applied to any type of data (more common in categorical data)
- The final structure of the classifier is quite simple and can be stored and handled in a graceful manner
- Result is normally robust in the training set
- The resulting trees are usually quite understandable and can be easily used to obtain a better understanding of the phenomenon in question

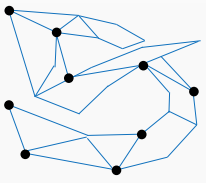


# Decision Tree

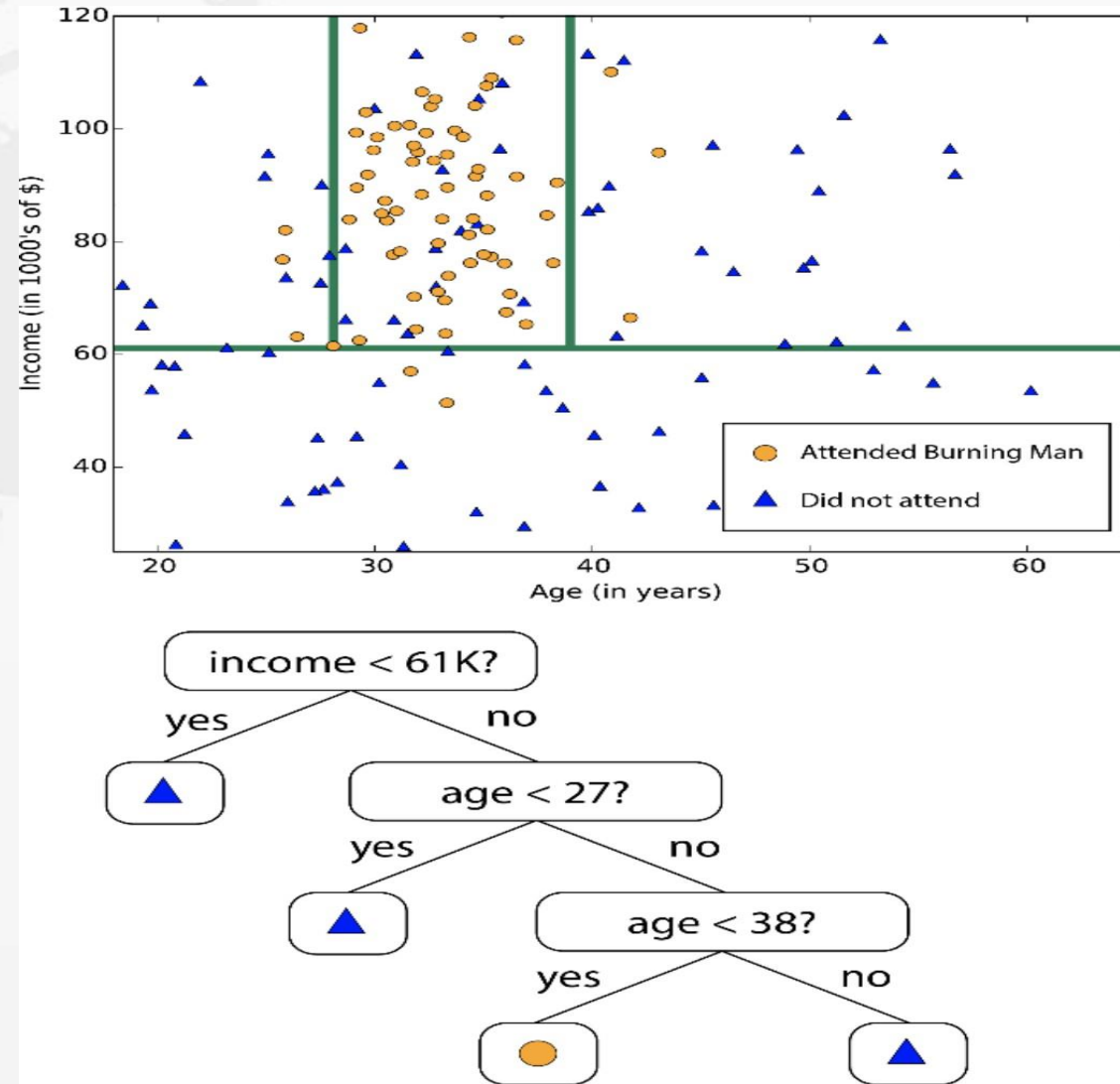


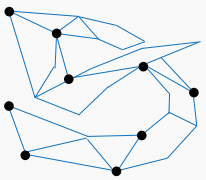
A tree showing survival of passengers on the [Titanic](#) ("sibsp" is the number of spouses or siblings aboard). The figures under the leaves show the probability of outcome and the percentage of observations in the leaf.





# Decision Tree





# Where to Split

We now need an objective criteria for judging how good a split is. The information gain measure is used to select the test attribute at each node in the tree. The attribute with the highest information gain (or Gini impurity) is chosen as the test attribute for the current node. This attribute minimizes the information needed to classify the samples in the resulting partitions.

## 1. Information Entropy

Generally, entropy refers to disorder or uncertainty. More predictable event will have low information entropy.

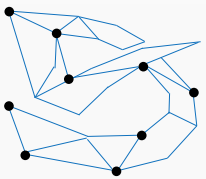
$$H(X) = E(-\ln(P(X))) = - \sum (P(X_i) * \log_2(P(X_i)))$$

## 2. Information Gain

\$ Information Gain = H (new) - H (old)\$ where H represents the information entropy.

## 3. Gini Impurity

$$\text{Gini Index} = \sum (P_i * (1 - P_i))$$



# Decision Tree is a Two-Step Process

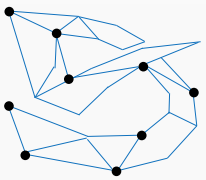
## **Step 1: Grow the tree (tree induction)**

Tree Growing is the task of taking a set of labeled data as input, deciding which attributes (or variable) are best to split on, splitting the dataset, and recursing on the resulting split datasets until all training instances are categorized. After each split, checking if the data fell into that branch is pure, if yes, then stop; if no, then continue split.

## **Step 2: Prune the tree**

Because a feature space can be subdivided into arbitrarily small regions, it's easy to imagine dividing it finely enough to have one data point per region. This is an extreme example of overfitting.

A completed decision tree model can be overly-complex, contain unnecessary structure, and be difficult to interpret. Tree pruning is the process of removing the unnecessary structure from a decision tree in order to make it more efficient, more easily-readable for humans, and more accurate as well.



