Homework 3

Pattern Mining and Social Network Analysis

BOUYSSOU Gatien , de POURTALES Caroline, LAMBA Ankit

08 novembre, 2020

Contents

Parameters in association rules

There are parameters controlling the number of rules to be generated.

Support

Support is an indication of how frequently the itemset appears in the dataset.

$$Support(A \rightarrow B) = \frac{\text{Number of transaction with both A and B}}{\text{Total Number of transaction}} = P(A \cap B)$$

Confidence

Confidence is an indication of how often the rule has been found to be true.
This says how likely B is induced by A.

$$Confidence(A \rightarrow B) = \frac{\text{Number of transaction with both A and B}}{\text{Total Number of transaction with A}} = \frac{P(A \cap B)}{P(A)} = \frac{P_A(B)}{P(A)^2}$$

Lift

Lift is the factor by which, the co-occurence of A and B exceeds the expected probability of A and B co-occuring, had they been independent. So, higher the lift, higher the chance of A and B occurring together.

$$Lift(A \rightarrow B) = \frac{P(A \cap B)}{P(A) * P(B)}$$

Leverage

The leverage compares the frequency of A and B appearing together and the frequency that would be expected if A and B were independent.

$$Levarage(A \rightarrow B) = P(A \cap B) - P(A) \times P(B)$$

Therefore, if A and B independent :

$Levarage(A \rightarrow B) = 0$

Conviction

The conviction correspond to the frequency of items that are not B in the transaction over the frequency of B that don't contain A among all the transcations with B. Therefore, if A and B are independent the conviction should be equal to 1. When the confidence tends toward 1 the conviction tends toward infinity. It would mean that A and B are higly dependant.

$$Conviction(A \rightarrow B) = \frac{P(A) * P(\bar{B})}{P(A \cap \bar{B})}$$

or :

$$Conviction(A \rightarrow B) = \frac{1 - P(B)}{1 - \frac{P(A \cap B)}{P(A)}}$$

Apriori algorithm

Definition

Apriori searches for frequent itemset browsing the lattice of itemsets in breadth.
The database is scanned at each level of lattice. Additionally, Apriori uses a pruning technique based on the properties of the itemsets, which are: If an itemset is frequent, all its sub-sets are frequent and not need to be considered.

Example on Groceries data on R

The Groceries data set contains 30 days of real-world point-of-sale transaction data from a typical local grocery outlet. The data set contains 9835 transactions and the items are aggregated to 169 categories.

We can see the class of the dataset is :

```
## [1] "transactions"
## attr(,"package")
## [1] "arules"
```

Looking at some examples of transaction :

```
##     items
## [1] {citrus fruit,
##      semi-finished bread,
##      margarine,
##      ready soups}
## [2] {tropical fruit,
##      yogurt,
##      coffee}
## [3] {whole milk}
```

Let's apply Apriori algorithm on the dataset :

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.2    0.1    1 none FALSE            TRUE       5    0.03      1
##  maxlen target  ext
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 295
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.01s].
## sorting and recoding items ... [44 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 done [0.00s].
## writing ... [26 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

We have a set of associations rules.

```
## set of 26 rules
```

We can sort them by confidence and look at the first 5 (so the 5 rules with highest confidence).

```
##     lhs                     rhs                support    confidence coverage
## [1] {whipped/sour cream} => {whole milk}       0.03223183 0.4496454  0.07168277
## [2] {root vegetables}    => {whole milk}       0.04890696 0.4486940  0.10899847
## [3] {root vegetables}    => {other vegetables} 0.04738180 0.4347015  0.10899847
## [4] {tropical fruit}     => {whole milk}       0.04229792 0.4031008  0.10493137
## [5] {yogurt}             => {whole milk}       0.05602440 0.4016035  0.13950178
##     lift     count
## [1] 1.759754 317
## [2] 1.756031 481
## [3] 2.246605 466
## [4] 1.577595 416
## [5] 1.571735 551
```

Using Frequent itemset to find rules

Concept

We call also use the ruleInduction method to find closed frequent itemset.

ruleInduction has as attribute a method function.

Closed Frequent itemsets :

An itemset X is a closed frequent itemset in set S if X is both closed and frequent in S.

Eclat algorithm :

Mine frequent itemsets
This algorithm uses simple intersection operations for equivalence class clustering along with bottom-up lattice traversal.

