Lecture 7

Dependency Modeling and Clustering

Assoc. Prof. Dr. Burkay Genç

2024-04-15

Packages used in these slides

```
library(arules)
library(dplyr)
library(arulesViz)
library(cluster)
library(fpc)
```

Seed used in these slides

```
set.seed(1024)
```

Dependency Modeling

Dependency Modeling

- Finding *interesting* relations between groups of variables
- We live in the big data era
  - Transaction data analysis
    - Associations between products
    - Which products are bought together?
  - Web navigation data analysis
    - Web analytics
    - Search engines
    - Digital libraries
  - Gene expression data analysis
    - DNA micro-arrays
- Association rules analysis
  - a.k.a. association rule mining

Transaction Data

- Typical data set size
  - Typical retailer: ~200 product groups, ~5000 products
  - Amazon: sells over 1.6 million packages per day
  - Google: 400+ billion pages in index
  - Human Genome Project: 3 billion base pairs

Transaction Data

- Typical example
  - milk, flour and eggs are frequently bought together
  - if someone buys milk and flour, it is very likely that they will also buy eggs
  - so, recommend them to buy eggs

Transaction Data

- We have a set of transactions D
  - bread, butter, milk
  - butter, milk
  - bread, potato, tomato
- Each transaction t is a set of items, $t = \{i | i \in I\}$
  - I is the set of all items
  - $\{$bread, butter, milk$\}$
  - Each item is indeed a binary representing existence in the basket

tr. id	bread	butter	milk	potato	tomato
1	1	1	1	0	0
2	0	1	1	0	0
3	1	0	0	1	1

Transaction Data

- Any data can be converted into transaction data via binarization

A	B	C
x	1	T
x	2	T
y	3	F

↓ binarize

A.x	A.y	B.1	B.2	B.3	C.T	C.F
1	0	1	0	0	1	0
1	0	0	1	0	1	0
0	1	0	0	1	0	1

Association Rule

- An association rule is an implication

$$X \rightarrow Y$$

  where
  - $X, Y \subseteq I$
  - $X \neq \emptyset, Y \neq \emptyset$
  - $X \cap Y = \emptyset$
- X is known as the *antecedent*
- Y is known as the *consequent*

$\{bread, butter\} \rightarrow \{milk\}$

Support

- Support of an item set $sup(X)$ is the proportion of the transactions including X.
  - Let n_X be the number of transactions having X, and n_D be the number of all transactions

$sup(X) = n_X / n_D = P(X)$

- Support of an association rule is

$sup(X \rightarrow Y) = n_{X \cup Y} / n_D = P(X \cup Y)$

  - Support can change between 0 and 1

Confidence

- Confidence of an association rule is

$$conf(X \rightarrow Y) = \frac{sup(X \cup Y)}{sup(X)} = P(Y|X)$$

  - Proportion of transactions containing Y in transactions containing X

- Each rule $X \rightarrow Y$ must satisfy the following restrictions:

$$supp(X \cup Y) \geq \sigma$$
$$conf(X \rightarrow Y) \geq \gamma$$

  - this is called the *support-confidence framework*

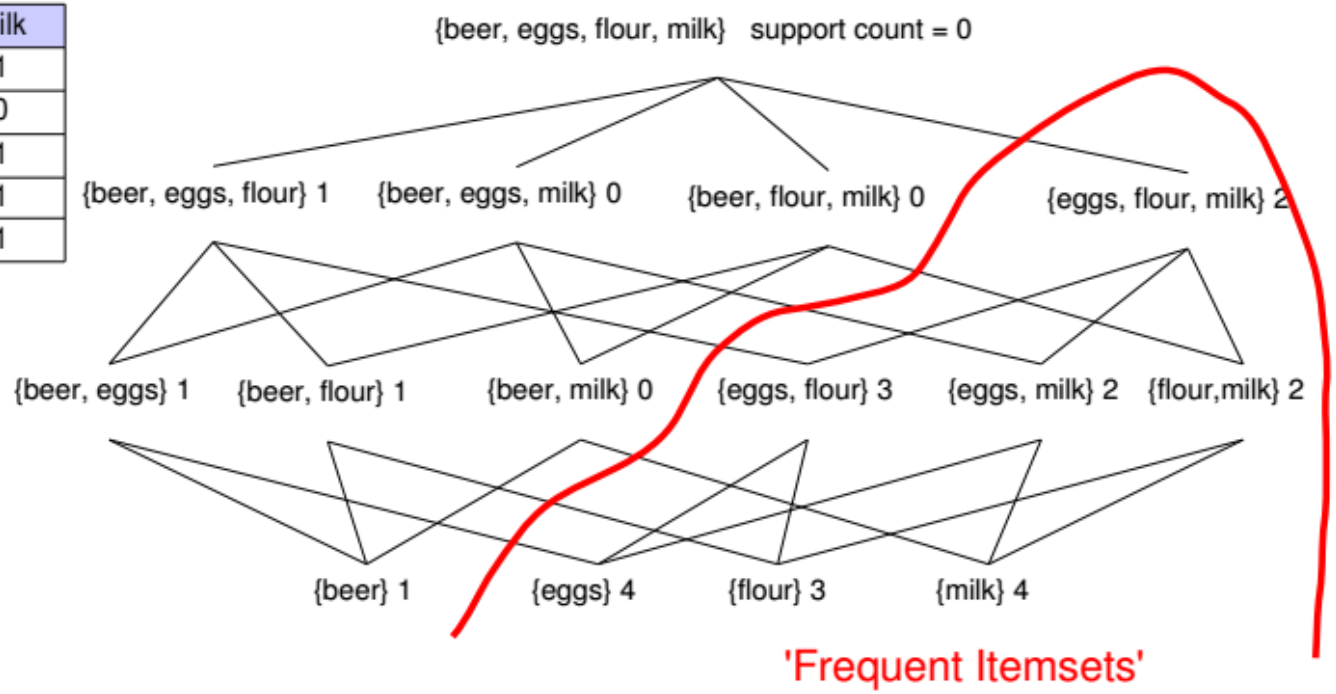
Apriori Algorithm

Approach

- Find all frequent itemsets above a certain minimal support, *minsup*$= \sigma$
- Generate candidate association rules from these itemsets
- Problem:
  - For k different items, we have $2^k - 1$ distinct subsets
  - Brute force search is impossible

Apriori Algorithm

Transaction ID	beer	eggs	flour	milk
1	0	1	1	1
2	1	1	1	0
3	0	1	0	1
4	0	1	1	1
5	0	0	0	1

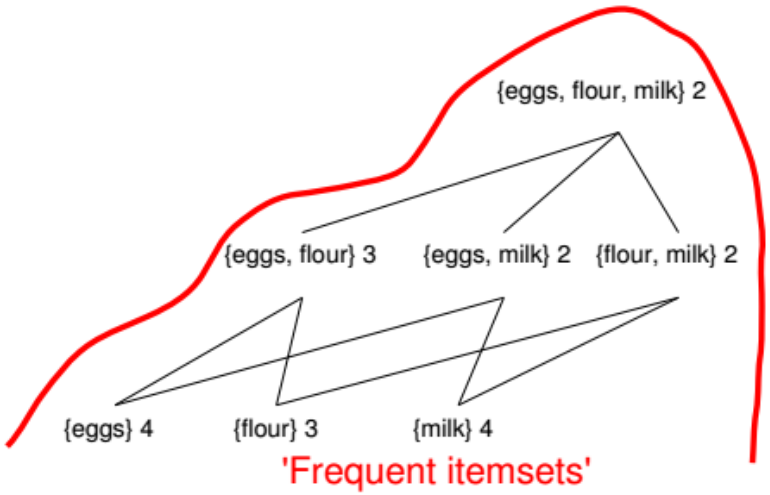


- Agrawal and Srikant, 1994
- Basic idea
  - If an itemset is frequent then its subsets are also frequent

$$sup(X \cup Y) \geq minsup \implies sup(X) \geq minsup \wedge sup(Y) \geq minsup$$

because, $sup(X) \geq sup(X \cup Y)$

Apriori Algorithm



			Confidence
{eggs}	→	{flour}	$3/4 = 0.75$
{flour}	→	{eggs}	$3/3 = 1$
{eggs}	→	{milk}	$2/4 = 0.5$
{milk}	→	{eggs}	$2/4 = 0.5$
{flour}	→	{milk}	$2/3 = 0.67$
{milk}	→	{flour}	$2/4 = 0.5$
{eggs, flour}	→	{milk}	$2/3 = 0.67$
{eggs, milk}	→	{flour}	$2/2 = 1$
{flour, milk}	→	{eggs}	$2/2 = 1$
{eggs}	→	{flour, milk}	$2/4 = 0.5$
{flour}	→	{eggs, milk}	$2/3 = 0.67$
{milk}	→	{eggs, flour}	$2/4 = 0.5$