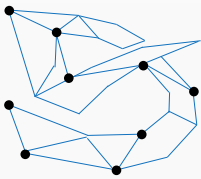
# Decision Tree R demo

```
# install.packages("ISLR")  
# install.packages("rpart")  
# install.packages("rpart.plot")  
require(ISLR)
```

```
## Loading required package: ISLR  
require(rpart)
```

```
## Loading required package: rpart  
require(rpart.plot)
```

```
## Loading required package:  
rpart.plot
```



```
### Load data
data(Carseats)
```

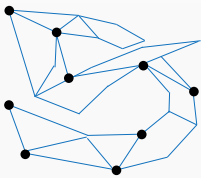
```
### Data Exploration
head(Carseats)
```

```
## 1  9.50      138      73      11      276  120      Bad  42
## 2 11.22      111      48      16      260   83     Good  65
## 3 10.06      113      35      10      269   80    Medium  59
## 4  7.40      117     100       4      466   97    Medium  55
## 5  4.15      141      64       3      340  128      Bad  38
## 6 10.81      124     113      13      501   72      Bad  78
```

```
## Education Urban US
## 1      17  Yes Yes
## 2      10  Yes Yes
## 3      12  Yes Yes
## 4      14  Yes Yes
## 5      13  Yes No
## 6      16   No Yes
```

```
str(Carseats)
```

```
## 'data.frame':   400 obs. of  11 variables:
## $ Sales       : num  9.5 11.22 10.06 7.4 4.15 ...
## $ CompPrice   : num  138 111 113 117 141 124 115 136 132 132 ...
## $ Income      : num   73 48 35 100 64 113 105 81 110 113 ...
## $ Advertising : num   11 16 10 4 3 13 0 15 0 0 ...
## $ Population. : num  276 260 269 466 340 501 45 425 108 131 ...
## $ Price       : num  120 83 80 97 128 72 108 120 124 124 ...
## $ ShelfLoc    : Factor w/ 3 levels "Bad","Good","Medium": 1 2 3 3 1 1 3 2 3 3 ...
## $ Age         : num   42 65 59 55 38 78 71 67 76 76 ...
## $ Education   : num   17 10 12 14 13 16 15 10 10 17 ...
## $ Urban       : Factor w/ 2 levels "No","Yes": 2 2 2 2 2 1 2 2 1 1 ...
## $ US          : Factor w/ 2 levels "No","Yes": 2 2 2 2 1 2 1 2 1 2 ...
```



```
## Median :14.0
## Mean :13.9 ## 3rd
Qu.:16.0 ## Max.
:18.0
table(is.na(Carseats))

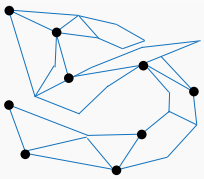
##
## FALSE ##
4400

## Create a categorical variables based on sales
Carseats$high = ifelse(Carseats$Sales>8,"Y","N") Carseats
=Carseats[, -1]

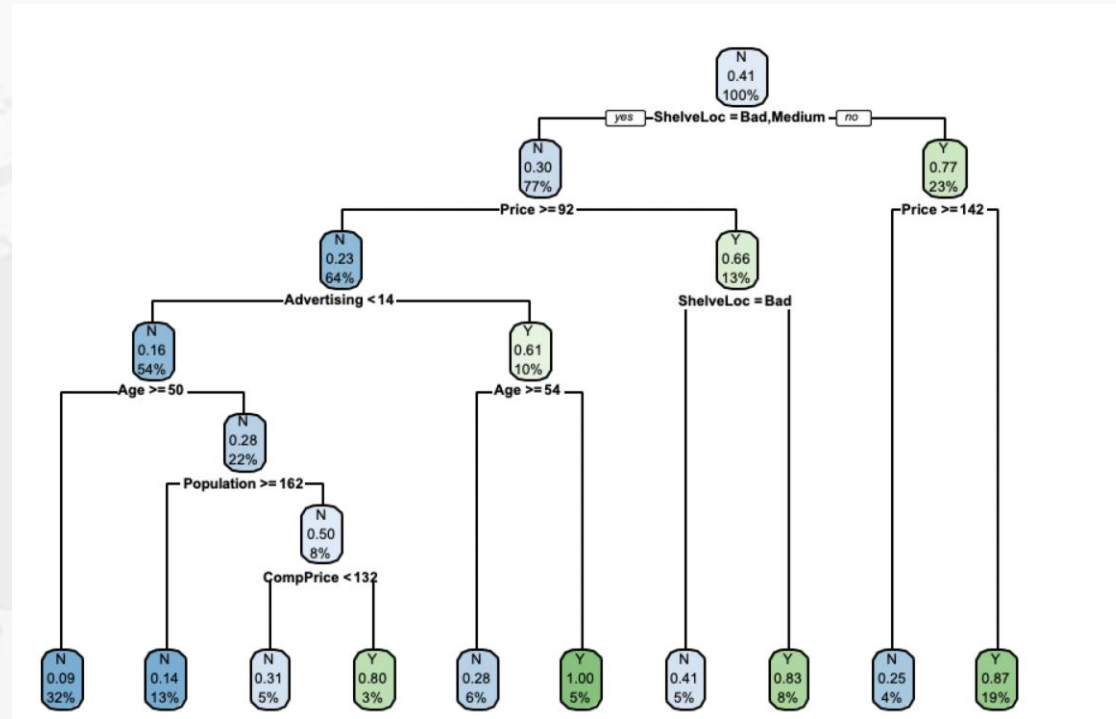
### Split data into train and test
set.seed(2)
train = sample(1:nrow(Carseats), nrow(Carseats)*0.8) test = -
train
training_data =Carseats[train,]
testing_data =Carseats[test,]

### Grow a tree on train
par(mfrow=c(1,1))
fit <- rpart(high ~.,data=training_data,method="class") fit

## n= 320 ##
## node), split, n, loss, yval, (yprob)
## * denotes terminal node ##
## 1) root 320 132 N 0.58750000 0.41250000)
## 2) ShelfLoc=Bad,Medium 246 75 N 0.69512195 0.30487805)
## 4) Price>=92.5 205 48 N 0.76585366 0.23414634)
## 8) Advertising< 13.5 172 28 N 0.83720930 0.16279070)
## 16) Age>=49.5 103 9 N 0.91262136 0.08737864) *
## 17) Age<49.5 69 19 N 0.72463768 0.27536232)
## 34) Population>=162 43 6 N 0.86046512 0.13953488) *
## 35) Population< 162 26 13 N 0.50000000 0.50000000)
## 70) CompPrice< 131.5 16 5 N 0.68750000 0.31250000) *
## 71) CompPrice>=131.5 10 2 Y 0.20000000 0.80000000) *
## 9) Advertising>=13.5 33 13 Y 0.39393939 0.60606061)
## 18) Age>=54.5 18 5 N 0.72222222 0.27777778) *
```



`rpart.plot(fit)`



```
### Check how the model is doing using the test
pred = predict(fit, testing_data, type="class")
table = table(testing_data$high, pred) #confusion matrix
table
```