Example on Adult data on R

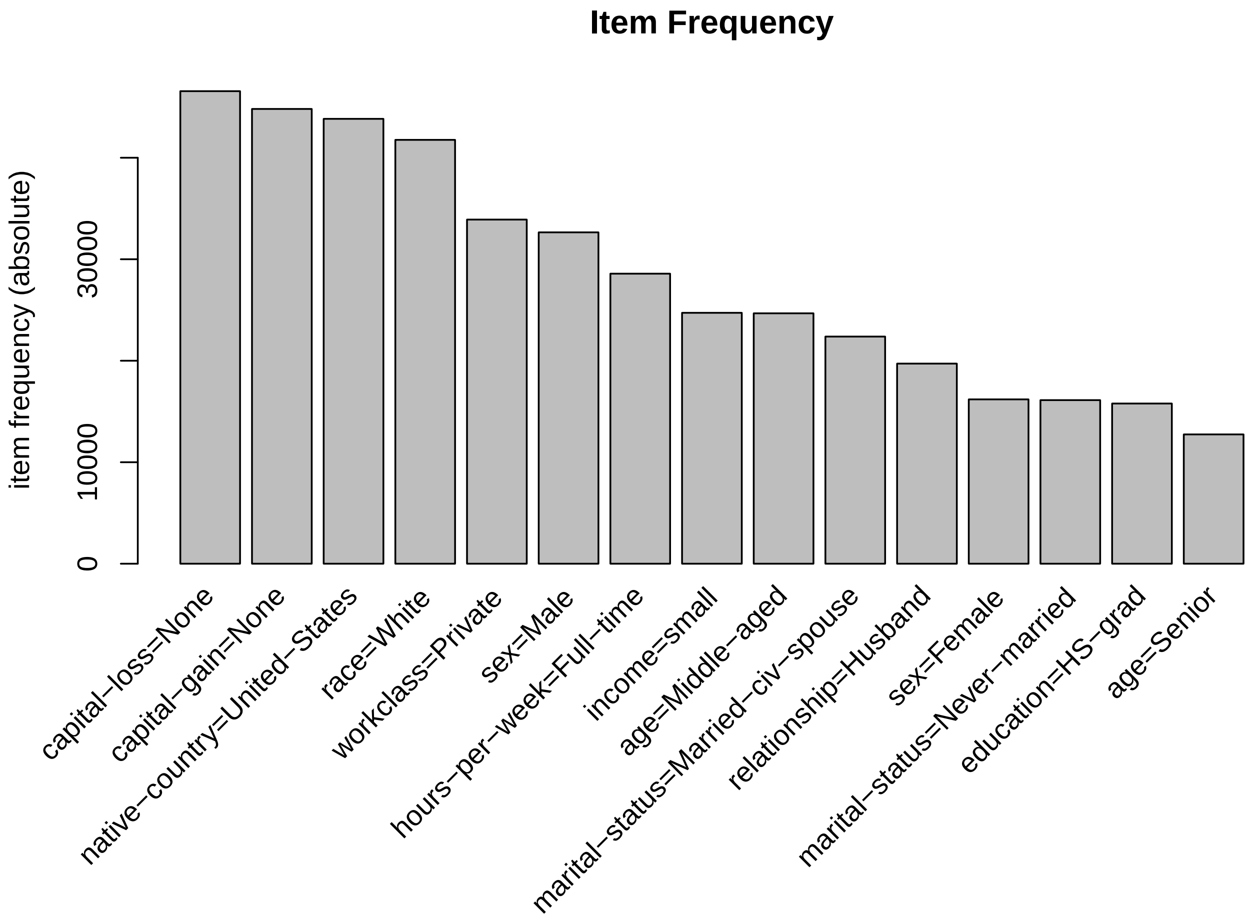
The Adult data set from R contains 48842 observations on the 15 variables (age, workclass, …).

```
## [1] "transactions"
## attr(,"package")
## [1] "arules"
```

We can look at the first transaction.

```
##     items                          transactionID
## [1] {age=Middle-aged,
##      workclass=State-gov,
##      education=Bachelors,
##      marital-status=Never-married,
##      occupation=Adm-clerical,
##      relationship=Not-in-family,
##      race=White,
##      sex=Male,
##      capital-gain=Low,
##      capital-loss=None,
##      hours-per-week=Full-time,
##      native-country=United-States,
##      income=small}                             1
```

We can find the 15 most common varaibles.