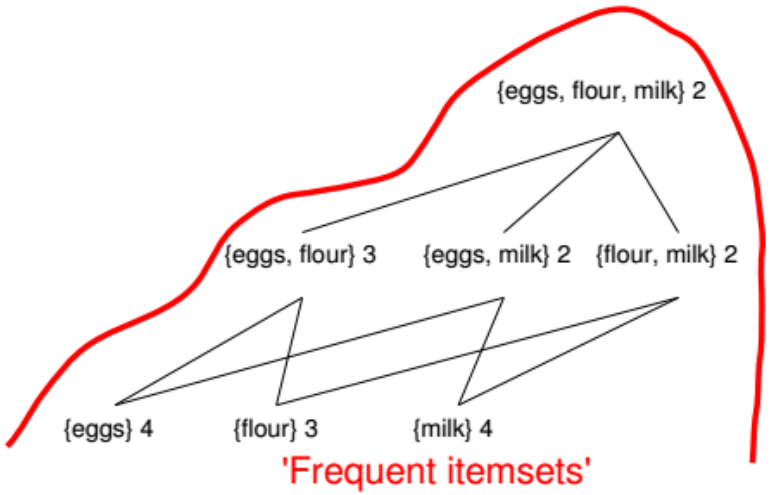
- Second step: Find a partiton $(s, i-s)$ of a frequent itemset i such that

$$conf(s \rightarrow (i-s)) \geq minconf = \gamma$$

$$conf(s \rightarrow (i-s)) = \frac{sup(s \cup (i-s))}{sup(s)} \geq \gamma$$

$$\frac{sup(i)}{sup(s)} \geq \gamma$$

Apriori Algorithm



			Confidence
$\{\text{eggs}\}$	$\rightarrow$	$\{\text{flour}\}$	$3/4 = 0.75$
$\{\text{flour}\}$	$\rightarrow$	$\{\text{eggs}\}$	$3/3 = 1$
$\{\text{eggs}\}$	$\rightarrow$	$\{\text{milk}\}$	$2/4 = 0.5$
$\{\text{milk}\}$	$\rightarrow$	$\{\text{eggs}\}$	$2/4 = 0.5$
$\{\text{flour}\}$	$\rightarrow$	$\{\text{milk}\}$	$2/3 = 0.67$
$\{\text{milk}\}$	$\rightarrow$	$\{\text{flour}\}$	$2/4 = 0.5$
$\{\text{eggs, flour}\}$	$\rightarrow$	$\{\text{milk}\}$	$2/3 = 0.67$
$\{\text{eggs, milk}\}$	$\rightarrow$	$\{\text{flour}\}$	$2/2 = 1$
$\{\text{flour, milk}\}$	$\rightarrow$	$\{\text{eggs}\}$	$2/2 = 1$
$\{\text{eggs}\}$	$\rightarrow$	$\{\text{flour, milk}\}$	$2/4 = 0.5$
$\{\text{flour}\}$	$\rightarrow$	$\{\text{eggs, milk}\}$	$2/3 = 0.67$
$\{\text{milk}\}$	$\rightarrow$	$\{\text{eggs, flour}\}$	$2/4 = 0.5$

At $\gamma = 0.7$ the following set of rules is generated:

	Support	Confidence
{eggs} → {flour}	3/5 = 0.6	3/4 = 0.75
{flour} → {eggs}	3/5 = 0.6	3/3 = 1
{eggs, milk} → {flour}	2/5 = 0.4	2/2 = 1
{flour, milk} → {eggs}	2/5 = 0.4	2/2 = 1

Apriori Algorithm

- There are many other assoication rule algorithm implementations
- Package `arules` provides an interface to some of these implementations

arules

Example

- Before applying apriori algorithm to a dataset, we need to first transform it into transaction format
  - All numeric variables are discretized into categorical variables
    - Requires domain expertise
  - All categorical variables are converted to binary variables
  - Each binary variable now represents an item’s existence

Example

```
data(Boston, package="MASS")
b <- Boston
head(b)
```

```
##      crim zn indus chas   nox    rm  age    dis rad tax ptratio  black lstat
## 1 0.00632 18  2.31    0 0.538 6.575 65.2 4.0900   1 296    15.3 396.90  4.98
## 2 0.02731  0  7.07    0 0.469 6.421 78.9 4.9671   2 242    17.8 396.90  9.14
## 3 0.02729  0  7.07    0 0.469 7.185 61.1 4.9671   2 242    17.8 392.83  4.03
## 4 0.03237  0  2.18    0 0.458 6.998 45.8 6.0622   3 222    18.7 394.63  2.94
## 5 0.06905  0  2.18    0 0.458 7.147 54.2 6.0622   3 222    18.7 396.90  5.33
## 6 0.02985  0  2.18    0 0.458 6.430 58.7 6.0622   3 222    18.7 394.12  5.21
##   medv
## 1 24.0
## 2 21.6
## 3 34.7
## 4 33.4
## 5 36.2
## 6 28.7
```

Example

```
# apriori only works with categorical data
b$chas <- factor(b$chas, labels = c("river", "noriver"))
b$rad <- factor(b$rad)
b$black <- cut(b$black, breaks = 4,
               labels = c(">31.5%", "18.5-31.5%", "8-18.5%", "<8%"))

discr <- function(x) cut(x, breaks = 4,
                         labels = c("low", "medLow", "medHigh", "high"))
b <- select(b, -one_of(c("chas", "rad", "black"))) %>%
  mutate_all(list(~discr(.))) %>%
  bind_cols(select(b, one_of(c("chas", "rad", "black"))))
b <- as(b, "transactions")
b
```

```
## transactions in sparse format with
##  506 transactions (rows) and
##  59 items (columns)
```