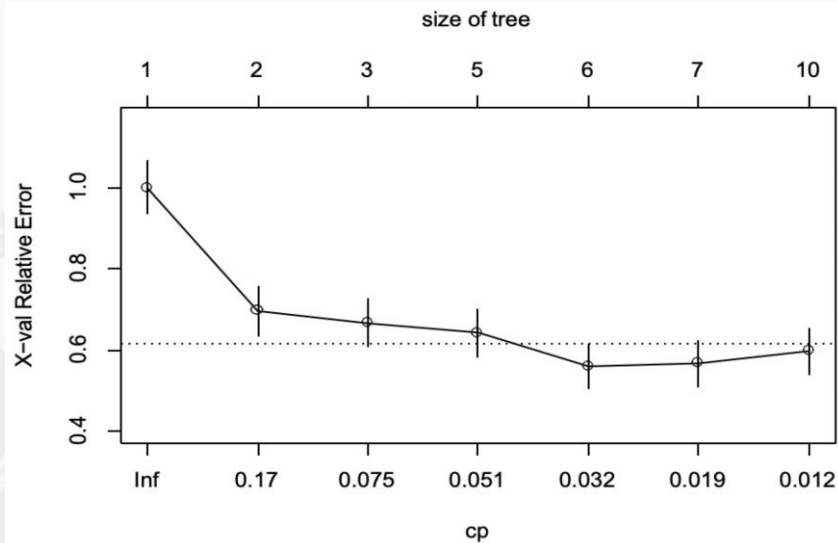
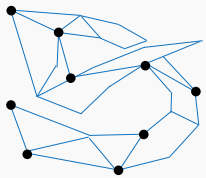
```
##      pred
##      N  Y
##  N 38 10
##  Y 13 19
```

```
1 - sum(diag(table)) / sum(table) #misclassification rate: 28.75
```

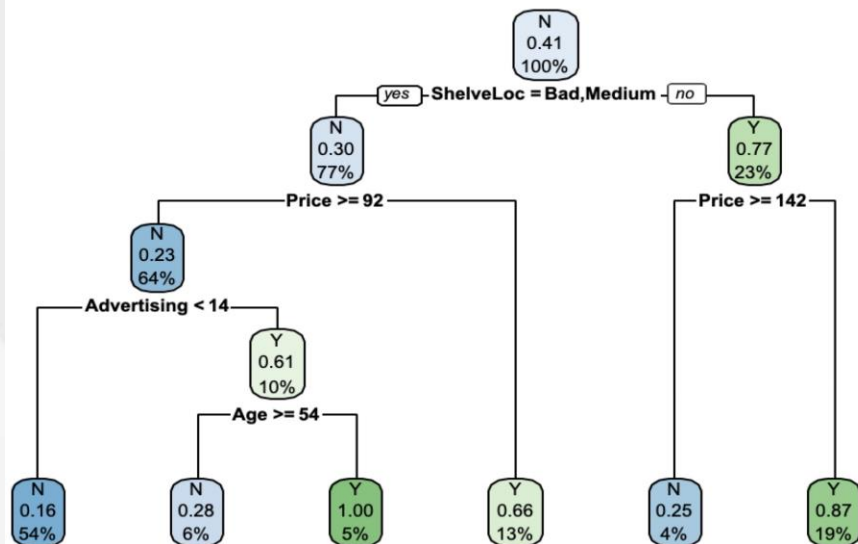
```
## [1] 0.2875
```

```
### pruning the tree
```

```
plotcp(fit)
```

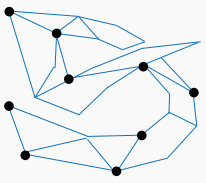


```
# printcp(fit)
fit2 =prune.rpart(fit,cp=0.032)
rpart.plot(fit2)
```



```
### Check how the model is doing using the test
pred2 =predict(fit2, testing_data,type="class")
table2 =table(testing_data$high,pred2) #confusion matrix
table2
## pred2
##      N Y
##  N 41  7
##   Y 13 19
1-sum(diag(table2))/sum(table2) #misclassification rate:25
## [1] 0.25
```



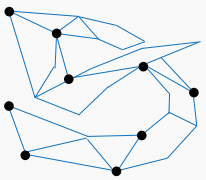


# Random Forest

What is the difference between random forest and decision tree? The simple answer is that a random forest is a collection or ensemble of decision trees.

The point of RF is to prevent overfitting. It does this by creating random subsets of the features and building smaller (shallow) trees using the subsets and then it combines the subtrees. The downside of RF is it can be slow if you have a single process but it can be parallelized.

A decision tree is built using the whole dataset considering all features, but in random forests a fraction of the number of rows is selected at random and a particular number of features are selected at random to train on and a decision tree is built on this subset.



# Random Forest R demo

```
### Load Data
#install.packages("MASS") #Package which contains the Boston Housing dataset
require(MASS)

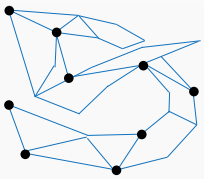
## Loading required package: MASS
require(randomForest)

## Loading required package: randomForest
## randomForest 4.6-12

## Type rfNews() to see new features/changes/bug fixes.

data(Boston)
?Boston
#head(Boston)
str(Boston)

## 'data.frame':  506 obs. of  14 variables:
## $ crim   : num  0.00632 0.02731 0.02729 0.03237 0.06905 ...
## $ zn     : num  18 0 0 0 0 12.5 12.5 12.5 12.5 ...
## $ indus  : num  2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7.87 ...
## $ chas   : int   0 0 0 0 0 0 0 0 0 ...
## $ nox    : num  0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.524 ...
## $ rm     : num  6.58 6.42 7.18 7 7.15 ...
```



```
## $ age : num 65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9 ...
## $ dis : num 4.09 4.97 4.97 6.06 6.06 ...
## $ rad : int 1 2 2 3 3 3 5 5 5 5 ...
## $ tax : num 296 242 242 222 222 222 311 311 311 311 ...
## $ ptratio: num 15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 15.2 ...
## $ black : num 397 397 393 395 397 ...
## $ lstat : num 4.98 9.14 4.03 2.94 5.33 ...
## $ medv : num 24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 ...
```

```
summary(Boston$medv)
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      5.00   17.02   21.20   22.53   25.00   50.00
```

```
### Create a new variable: level
```