We apply ECLAT algorithm on the data set.
This returns the most frequent itemsets along with their support.

```
## Eclat
##
## parameter specification:
##  tidLists support minlen maxlen            target  ext
##     FALSE    0.01      1    100 frequent itemsets TRUE
##
## algorithmic control:
##  sparse sort verbose
##       7   -2    TRUE
##
## Absolute minimum support count: 488
##
## create itemset ...
## set transactions ...[115 item(s), 48842 transaction(s)] done [0.07s].
## sorting and recoding items ... [67 item(s)] done [0.01s].
## creating bit matrix ... [67 row(s), 48842 column(s)] done [0.01s].
## writing  ... [80228 set(s)] done [0.29s].
## Creating S4 object  ... done [0.03s].
```

```
##     items                                  support    transIdenticalToItemsets
## [1] {education=5th-6th,capital-loss=None}   0.01009377 493
## [2] {education=Doctorate,capital-loss=None} 0.01076942 526
## [3] {education=Doctorate,race=White}        0.01076942 526
##     count
## [1] 493
## [2] 526
## [3] 526
```

We can also find the rules from the most frequent itemset.

If in control method = "apriori" is used, a very simple rule induction method is used. All rules are mined from the transactions data set using Apriori with the minimal support found in itemsets. And in a second step all rules which do not stem from one of the itemsets are removed. This procedure will be in many cases very slow.

```
##     lhs                                    rhs                    support confidence     lift
## [1] {marital-status=Married-civ-spouse,
##      sex=Female,
##      capital-gain=None,
##      native-country=United-States,
##      income=large}                      => {relationship=Wife} 0.01095369  0.9870849 20.68263
## [2] {marital-status=Married-civ-spouse,
##      race=White,
##      sex=Female,
##      capital-gain=None,
##      income=large}                      => {relationship=Wife} 0.01076942  0.9868668 20.67806
## [3] {marital-status=Married-civ-spouse,
##      race=White,
##      sex=Female,
##      native-country=United-States,
##      income=large}                      => {relationship=Wife} 0.01238688  0.9837398 20.61254
```

If in control method = "ptree" is used, the transactions are counted into a prefix tree and then the rules are selectively generated using the counts in the tree. This is usually faster than the above approach.

Thanks to the implementation of ECLAT algorithm, we can find the rules with a specific given result.

For example, let's answer the question :
How to be rich ?

```
## Eclat
##
## parameter specification:
##  tidLists support minlen maxlen            target  ext
##     FALSE    0.01      1    200 frequent itemsets TRUE
##
## algorithmic control:
##  sparse sort verbose
##       7   -2    TRUE
##
## Absolute minimum support count: 488
##
## create itemset ...
## set transactions ...[115 item(s), 48842 transaction(s)] done [0.06s].
## sorting and recoding items ... [67 item(s)] done [0.01s].
## creating bit matrix ... [67 row(s), 48842 column(s)] done [0.01s].
## writing  ... [80228 set(s)] done [0.26s].
## Creating S4 object  ... done [0.02s].
```

We take the 3 best rules according to lift.

```
## set of 14 rules
```

```
##     lhs                                rhs                     support confidence     lift
## [1] {capital-loss=None,
##      hours-per-week=Over-time,
##      income=large}                  => {capital-gain=High} 0.01148602  0.1817887 5.253802
## [2] {race=White,
##      capital-loss=None,
##      hours-per-week=Over-time,
##      income=large}                  => {capital-gain=High} 0.01052373  0.1779778 5.143665
## [3] {capital-loss=None,
##      hours-per-week=Over-time,
##      native-country=United-States,
##      income=large}                  => {capital-gain=High} 0.01046231  0.1779248 5.142132
```

We see a pattern for people with a high capital gain : they have often a large income, work over-time and have no capital loss.

Example on mushroom data on python with scikit-learn

This database contains a lot of mushrooms with a set of characteristics. Each mushroom is classified either as edible or poisonous. The database has been found in kaggle and is available here : https://www.kaggle.com/uciml/mushroom-classification.