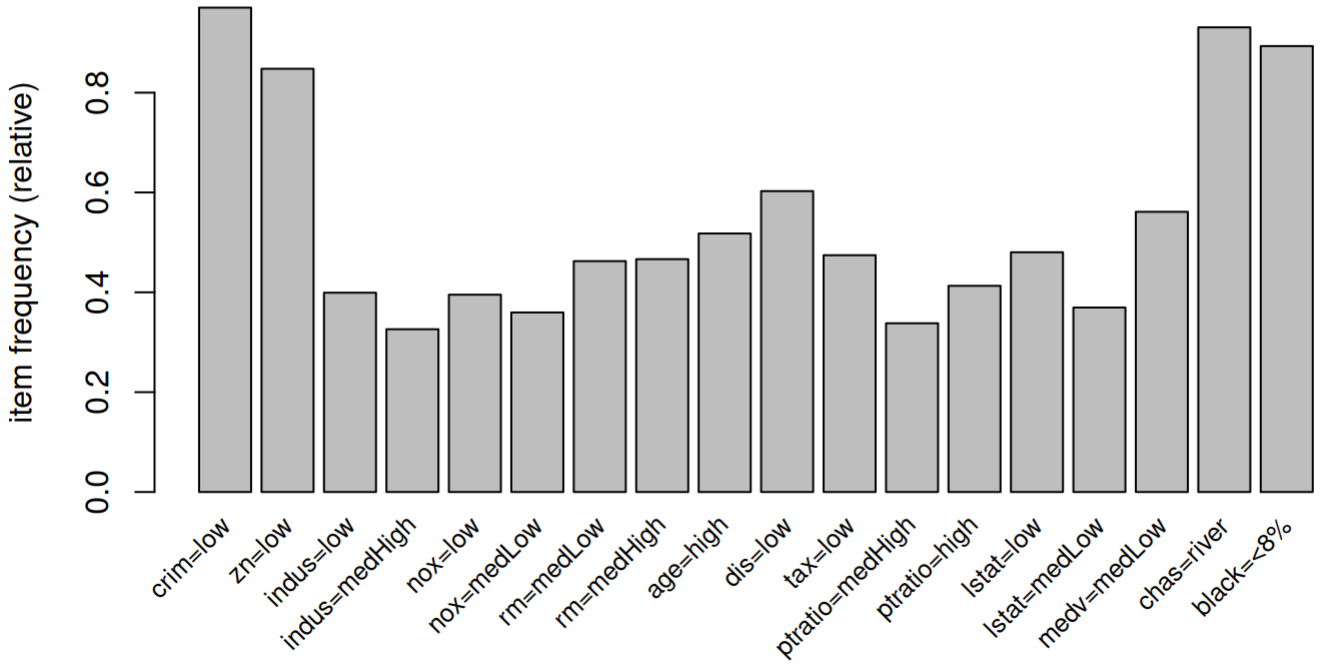
Example

```
summary(b)
```

```
## transactions as itemMatrix in sparse format with
##  506 rows (elements/itemsets/transactions) and
##  59 columns (items) and a density of 0.2372881
##
## most frequent items:
##   crim=low chas=river  black=<8%     zn=low    dis=low    (Other)
##        491        471        452        429        305       4936
##
## element (itemset/transaction) length distribution:
## sizes
##  14
## 506
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      14      14      14      14      14      14
##
## includes extended item information - examples:
##         labels variables  levels
## 1     crim=low      crim     low
## 2  crim=medLow      crim  medLow
## 3 crim=medHigh      crim medHigh
##
## includes extended transaction information - examples:
##   transactionID
## 1             1
## 2             2
## 3             3
```

Example

```
itemFrequencyPlot(b, support = 0.3, cex.names = 0.8)
```



Example

Now we can run the apriori algorithm:

```
ars <- apriori(b, parameter = list(
  support = 0.025, confidence = 0.75, maxlen = 5))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##        0.75    0.1    1 none FALSE            TRUE       5   0.025      1
##  maxlen target  ext
##       5  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 12
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[59 item(s), 506 transaction(s)] done [0.00s].
## sorting and recoding items ... [52 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 5 done [0.01s].
## writing ... [76499 rule(s)] done [0.00s].
## creating S4 object  ... done [0.01s].
```

Example

```
inspect(head(ars))
```

```
##     lhs                   rhs           support    confidence coverage   lift
## [1] {}                 => {zn=low}      0.84782609 0.8478261  1.00000000 1.000000
## [2] {}                 => {black=<8%}   0.89328063 0.8932806  1.00000000 1.000000
## [3] {}                 => {chas=river}  0.93083004 0.9308300  1.00000000 1.000000
## [4] {}                 => {crim=low}    0.97035573 0.9703557  1.00000000 1.000000
## [5] {black=8-18.5%} => {age=high}    0.02766798 0.9333333  0.02964427 1.802545
## [6] {black=8-18.5%} => {dis=low}     0.02766798 0.9333333  0.02964427 1.548415
##     count
## [1] 429
## [2] 452
## [3] 471
## [4] 491
## [5]  14
## [6]  14
```

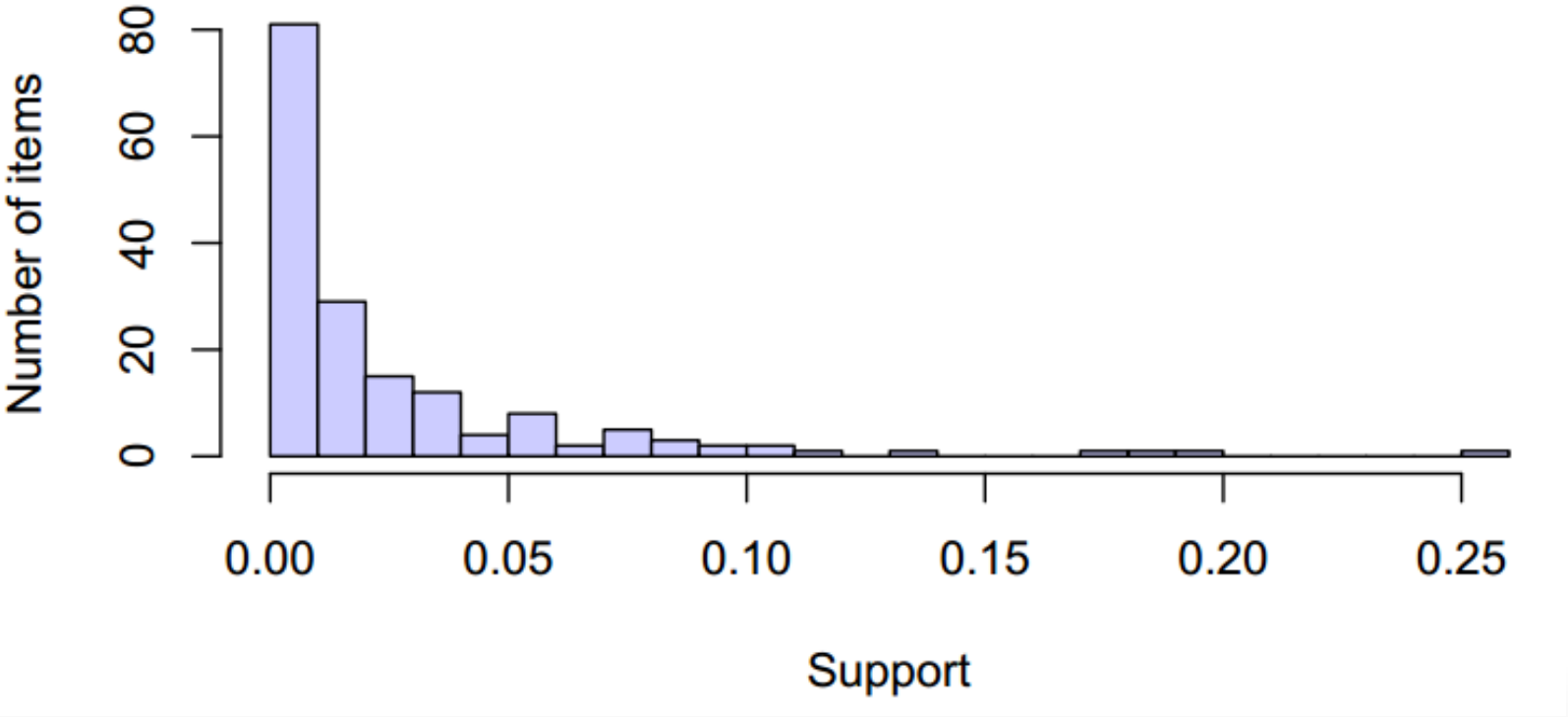
```
inspect(head(subset(ars, rhs %in% "medv=high"), 5, by="confidence"))
```

```
##     lhs                rhs             support confidence   coverage    lift count
## [1] {rm=high,
##      ptratio=low}   => {medv=high} 0.02964427          1 0.02964427 15.8125    15
## [2] {rm=high,
##      ptratio=low,
##      lstat=low}     => {medv=high} 0.02964427          1 0.02964427 15.8125    15
## [3] {rm=high,
##      ptratio=low,
##      black=<8%}     => {medv=high} 0.02964427          1 0.02964427 15.8125    15
## [4] {crim=low,
##      rm=high,
##      ptratio=low}   => {medv=high} 0.02964427          1 0.02964427 15.8125    15
## [5] {rm=high,
##      ptratio=low,
##      lstat=low,
##      black=<8%}     => {medv=high} 0.02964427          1 0.02964427 15.8125    15
```

Weakness of Support

- Support suffers from the *rare item problem* (Liu et al., 1999a)
  - Infrequent items not meeting minimum support are ignored which is problematic if rare items are important.
  - E.g. rarely sold products which account for a large part of revenue or profit.