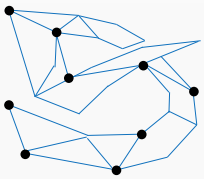
```
Boston$level = as.factor(ifelse(Boston$medv>=24,"high",
                                ifelse(Boston$medv<=19,"low","normal")))
```

```
table(Boston$level)
```

```
##
##   high low normal
##   157 175   174
```

```
str(Boston)
```

```
## 'data.frame': 506 obs. of 15 variables:
## $ crim : num 0.00632 0.02731 0.02729 0.03237 0.06905 ...
## $ zn : num 18 0 0 0 0 12.5 12.5 12.5 12.5 ...
## $ indus : num 2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7.87 ...
## $ chas : int 0 0 0 0 0 0 0 0 0 ...
## $ nox : num 0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.524 ...
## $ rm : num 6.58 6.42 7.18 7 7.15 ...
## $ age : num 65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9 ...
## $ dis : num 4.09 4.97 4.97 6.06 6.06 ...
## $ rad : int 1 2 2 3 3 3 5 5 5 5 ...
## $ tax : num 296 242 242 222 222 222 311 311 311 311 ...
## $ ptratio: num 15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 15.2 ...
## $ black : num 397 397 393 395 397 ...
## $ lstat : num 4.98 9.14 4.03 2.94 5.33 ...
## $ medv : num 24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 ...
## $ level : Factor w/ 3 levels "high","low","normal": 1 3 1 1 1 1 3 1 2 2 ...
```



```
### Split train and test
set.seed(123)
samp = sample(1:nrow(Boston), 0.7 * nrow(Boston))
train <- Boston[samp, ]
test <- Boston[-samp, ]

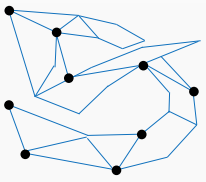
### Build random forest classification model
rf = randomForest(level ~. - medv, data = train)
rf # oob error is 20.9 (misclassification rate)

##
## Call:
## randomForest(formula = level ~. - medv, data = train)
## Type of random forest: classification
## Number of trees: 500
## No. of variables tried at each split: 3
##
## OOB estimate of error rate: 20.9%
## Confusion matrix:

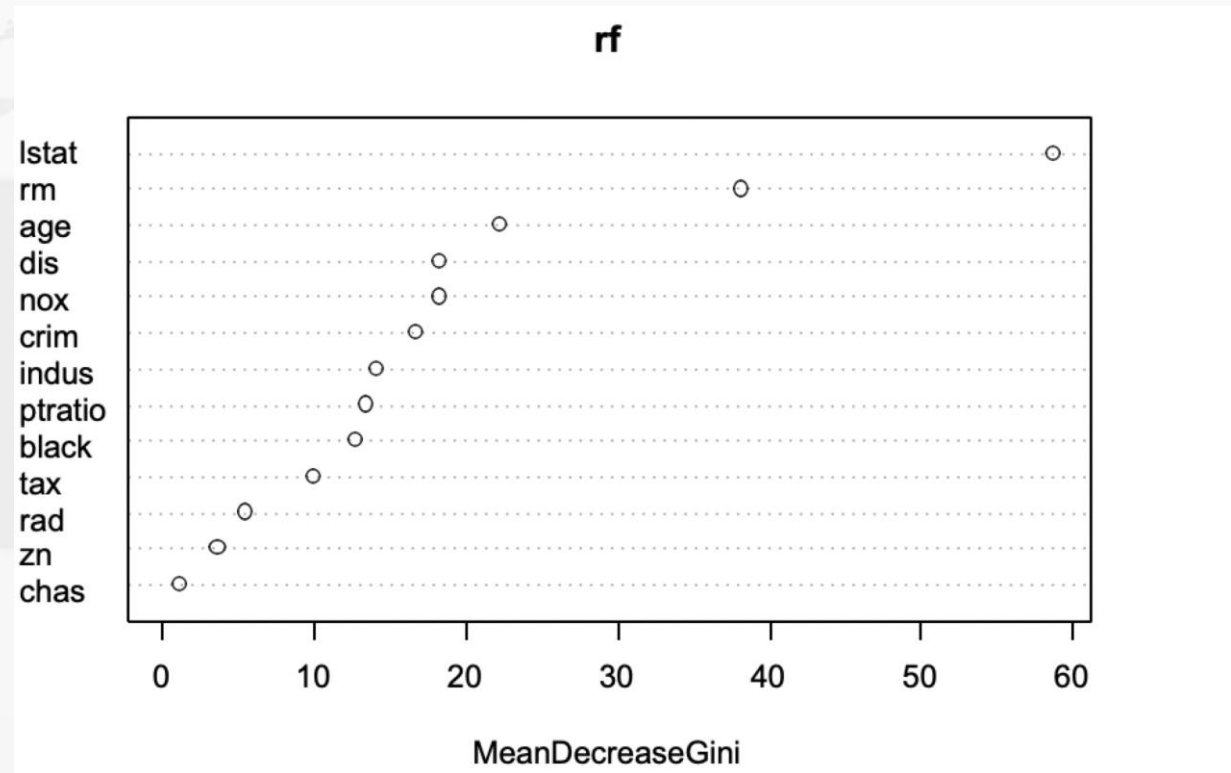
##          high low normal class.error
## high      93   0    19  0.1696429
## low        2  89    23  0.2192982
## normal    18  12    98  0.2343750

importance(rf)

##          MeanDecreaseGini
## crim      16.695479
## zn         3.675720
## indus     14.112419
## chas       1.203133
## nox       18.234082
## rm        38.230112
## age       22.287011
## dis       18.358425
## rad        5.497724
## tax       10.038365
## ptratio   13.465393
## black     12.715362
## lstat     58.790309
```



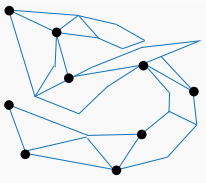
`varImpPlot(rf)`



```
### Predict
pred = predict(rf,newdata=test)

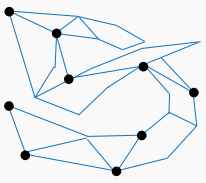
### misclassification rate
table = table(pred,test$level)
1- sum(diag(table))/sum(table) #6.4 error rate

## [1] 0.1776316
```



# Cross Validation

3



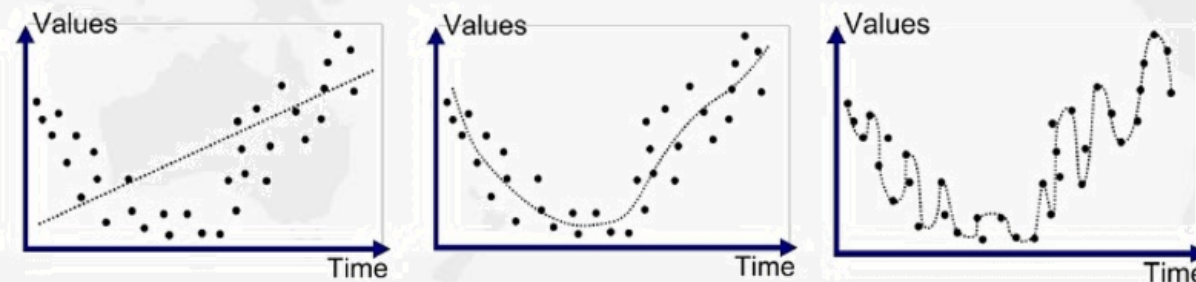
# Model Overfitting vs. Underfitting

## *Overfitting:*

- The model is excessively complex, performs well on the training data but does not perform well on the evaluation data
- Describe random error or noise instead of the underlying relationship
- Show low bias but high variance

## *Underfitting:*

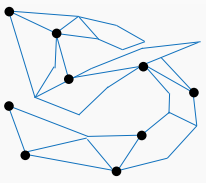
- The model performs poorly on the training data
- Unable to capture the relationship between the input examples and the target values
- Show low variance but high bias



Underfitted

Good Fit/Robust

Overfitted



# Model Overfitting vs. Underfitting

## Solutions

### *Underfitting:*

- Add more features or other relevant variables to boost the performance
- More complex model to capture the complex relationship of the data

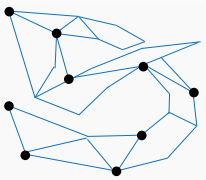
### *Overfitting:*

- Reduce the features
- Simpler model instead of complex one

### *Example:*

An intuitive explanation of overfitting (Quora)