First, we want to have an overview of the data.

```
##   class cap-shape cap-surface  ... spore-print-color population habitat
## 0     p         x           s  ...                 k          s       u
## 1     e         x           s  ...                 n          n       g
## 2     e         b           s  ...                 n          n       m
## 3     p         x           y  ...                 k          s       u
## 4     e         x           s  ...                 n          a       g
## 
## [5 rows x 23 columns]
```

As we can see, each column contains values that are single characters. Their meaning is given by the file values_name.txt.

```
## 8124
```

Now, we want to know the data repartition for each columns.

```
## 
## class
## e    4208
## p    3916
## Name: class, dtype: int64
## 
## cap-shape
## x    3656
## f    3152
## k     828
## b     452
## s      32
## c       4
## Name: cap-shape, dtype: int64
## 
## cap-surface
## y    3244
## s    2556
## f    2320
## g       4
## Name: cap-surface, dtype: int64
## 
## cap-color
## n    2284
## g    1840
## e    1500
## y    1072
## w    1040
## b     168
## p     144
## c      44
## u      16
## r      16
## Name: cap-color, dtype: int64
##
## bruises
## f    4748
## t    3376
## Name: bruises, dtype: int64
##
## odor
## n    3528
## f    2160
## y     576
## s     576
## a     400
## l     400
## p     256
## c     192
## m      36
## Name: odor, dtype: int64
##
## gill-attachment
## f    7914
## a     210
## Name: gill-attachment, dtype: int64
##
## gill-spacing
## c    6812
## w    1312
## Name: gill-spacing, dtype: int64
##
## gill-size
## b    5612
## n    2512
## Name: gill-size, dtype: int64
##
## gill-color
## b    1728
## p    1492
## w    1202
## n    1048
## g     752
## h     732
## u     492
## k     408
## e      96
## y      86
## o      64
## r      24
## Name: gill-color, dtype: int64
##
## stalk-shape
## t    4608
## e    3516
## Name: stalk-shape, dtype: int64
##
```

```
## stalk-root
## b    3776
## ?    2480
## e    1120
## c     556
## r     192
## Name: stalk-root, dtype: int64
##
## stalk-surface-above-ring
## s    5176
## k    2372
## f     552
## y      24
## Name: stalk-surface-above-ring, dtype: int64
##
## stalk-surface-below-ring
## s    4936
## k    2304
## f     600
## y     284
## Name: stalk-surface-below-ring, dtype: int64
##
## stalk-color-above-ring
## w    4464
## p    1872
## g     576
## n     448
## b     432
## o     192
## e      96
## c      36
## y       8
## Name: stalk-color-above-ring, dtype: int64
##
## stalk-color-below-ring
## w    4384
## p    1872
## g     576
## n     512
## b     432
## o     192
## e      96
## c      36
## y      24
## Name: stalk-color-below-ring, dtype: int64
##
## veil-type
## p    8124
## Name: veil-type, dtype: int64
##
## veil-color
## w    7924
## n      96
## o      96
## y       8
## Name: veil-color, dtype: int64
##
## ring-number
## o    7488
## t     600
## n      36
## Name: ring-number, dtype: int64
##
## ring-type
## p    3968
## e    2776
## l    1296
## f      48
## n      36
## Name: ring-type, dtype: int64
##
## spore-print-color
## w    2388
## n    1968
## k    1872
## h    1632
## r      72
## y      48
## b      48
## u      48
## o      48
## Name: spore-print-color, dtype: int64
##
## population
## v    4040
## y    1712
## s    1248
## n     400
## a     384
## c     340
## Name: population, dtype: int64
##
## habitat
## d    3148
## g    2148
## p    1144
## l     832
## u     368
## m     292
## w     192
## Name: habitat, dtype: int64
```

As you can see the there is almost as much poisonous as edible mushrooms. Moreover, the dataset contains some unknown values in the column stalk-root. We are going to discard those rows to keep lines that are complete.

```
## 5644
```

```
## e    3488
## p    2156
## Name: class, dtype: int64
```