Typical support distribution (retail point-of-sale data with 169 items):



Weakness of Confidence

- Confidence ignores the frequency of *consequence* (Aggarwal and Yu, 1998; Silverstein et al., 1998).

	X=0	X=1	Σ
Y=0	5	5	10
Y=1	70	**20**	**90**
Σ	75	**25**	100

$$\text{conf}(X \rightarrow Y) = \frac{n_{X \cup Y}}{n_X} = \frac{20}{25} = .8$$

Confidence of the rule is relatively high with $\hat{P}(E_Y|E_X) = .8$. But the unconditional probability $\hat{P}(E_Y) = n_Y/n_D = 90/100 = .9$ is higher!

Lift

$$conf(X \to Y) = \frac{sup(X \cup Y)}{sup(X)}$$

$$lift(X \to Y) = \frac{sup(X \cup Y)}{sup(X)sup(Y)}$$

$$lift(X \to Y) = \frac{conf(X \to Y)}{sup(Y)}$$

$$lift(X \to Y) = \frac{P(Y|X)}{P(Y)}$$

- If Y's frequency is very high in the dataset, its confidence will be usually high
- Lift fixes this
- Lift is high iff conditioning on X improves Y's probability

Lift

- In marketing values of lift are interpreted as:
  - $lift(X \rightarrow Y) = 1$ . . . X and Y are independent
  - $lift(X \rightarrow Y) > 1$ . . . complementary effects between X and Y
  - $lift(X \rightarrow Y) < 1$ . . . substitution effects between X and Y Example

Example

```
inspect(head(subset(ars,
                    subset = lhs %in% "nox=high" | rhs %in% "nox=high"),
             5, by="confidence"))
```

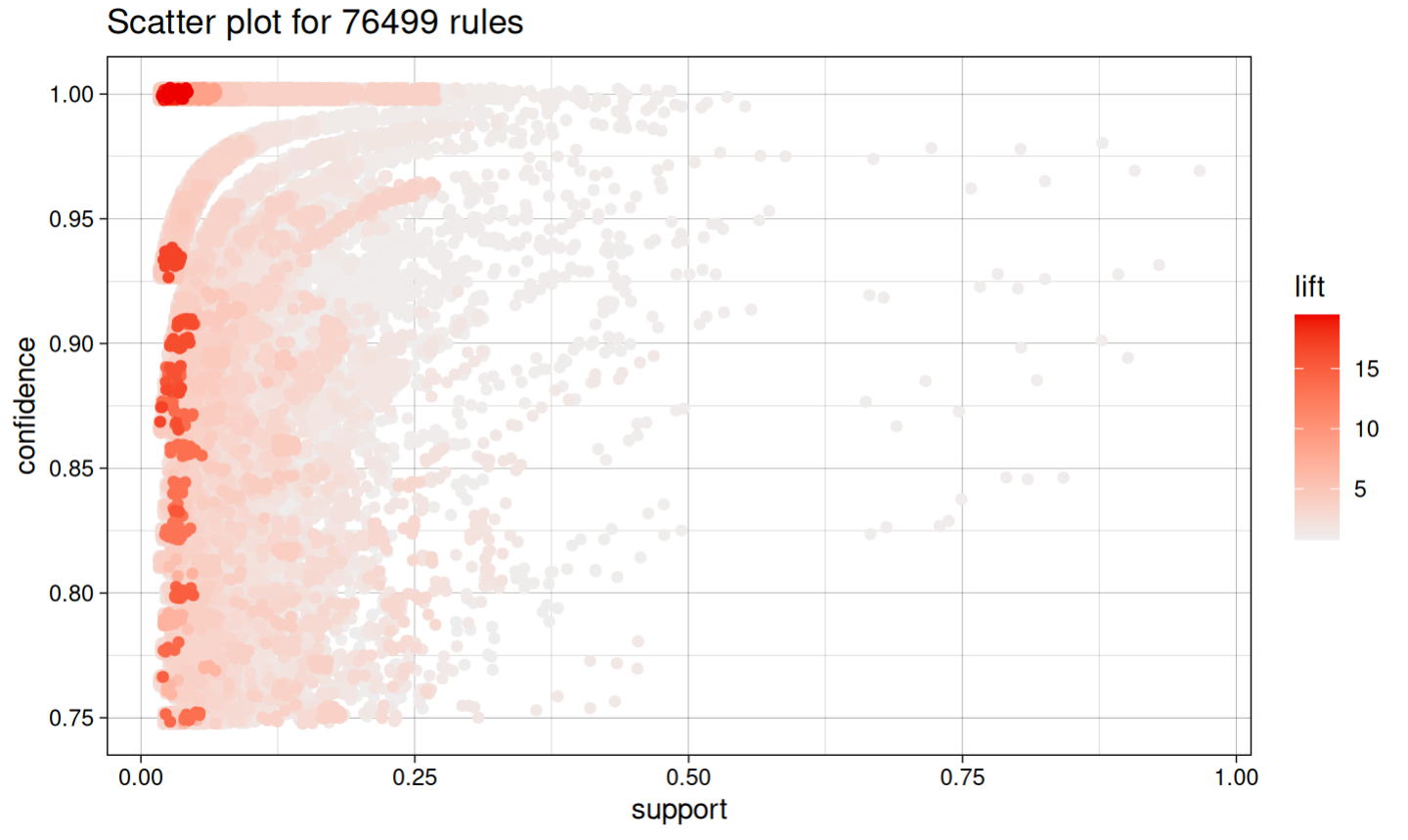
```
##     lhs           rhs              support    confidence coverage   lift
## [1] {nox=high} => {indus=medHigh} 0.04743083 1          0.04743083 3.066667
## [2] {nox=high} => {age=high}      0.04743083 1          0.04743083 1.931298
## [3] {nox=high} => {dis=low}       0.04743083 1          0.04743083 1.659016
## [4] {nox=high} => {zn=low}        0.04743083 1          0.04743083 1.179487
## [5] {nox=high} => {crim=low}      0.04743083 1          0.04743083 1.030550
##     count
## [1] 24
## [2] 24
## [3] 24
## [4] 24
## [5] 24
```

Visualization

- use `arulesViz` package

```
plot(ars)
```

```
## To reduce overplotting, jitter is added! Use jitter = 0 to prevent jitter.
```



Other functions

- quality() returns a data frame of measures by itemsets
- size() returns the length of lhs or rhs

```
goodRules <- ars[quality(ars)$confidence > 0.8 & quality(ars)$lift > 5]
inspect(head(goodRules, 3, by="confidence"))
```

```
##     lhs                          rhs           support    confidence coverage
## [1] {nox=high, rad=5}         => {ptratio=low} 0.03162055 1          0.03162055
## [2] {nox=high, tax=medLow}    => {ptratio=low} 0.03162055 1          0.03162055
## [3] {rm=high, ptratio=low}    => {medv=high}   0.02964427 1          0.02964427
##     lift      count
## [1]  8.724138 16
## [2]  8.724138 16
## [3] 15.812500 15
```

```
shortRules <- subset(ars, subset = size(lhs) + size(rhs) <= 3)
inspect(head(shortRules, 3, by="confidence"))
```

```
##     lhs                  rhs          support    confidence coverage   lift
## [1] {black=8-18.5%}   => {zn=low}     0.02964427 1          0.02964427 1.179487
## [2] {black=8-18.5%}   => {chas=river} 0.02964427 1          0.02964427 1.074310
## [3] {zn=medHigh}      => {nox=low}    0.03162055 1          0.03162055 2.530000
##     count
## [1] 15
## [2] 15
## [3] 16
```