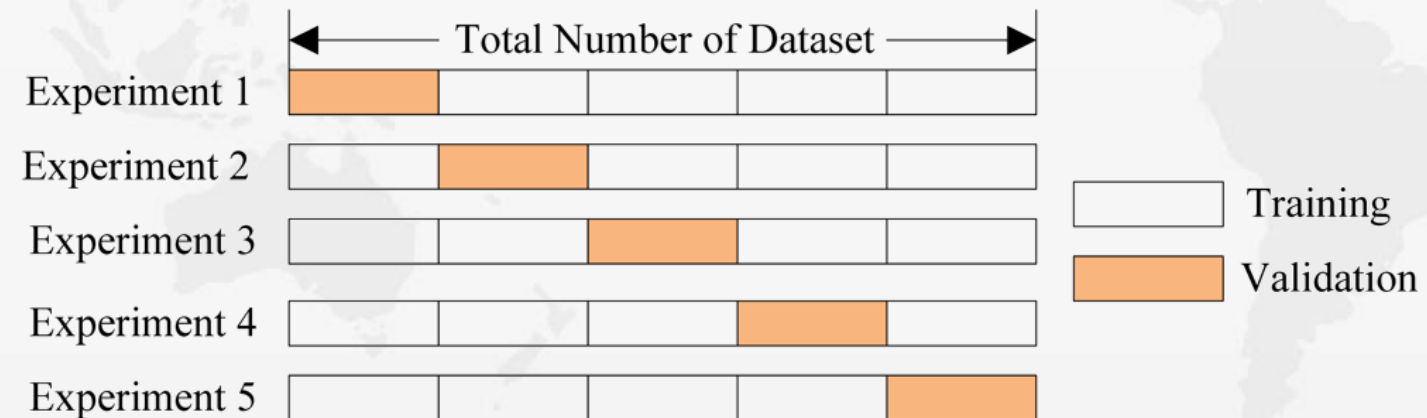


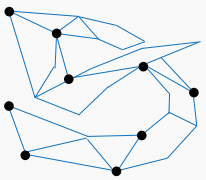
# Cross Validation

A technique for evaluating ML models by training several ML models on subsets of the available input data and evaluating them on the complementary subset of the data.

## *K-fold Cross Validation:*

1. Divide the sample data into  $k$  parts
2. Use  $k-1$  of the parts for training, and 1 for testing
3. Repeat the procedure  $k$  times, rotating the test set
4. Determine an expected performance metric based on the results across the iterations



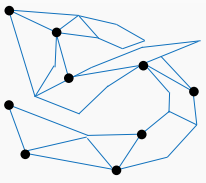


# R Example

```
data(iris)
set.seed(2)
N <- nrow(iris)
train <- sample(1:N, N * 0.8)
test <- -train

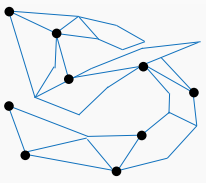
#train
#test
train_data <- iris[train,]
test_data <- iris[test,]
dim(train_data)
## [1] 120 5
dim(test_data)
## [1] 30 5
dim(iris)
## [1] 150 5
```

```
library(caret)
## Loading required package: lattice
## Loading required package: ggplot2
train_set <- createDataPartition(iris$Species, p = 0.8, list = FALSE)
training <- iris[train_set,]
testing <- iris[-train_set,]
dim(training)
## [1] 120 5
dim(testing)
## [1] 30 5
dim(iris)
## [1] 150 5
```



# Clustering

4



# R Example

## Goal

The management team of a large shopping mall would like to understand the types of people who are, or could be, visiting their mall. They are considering designing and positioning the shopping mall services better in order to attract mainly a few profitable market segments, or to differentiate their services across market segments.

## The Data

The Market Research Survey Questions :

*V1*: Shopping is fun (scale 1-7)

*V2*: Shopping is bad for your budget (scale 1-7)

*V3*: I combine shopping with eating out (scale 1-7)

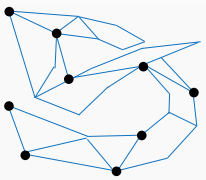
*V4*: I try to get the best buys while shopping (scale 1-7)

*V5*: I don't care about shopping (scale 1-7)

*V6*: You can save lot of money by comparing prices (scale 1-7)

*Income*: the household income of the respondent (in dollars)

*Mall Visits*: how often they visit the mall (scale 1-7)



# R Example

```
library(ggplot2)
```

```
shopping <- read.csv("~/Dropbox/DataLab/Clustering/shopping_mall.csv")
```

```
str(shopping)
```

```
## 'data.frame': 40 obs. of 9 variables:
```

```
## $ ID : int 1 2 3 4 5 6 7 8 9 10 ...
```

```
## $ V1 : int 6 2 7 4 1 6 5 7 2 3 ...
```

```
## $ V2 : int 4 3 2 6 3 4 3 3 4 5 ...
```

```
## $ V3 : int 7 1 6 4 2 6 6 7 3 3 ...
```

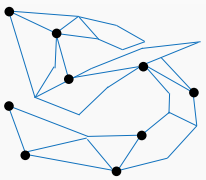
```
## $ V4 : int 3 4 4 5 2 3 3 4 3 6 ...
```

```
## $ V5 : int 2 5 1 3 6 3 3 1 6 4 ...
```

```
## $ V6 : int 3 4 3 6 4 4 4 4 3 6 ...
```

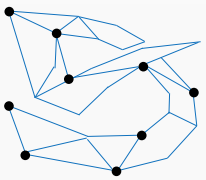
```
## $ Income : Factor w/ 11 levels "25,000","30,000",...: 8 2 10 2 8 6 9 7 10 1 ...
```

```
## $ Mall.Visits: int 3 1 3 7 1 2 3 4 0 6 ...
```



# R Example

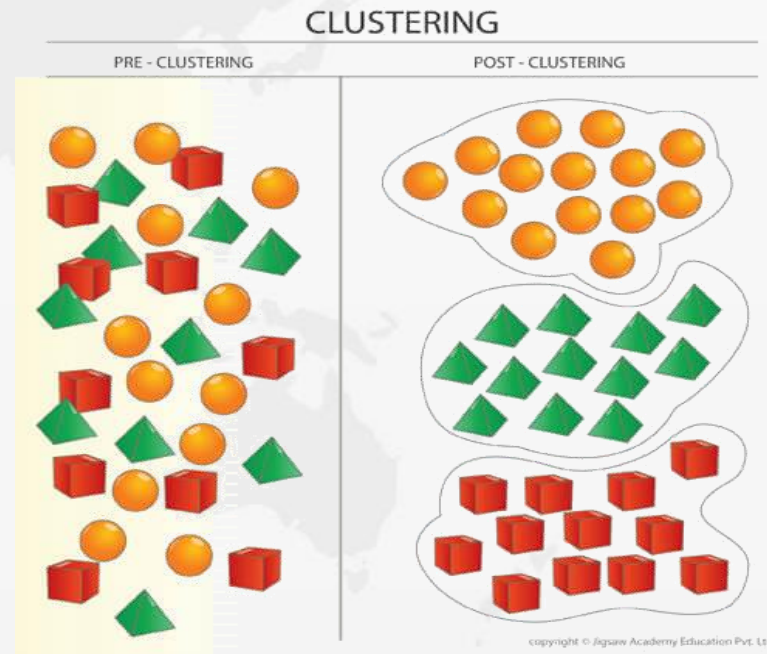
```
### convert Income into number
head(shopping$Income)
## [1] 60,000 30,000 70,000 30,000 60,000 50,000
## 11 Levels: 25,000 30,000 35,000 40,000 45,000 50,000 55,000 ... 80,000
shopping$Income <- as.numeric(gsub(",", "", shopping$Income))
head(shopping$Income)
## [1] 60000 30000 70000 30000 60000 50000
str(shopping)
## 'data.frame': 40 obs. of 9 variables:
## $ ID : int 1 2 3 4 5 6 7 8 9 10 ...
## $ V1 : int 6 2 7 4 1 6 5 7 2 3 ...
## $ V2 : int 4 3 2 6 3 4 3 3 4 5 ...
## $ V3 : int 7 1 6 4 2 6 6 7 3 3 ...
## $ V4 : int 3 4 4 5 2 3 3 4 3 6 ...
## $ V5 : int 2 5 1 3 6 3 3 1 6 4 ...
## $ V6 : int 3 4 3 6 4 4 4 4 3 6 ...
## $ Income : num 60000 30000 70000 30000 60000 50000 65000 55000 70000 25000 ...
## $ Mall.Visits: int 3 1 3 7 1 2 3 4 0 6 ...
?scale
# normalizaing column except the first one
shopping_norm <- scale(shopping[, -1])
```

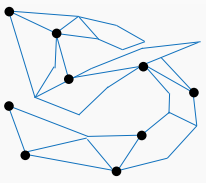


# Clustering

## *Cluster analysis or clustering:*

- Unsupervised learning
- The task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar to each other than to those in other groups (clusters).

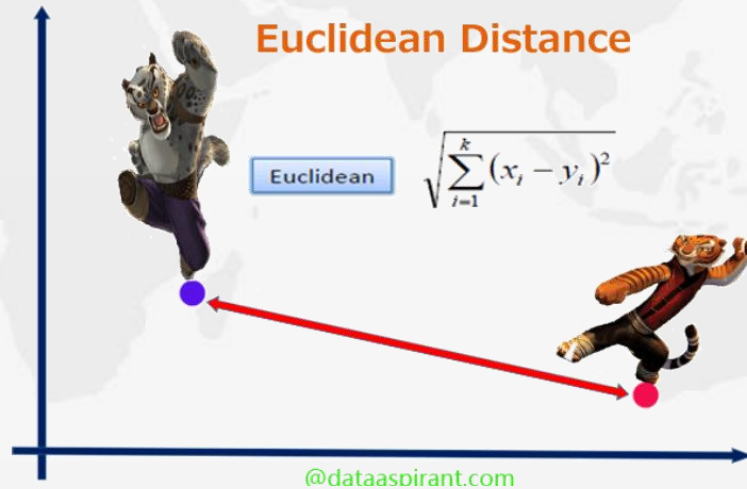




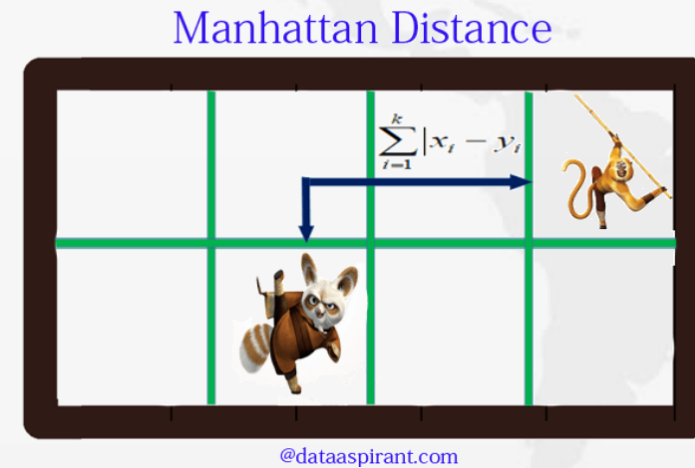
# Similarity Measure

## Two Data Points Similarity Measure:

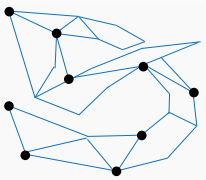
The **Euclidean distance** between two points is the length of the path connecting them. The Pythagorean theorem gives this distance between two points.



**Manhattan distance** is a metric in which the distance between two points is the sum of the absolute differences of their Cartesian coordinates. In a simple way of saying it is the total sum of the difference between the x-coordinates and y-coordinates.





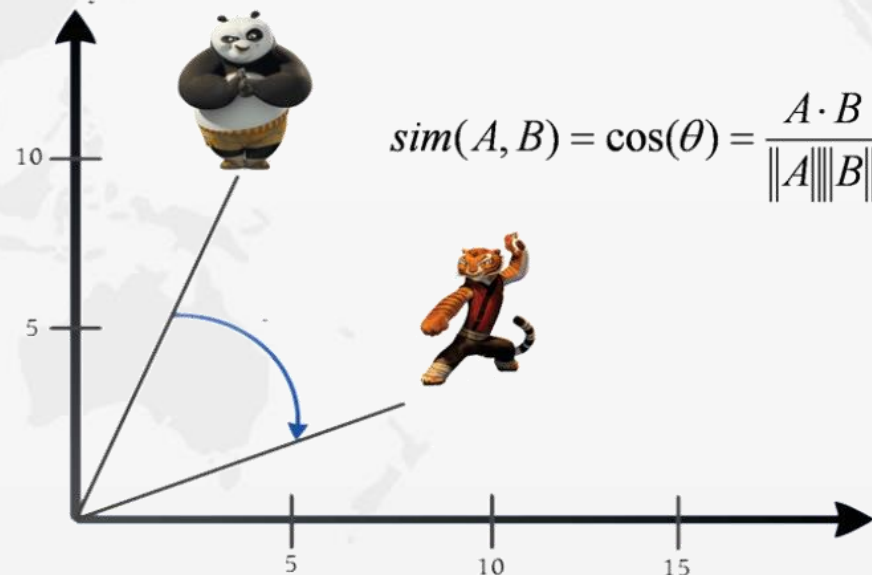


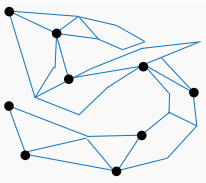
# Similarity Measure

## Cosine similarity

Cosine similarity metric finds the normalized dot product of the two attributes. By determining the cosine similarity, we would effectively try to find the cosine of the angle between the two objects.