Other functions

- `lhs %in% X`: lhs contains an item from X
- `lhs %ain% X`: lhs contains all items in X
- `lhs %oin% X`: lhs contains only items from X
- `lhs %pin% X`: lhs contains items partially matching from X

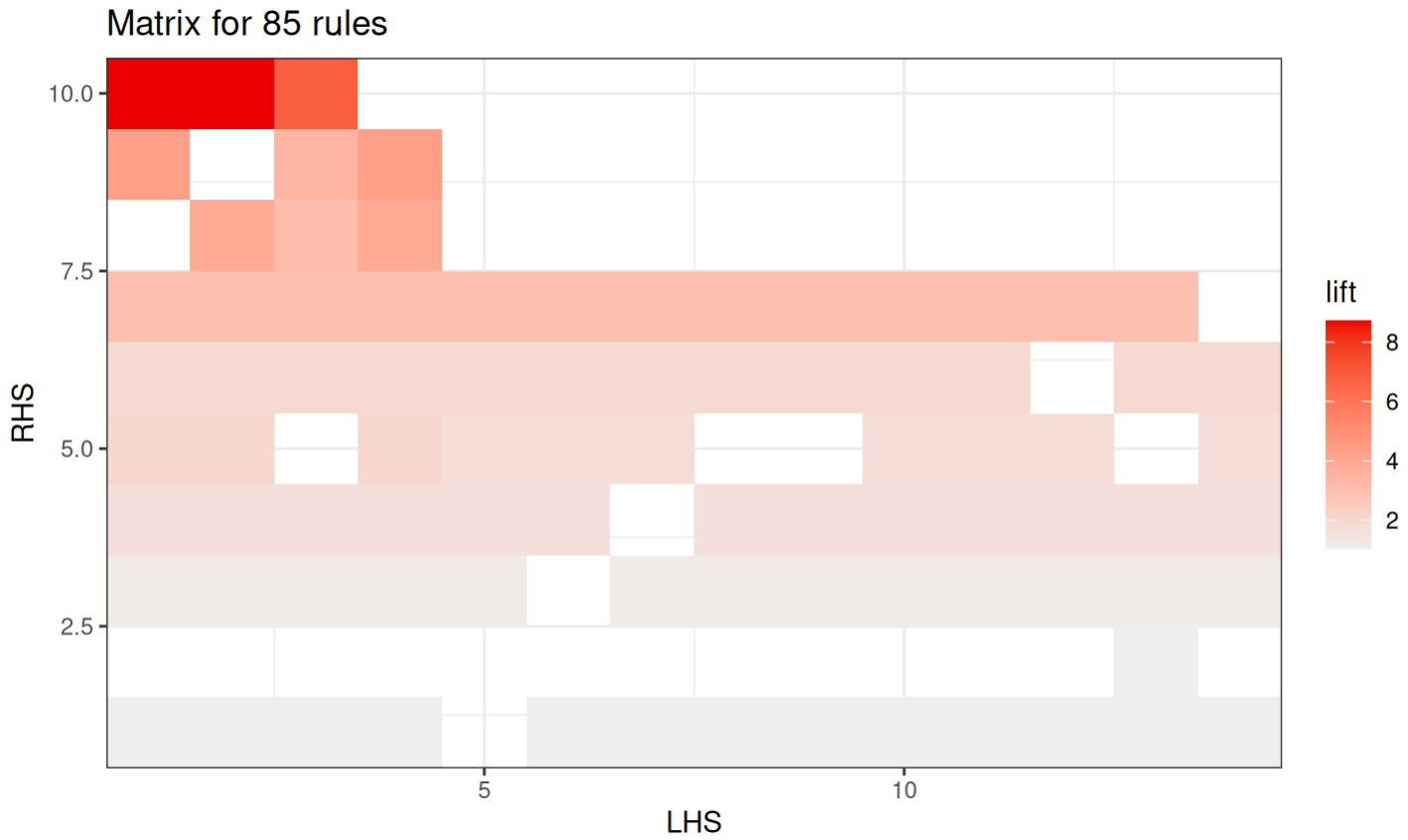
Other functions

```
somerules <- subset(goodRules, subset =
                      lhs %pin% "crim" &
                      rhs %oin% c("medv=high", "medv=medHigh"))
inspect(head(somerules, 3, by="confidence"))
```

```
##     lhs                rhs            support confidence   coverage    lift count
## [1] {crim=low,
##      rm=high,
##      ptratio=low}  => {medv=high} 0.02964427          1 0.02964427 15.8125    15
## [2] {crim=low,
##      rm=high,
##      ptratio=low,
##      lstat=low}    => {medv=high} 0.02964427          1 0.02964427 15.8125    15
## [3] {crim=low,
##      rm=high,
##      ptratio=low,
##      black=<8%}    => {medv=high} 0.02964427          1 0.02964427 15.8125    15
```

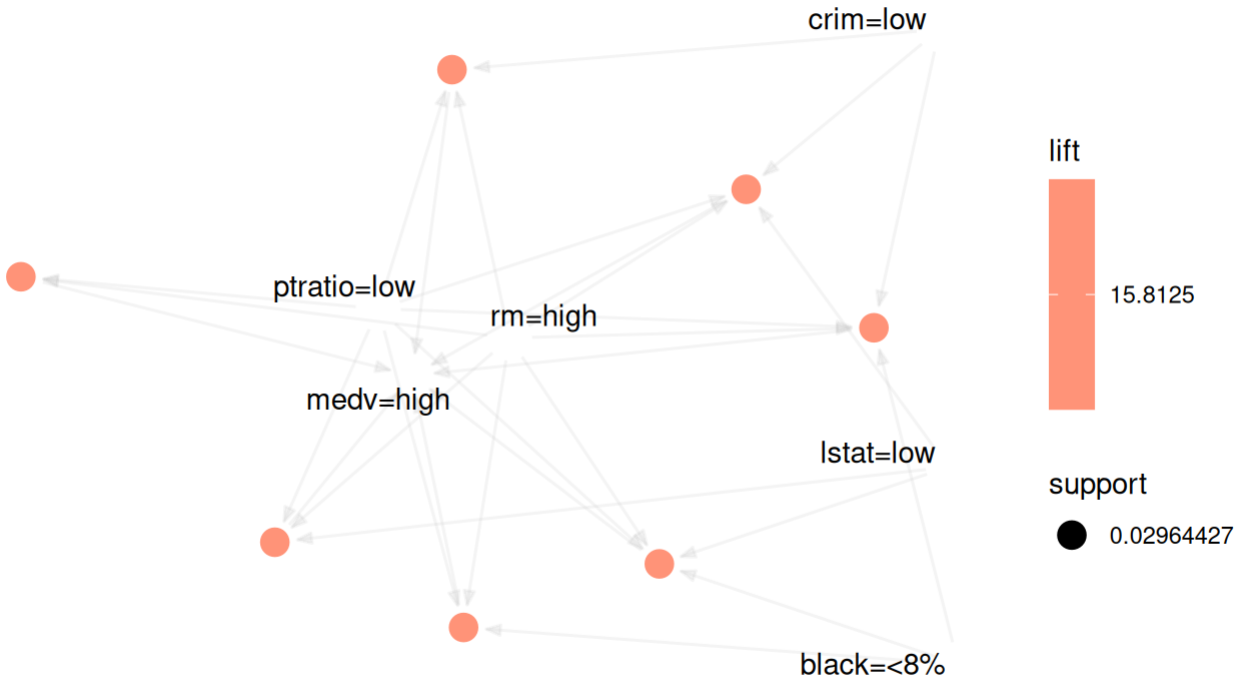
Matrix Visualization

```
somerules <- subset(ars, subset = (lhs %in% "nox=high" &
                    size(lhs) < 3 & lift >= 1))
plot(somerules, method="matrix", measure="lift")
```



Graph Visualization

```
somerules <- subset(ars, subset = rhs %in% "medv=high" & lift > 15)
plot(somerules, method="graph")
```



Detailed demo

Demo

Clustering

Clustering

- Partitioning of a dataset into groups
  - Union of groups is the whole dataset
  - Each observation belongs to exactly one group
  - Observations within a group are *similar*
  - Observations from different groups are *different*
- How to measure similarity and difference?
  - similar = close
  - different = distant

Distance measures

- Euclidean distance

$$d(x, y) = \sqrt{\sum_{i=1}^{d} (x_i - y_i)^2}$$

where x_i is the value of variable i on observation x.

- Euclidean distance is a special form of Minkowski distance

$$d(x, y) = \left(\sum_{i=1}^{d} |x_i - y_i|^p \right)^{1/p}$$

where x_i is the value of variable i on observation x.

- p = 1 -> Manhattan distance
- p = 2 -> Euclidean distance

Distance measures

```
randDat <- matrix(rnorm(50), nrow = 5)
dist(randDat)     # Euclidean distance by default
```

```
##          1        2        3        4
## 2 5.814467
## 3 4.721438 3.320670
## 4 3.673762 3.837265 3.805705
## 5 4.804807 4.952012 3.994814 5.413585
```

```
dist(randDat, method="manhattan")
```

```
##           1         2         3         4
## 2 15.538209
## 3 12.087300  7.881318
## 4 10.006198 10.580025 10.408077
## 5 12.736963 11.085177 10.111811 13.300550
```

```
dist(randDat, method="minkowski", p = 4)
```

```
##          1        2        3        4
## 2 4.092154
## 3 3.266844 2.597013
## 4 2.469316 2.474234 2.594040
## 5 3.240055 3.920720 3.050767 4.191781
```

Distance measures

- What to do with categorical variables?
  - For ordinals we may still compute minkowksi distances
  - For nominals
    - 0 if values are different
    - 1 if values are the same
  - Hybrid data

$$\delta_i(v_1, v_2) = \begin{cases} 1 & i \text{ is nominal and } v_1 \neq v_2 \\ 0 & i \text{ is nominal and } v_1 = v_2 \\ \frac{|v_1 - v_2|}{range(i)} & i \text{ is numeric} \end{cases}$$

  - This makes sure all distances lie in $[0, 1]$
    - Implemented in `daisy()` function of the `cluster` package.

Methods

- Partitioning methods
  - Partition into k clusters
- Hierarchical methods
  - Provide alternative clusterings
  - Divisive vs. agglomerative
- Density-based methods
  - Find high density regions of the feature space
- Grid-based methods
  - Compute clusters within grid cells
  - Highly efficient
  - Combined with other methods

Measures

- How to measure the quality of a clustering?
  - Assume $h(c)$ -> score of cluster c

$$H(C,k) = \sum_{c \in C} \frac{h(c)}{|C|}$$

$$H(C,k) = \min_{c \in C} h(c)$$

$$H(C,k) = \max_{c \in C} h(c)$$

Measures

- How to measure $h(c)$?
  - The sum of squares to the center of each cluster

$$h(c) = \sum_{x \in c} \sum_{i=1}^{d} (v_{x,i} - \hat{v}_i^c)^2$$

  - The L_1 measure

$$h(c) = \sum_{x \in c} \sum_{i=1}^{d} |v_{x,i} - \tilde{v}_i^c|$$

k-means

- Idea
  - Randomly assign observations to k clusters
  - Repeat
    - Compute cluster centers
    - Assign each observation to the closest center
  - Until clusters are stable
- Tries to maximize the inter-cluster distance
  - Does not necessarily guarantee optimality

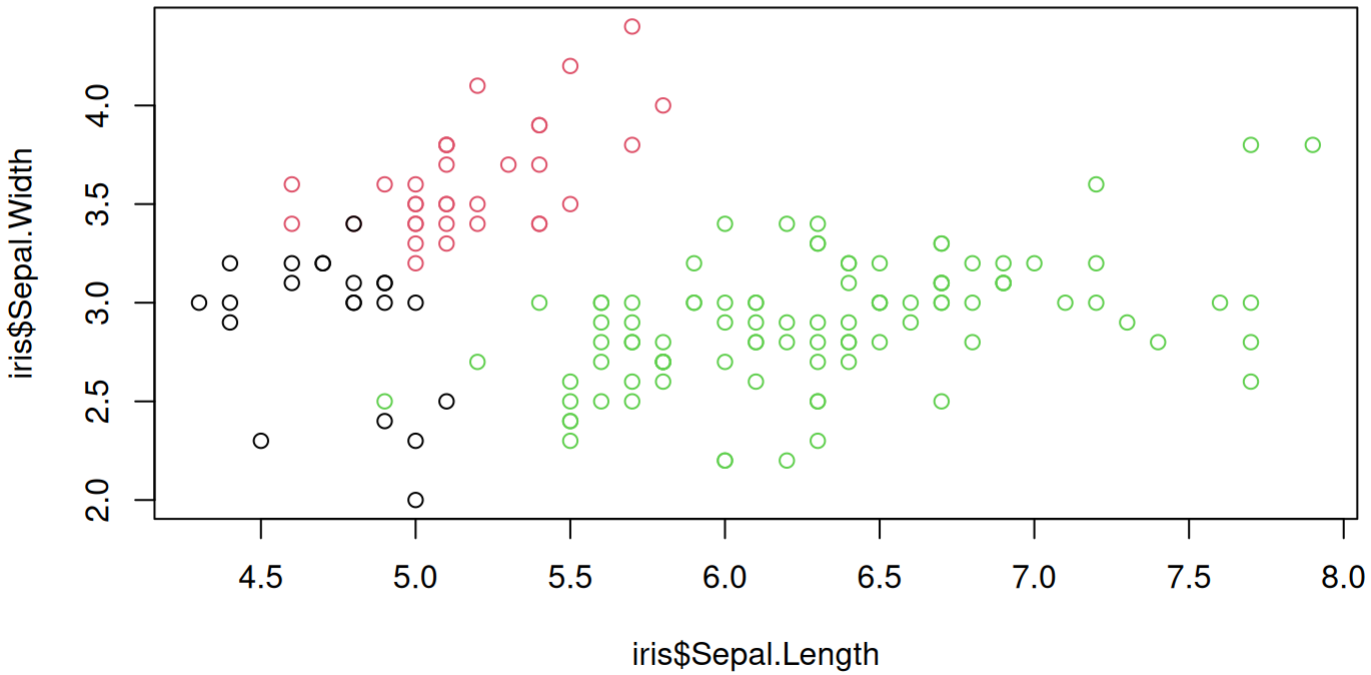
Example

```
data(iris)
ir3 <- kmeans(iris[, -5], centers = 3, iter.max = 200)
ir3
```

```
## K-means clustering with 3 clusters of sizes 21, 33, 96
##
## Cluster means:
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1     4.738095    2.904762     1.790476   0.3523810
## 2     5.175758    3.624242     1.472727   0.2727273
## 3     6.314583    2.895833     4.973958   1.7031250
##
## Clustering vector:
##   [1] 2 1 1 1 2 2 2 2 1 1 2 2 1 1 2 2 2 2 2 2 2 2 2 1 1 2 2 2 1 1 2 2 2 1 2 2
##  [38] 2 1 2 2 1 1 2 2 1 2 1 2 2 3 3 3 3 3 3 3 1 3 3 1 3 3 3 3 3 3 3 3 3 3 3 3
##  [75] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 3 3 3 3 1 3 3 3 3 3 3 3 3 3 3
## [112] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
## [149] 3 3
##
## Within cluster sum of squares by cluster:
## [1]  17.669524   6.432121 118.651875
##  (between_SS / total_SS =  79.0 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

Visualization