### Cosine Similarity

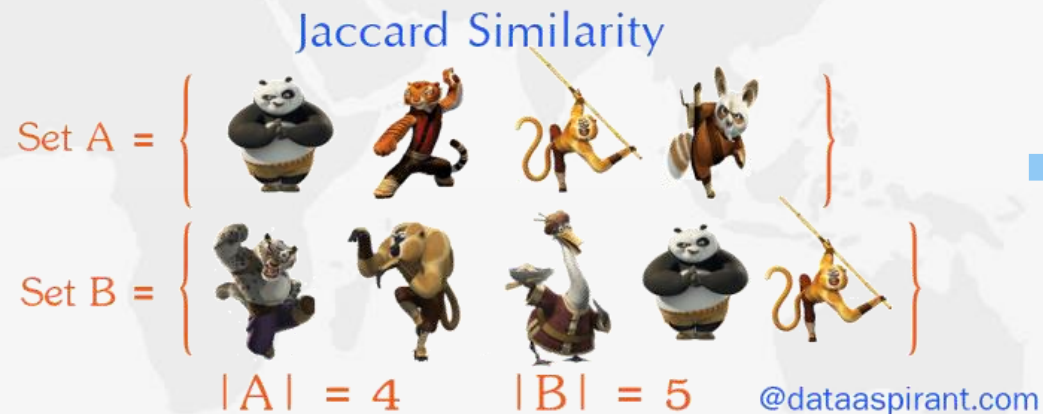


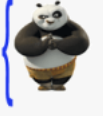



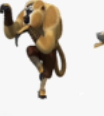
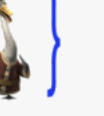
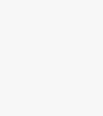




# Similarity Measure

## Jaccard similarity

The Jaccard similarity measures the similarity between finite sample sets and is defined as the cardinality of the intersection of sets divided by the cardinality of the union of the sample sets. Suppose you want to find Jaccard similarity between two sets A and B it is the ratio of intersection divided by union.



Union(A,B) = {        }

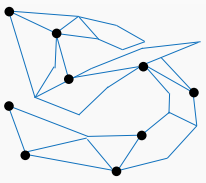
Intersection (A,B) = {   }

$| \text{Union (A,B)} | = 7$        $| \text{Intersection (A,B)} | = 2$

Jaccard Similarity  $J(A,B) = | \text{Intersection (A,B)} | / | \text{Union (A,B)} |$

$= 2 / 7$

$= 0.286$



# Similarity Measure

Two Clusters Similarity Measure There are a few ways to determine how close two clusters are:

## *Complete linkage clustering*

Find the maximum possible distance between points belonging to two different clusters.

## *Single linkage clustering*

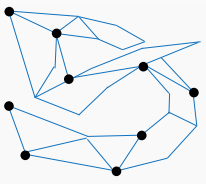
Find the minimum possible distance between points belonging to two different clusters.

## *Mean linkage clustering*

Find all possible pairwise distances for points belonging to two different clusters and then calculate the average.

## *Centroid linkage clustering*

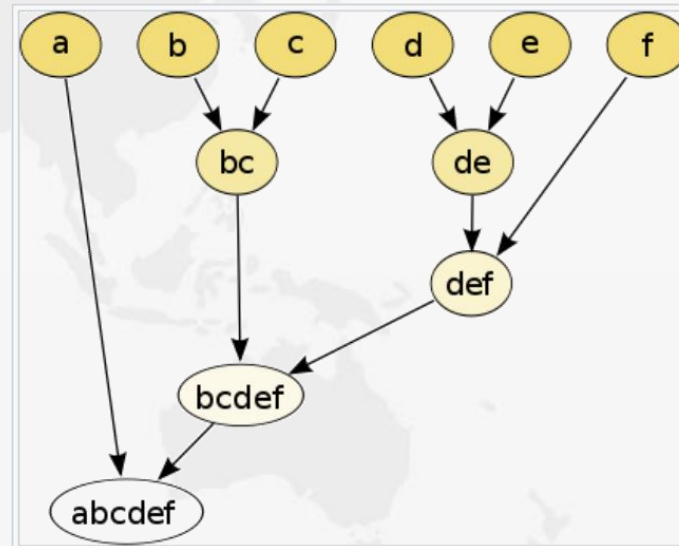
Find the centroid of each cluster and calculate the distance between centroids of two clusters.

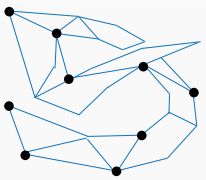


# Hierarchical Cluster Analysis (HCA)

## *Hierarchical clustering:*

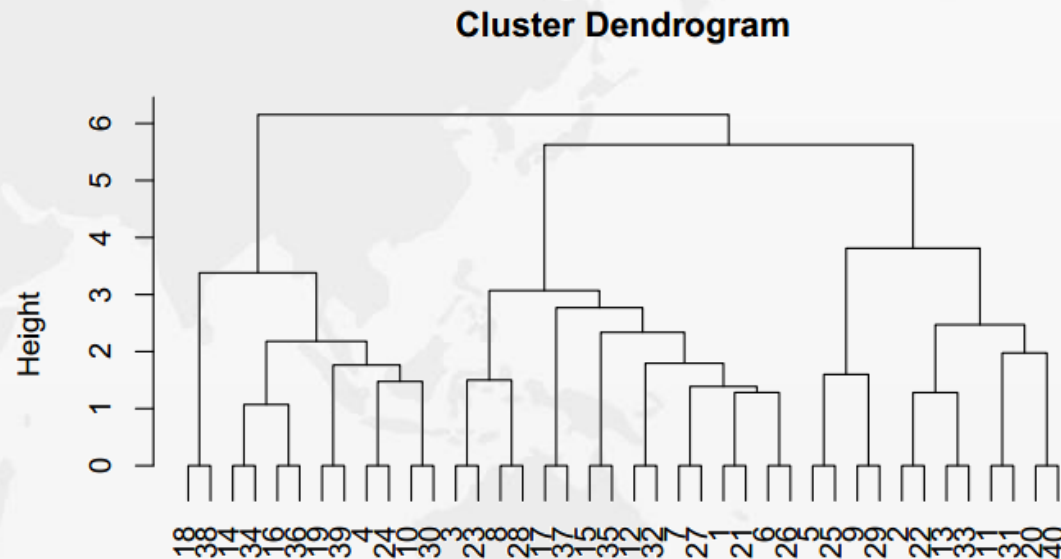
- An algorithm that groups similar objects into groups called clusters.
- The endpoint is a set of clusters, where each cluster is distinct from each other cluster, and the objects within each cluster are broadly similar to each other.