```
plot(iris$Sepal.Length, iris$Sepal.Width, col = ir3$cluster)
```



Validation

- External information

```
table(ir3$cluster, iris$Species)
```

```
##
##     setosa versicolor virginica
##   1     17          4         0
##   2     33          0         0
##   3      0         46        50
```

- Internal information - the silhouette method

$$s_i = \frac{b_i - a_i}{\max(a_i, b_i)}$$

where

- a_i is the average distance of observation i to observations **within** its cluster
- b_i is the average distance of observation i to observations **in other** clusters

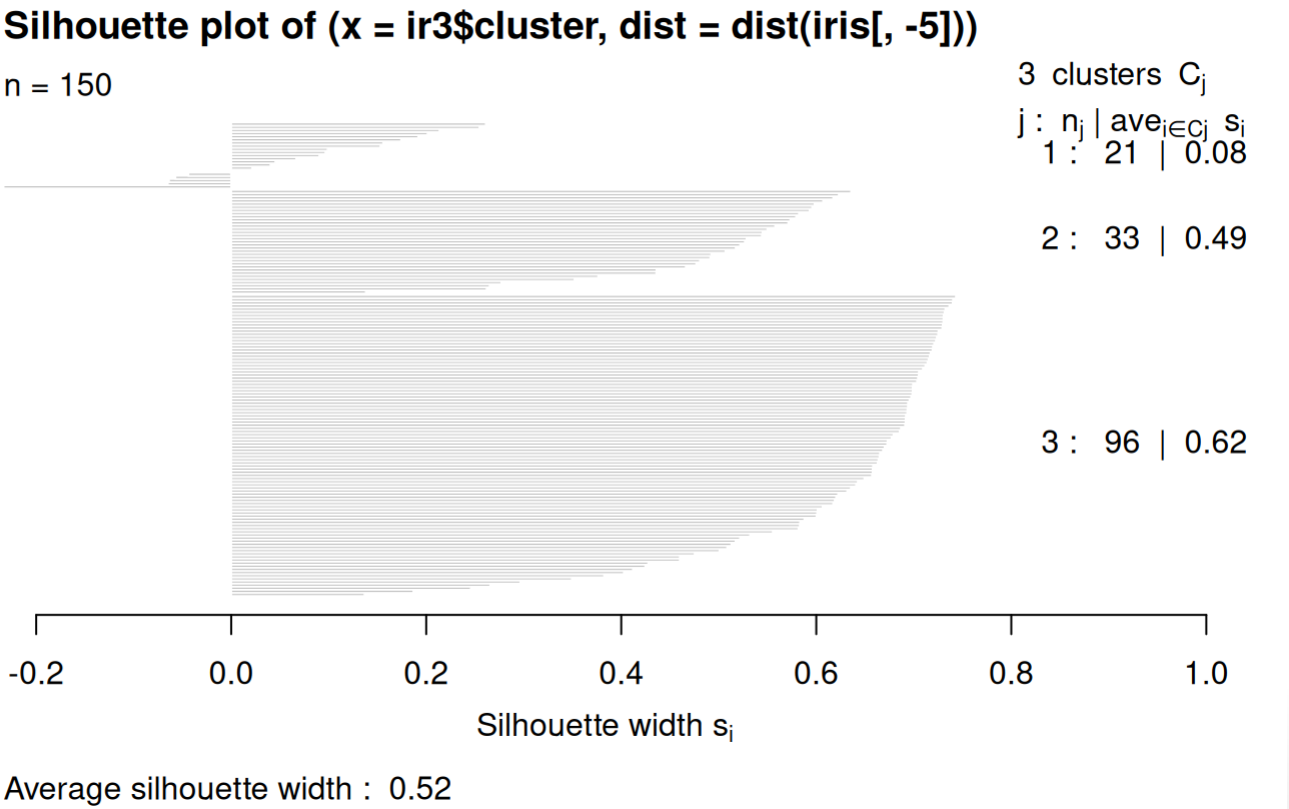
Validation

- `silhouette` is implemented in `cluster`

```
s <- silhouette(ir3$cluster, dist(iris[,-5]))
s[1:5, 3]
```

```
## [1]  0.57933436  0.02148063 -0.05730046  0.15589126  0.57128261
```

```
plot(s)
```



Validation

- For further discussion on k-means
  - read Aggarwal and Reddy (2014), Chapter 23
  - check package `clv`
  - also check k-medoids
    - function `pam()` in package `cluster`

Hierarchical Clustering

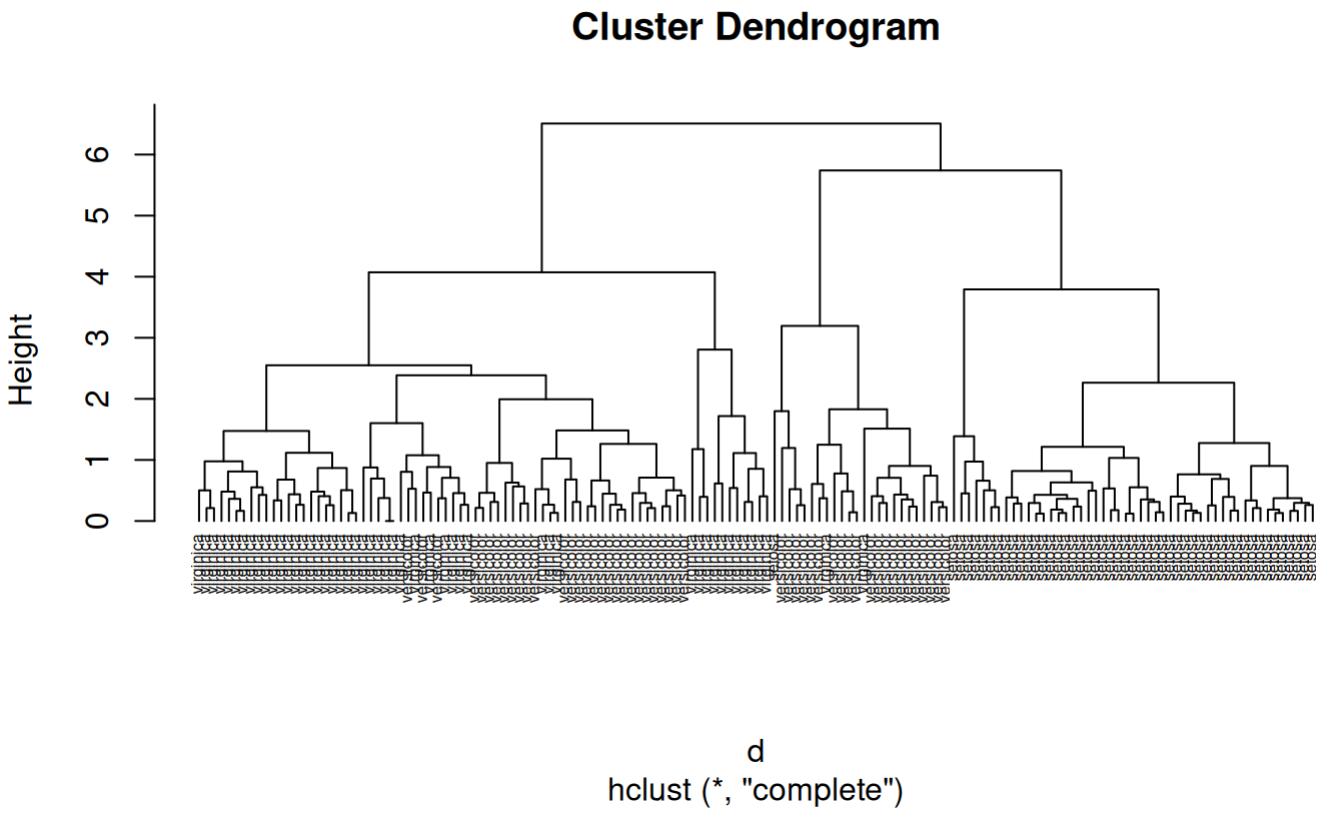
- Provides a hierarchy of alternative clusterings
  - From 1 cluster to n clusters

Agglomerative Hierarchical Clustering

- Start with unit clusters
- Merge two closest clusters into a single cluster
  - Single linkage: smallest distance
  - Complete linkage: largest distance
  - Average linkage: average distance

Hierarchical Clustering