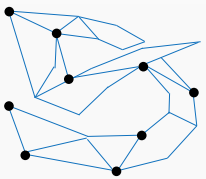


# R Example

```
# by default: method = "euclidean"  
# type "?dist" to see more details  
hc <- hclust(dist(shopping_norm))  
plot(hc)
```

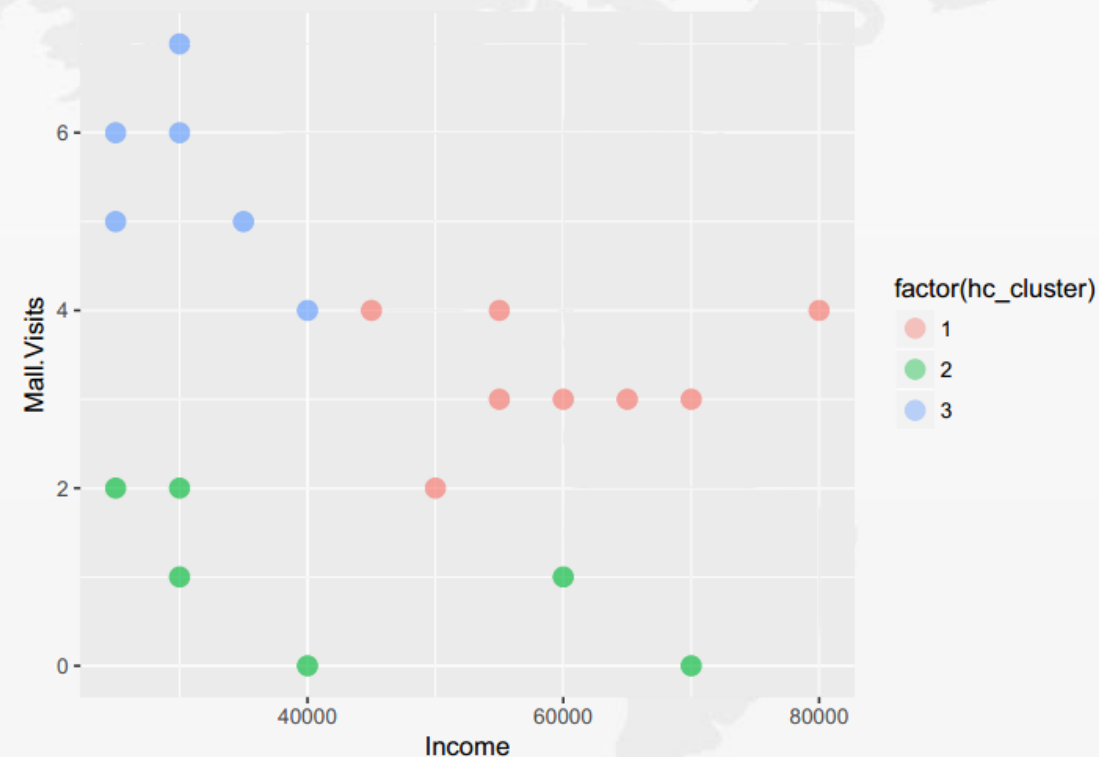


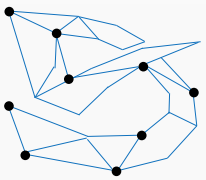
```
dist(shopping_norm)  
hclust (*, "complete")
```



# R Example

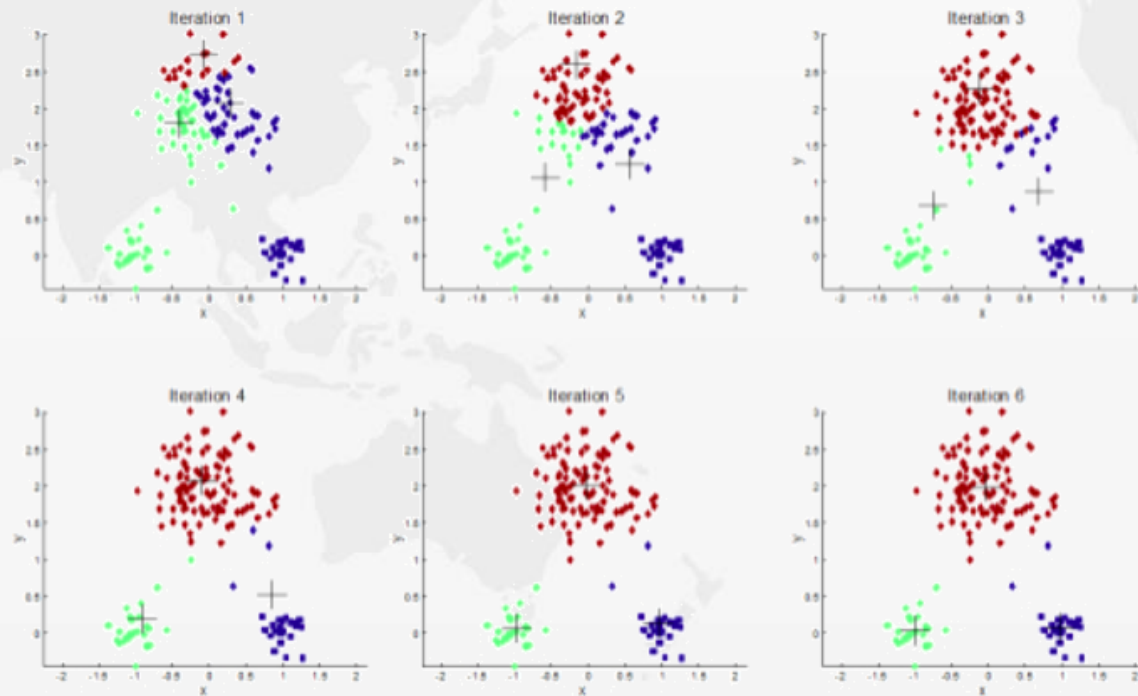
```
hc_cluster <- cutree(hc, 3)
head(hc_cluster)
## [1] 1 2 1 3 2 1
### See the difference of clusters
aggregate(shopping[, -1], by = list(hc_cluster), mean)
## Group.1 V1 V2 V3 V4 V5 V6 Income
## 1 1 5.750000 3.625000 6.000000 3.125 1.875 3.875000 60000.00
## 2 2 1.666667 3.000000 1.833333 3.500 5.500 3.333333 42500.00
## 3 3 3.500000 5.833333 3.333333 6.000 3.500 6.000000 30833.33
## Mall.Visits
## 1 3.25
## 2 1.00
## 3 5.50
ggplot(shopping, aes(Income, Mall.Visits,
                     color = factor(hc_cluster))) +
  geom_point(alpha = 0.4, size = 3.5)
```

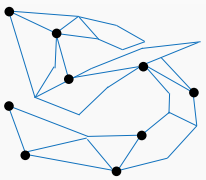




# K-means Clustering Algorithm

- The goal is to find groups in the data, with the number of groups represented by the variable  $K$ .
- The algorithm works iteratively to assign each data point to one of  $K$  groups based on the features that are provided.
- Data points are clustered based on feature similarity.





# R Example

```
# to reproduce the same result, set seed first
set.seed(123)
kmeans_cluster <- kmeans(shopping_norm, 3)
#kmeans_cluster
head(kmeans_cluster$cluster)
## [1] 1 2 1 3 2 1
# to summarize the result
aggregate(shopping[, -1], by = list(kmeans_cluster$cluster), mean)
## Group.1 V1 V2 V3 V4 V5 V6 Income
## 1 1 5.750000 3.625000 6.000000 3.125 1.875 3.875000 60000.00
## 2 2 1.666667 3.000000 1.833333 3.500 5.500 3.333333 42500.00
## 3 3 3.500000 5.833333 3.333333 6.000 3.500 6.000000 30833.33
## Mall.Visits
## 1 3.25
## 2 1.00
## 3 5.50
ggplot(shopping, aes(Income, Mall.Visits,
                     color = factor(kmeans_cluster$cluster))) +
  geom_point(alpha = 0.4, size = 3.5)
```