```
d <- dist(scale(iris[,-5]))
h <- hclust(d)
plot(h, hang = -0.1, labels = iris[["Species"]], cex = 0.5)
```



Hierarchical Clustering

```
clus3 <- cutree(h, 3)
clus3
```

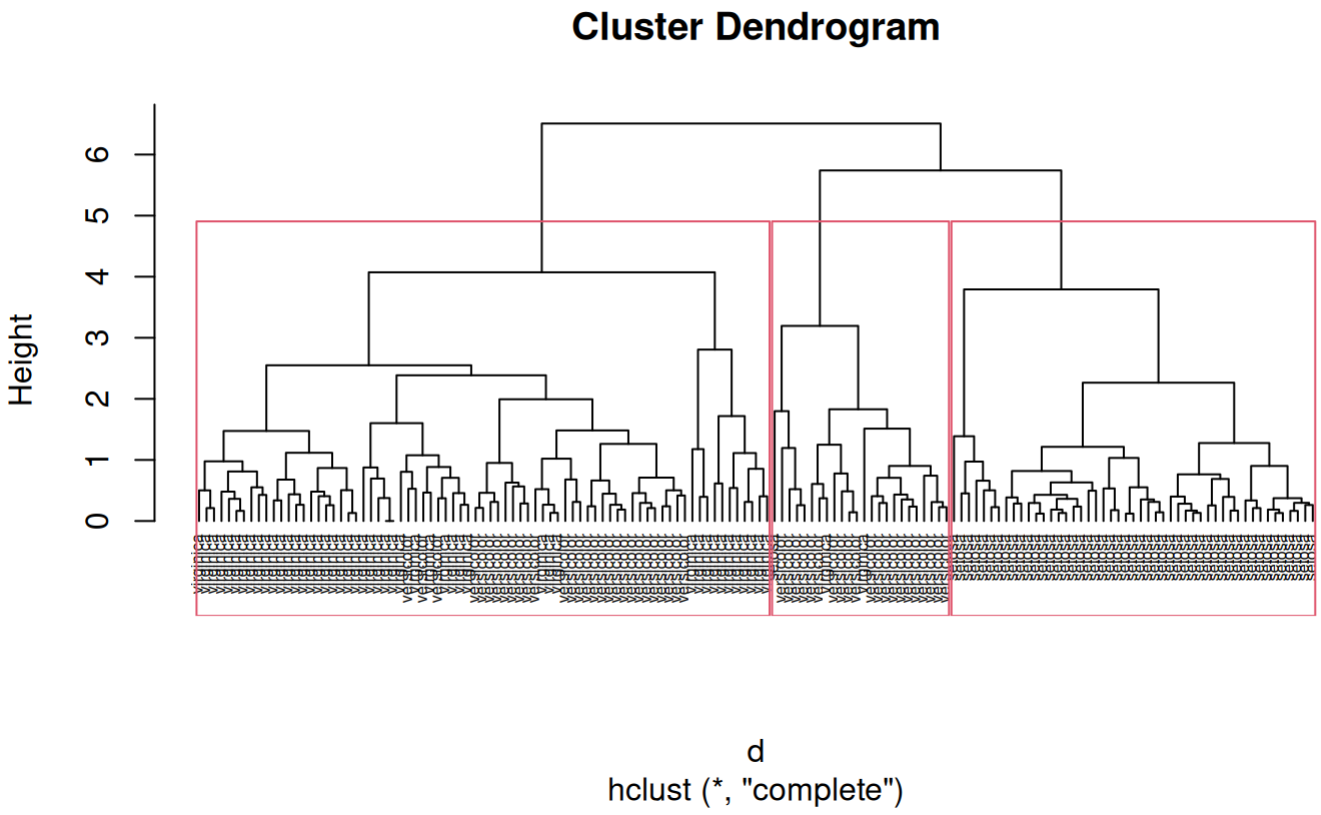
```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 2 1 1 1 1 1 1 1 3 3 3 2 3 2 3 2 3 2 2 3 2 3 3 3 3 2 2 2 3 3 3 3
##  [75] 3 3 3 3 3 2 2 2 2 3 3 3 3 2 3 2 2 3 2 2 2 3 3 3 2 2 3 3 3 3 3 3 2 3 3 3
## [112] 3 3 3 3 3 3 3 3 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
## [149] 3 3
```

```
cm <- table(clus3, iris$Species)
cm
```

```
##
## clus3 setosa versicolor virginica
##     1     49          0         0
##     2      1         21         2
##     3      0         29        48
```

Hierarchical Clustering

```
plot(h, hang = -0.1, labels = iris[["Species"]], cex = 0.5)
rect.hclust(h, k = 3)
```



Finding the best

```
clus_methods <- c("complete", "single", "average")
for (m in 1:3)
{
  cat("Method", clus_methods[m], "\n")
  hc <- hclust(d, meth = clus_methods[m])
  for (k in 2:5)
  {
    clusters <- cutree(hc, k)
    sil <- silhouette(clusters, d)
    cat("  k=", k, " -> s=", mean(sil[,3]), "\n")
  }
}
```

```
## Method complete
##   k= 2  -> s= 0.4408121
##   k= 3  -> s= 0.4496185
##   k= 4  -> s= 0.4106071
##   k= 5  -> s= 0.352063
## Method single
##   k= 2  -> s= 0.58175
##   k= 3  -> s= 0.5046456
##   k= 4  -> s= 0.4067465
##   k= 5  -> s= 0.3424089
## Method average
##   k= 2  -> s= 0.58175
##   k= 3  -> s= 0.4802669
##   k= 4  -> s= 0.4067465
##   k= 5  -> s= 0.3746013
```

Divisive Methods

- DIANA
  - Find cluster with largest diameter
  - Find observation in this cluster with maximum average distance to the rest of the group
  - Split this cluster between old cluster's center and this "outlier"
- DIANA algorithm is implemented in `cluster`
  - function `diana()`

```
di <- diana(iris[, -5], metric = "euclidean", stand = TRUE)
clusters <- cutree(di, 3)
table(clusters, iris$Species)
```

```
##
## clusters setosa versicolor virginica
##        1     50          0         0
##        2      0         11        33
##        3      0         39        17
```

Density-based Methods

- DBSCAN
  - Split points into 3 categories, using *reachability distance* and *count*
    - core points: contains more than *count* points within *reachability distance*
    - border points: not a core point but lies within *reachability distance* of a core point
    - noise points: neither core nor border points
  - Then create clusters out of core points
  - Assign border points to these clusters
- Number of clusters is automatically computed as a result

Density-based Methods

- implemented in `fpc` as `dbscan()`

```
d <- scale(iris[,-5])
db <- dbscan(d, eps=0.9, MinPts=5)
db
```

```
## dbscan Pts=150 MinPts=5 eps=0.9
##        0  1  2
## border 4  1  4
## seed   0 48 93
## total  4 49 97
```

```
table(db$cluster,iris$Species)
```

```
##
##     setosa versicolor virginica
##   0      1          0         3
##   1     49          0         0
##   2      0         50        47
```