Even without the discarded lines the dataset still have plenty of data and the class label is almost balanced.

```
## [['b', 'c', 'x', 'f', 'k', 's'], ['bell', 'conical', 'convex', 'flat', 'knobbed', 'sunken']]
```

```
##        class cap-shape cap-surface  ... spore-print-color population  habitat
## 0  poisonous    convex      smooth  ...             black  scattered    urban
## 1     edible    convex      smooth  ...             brown   numerous  grasses
## 2     edible      bell      smooth  ...             brown   numerous  meadows
## 3  poisonous    convex       scaly  ...             black  scattered    urban
## 4     edible    convex      smooth  ...             brown   abundant  grasses
##
## [5 rows x 23 columns]
```

```
## ['abundant', 'almond', 'anise', 'attached', 'bell', 'black', 'broad', 'brown', 'bruises', 'buff', 'bu
```

```
## 64
```

```
##       support                                       itemsets  length
## 0    0.875266                                        (broad)       1
## 1    0.642452                                        (brown)       1
## 2    0.669029                                      (bulbous)       1
## 3    0.818568                                        (close)       1
## 4    0.618001                                       (edible)       1
## ..        ...                                            ...     ...
## 186  0.616584        (pendant, smooth, partial, white, free)       5
## 187  0.608079  (close, broad, partial, bulbous, white, free)       6
## 188  0.711552      (close, one, broad, partial, white, free)       6
## 189  0.600992     (one, broad, smooth, partial, white, free)       6
## 190  0.603827    (close, one, partial, bulbous, white, free)       6
##
## [191 rows x 3 columns]
```

```
##       support                             itemsets  length
## 4    0.618001                             (edible)       1
## 31   0.618001                       (free, edible)       2
## 32   0.609497                        (one, edible)       2
## 33   0.618001                    (partial, edible)       2
## 34   0.618001                      (white, edible)       2
## 89   0.609497                  (one, free, edible)       3
## 90   0.618001              (partial, free, edible)       3
## 91   0.618001                (white, free, edible)       3
## 92   0.609497               (one, partial, edible)       3
## 93   0.609497                 (white, one, edible)       3
## 94   0.618001             (white, partial, edible)       3
## 151  0.609497         (one, partial, free, edible)       4
## 152  0.609497           (white, one, free, edible)       4
## 153  0.618001       (white, partial, free, edible)       4
## 154  0.609497        (white, one, partial, edible)       4
## 184  0.609497  (one, partial, edible, white, free)       5
```

```
## Empty DataFrame
## Columns: [support, itemsets, length]
## Index: []
```

```
##            antecedents                 consequents  ...  leverage  conviction
## 19           (edible)                         (free)  ...  0.001971        inf
## 20           (edible)                          (one)  ...  0.008577   2.008505
## 21           (edible)                      (partial)  ...  0.000000        inf
## 22           (edible)                        (white)  ...  0.000000        inf
## 152     (one, edible)                         (free)  ...  0.001944        inf
## ..                ...                            ...  ...       ...        ...
## 818     (one, edible)       (white, partial, free)  ...  0.001944        inf
## 819  (partial, edible)           (white, one, free)  ...  0.008577   2.008505
## 820    (white, edible)         (one, partial, free)  ...  0.008577   2.008505
## 821     (free, edible)        (white, one, partial)  ...  0.008577   2.008505
## 822           (edible)  (white, one, partial, free)  ...  0.008577   2.008505
##
## [65 rows x 9 columns]
```

```
##       support                             itemsets
## 9    0.618001                             (edible)
## 129  0.618001                       (free, edible)
## 130  0.618001                    (partial, edible)
## 131  0.618001                      (white, edible)
## 132  0.609497                        (one, edible)
## 133  0.618001              (partial, free, edible)
## 134  0.618001                (white, free, edible)
## 135  0.609497                  (one, free, edible)
## 136  0.618001             (white, partial, edible)
## 137  0.609497               (one, partial, edible)
## 138  0.609497                 (white, one, edible)
## 139  0.618001       (white, partial, free, edible)
## 140  0.609497         (one, partial, free, edible)
## 141  0.609497           (white, one, free, edible)
## 142  0.609497        (white, one, partial, edible)
## 143  0.609497  (one, partial, edible, white, free)
```

```
## Empty DataFrame
## Columns: [antecedents, consequents, antecedent support, consequent support, support, confidence, lif
## Index: []
```

```
##          antecedents                     consequents  ...  leverage  conviction
## 1284           (free)                        (edible)  ...  0.001971    1.005203
## 1286        (partial)                        (edible)  ...  0.000000    1.000000
## 1288          (white)                        (edible)  ...  0.000000    1.000000
## 1290            (one)                        (edible)  ...  0.008577    1.023637
## 1292  (partial, free)                        (edible)  ...  0.001971    1.005203
## ...               ...                             ...  ...       ...         ...
## 1408    (white, free)          (one, partial, edible)  ...  0.001944    1.005019
## 1409            (one)  (white, partial, free, edible)  ...  0.008577    1.023637
## 1410        (partial)      (white, one, free, edible)  ...  0.000000    1.000000
## 1412          (white)    (one, partial, free, edible)  ...  0.000000    1.000000
## 1413           (free)   (white, one, partial, edible)  ...  0.001944    1.005019
##
## [65 rows x 9 columns]
```

Clustering with Apriori algorithm as dissimilarity measure

Concept

TO DO

Jaccard dissimilarity

A direct approach to cluster itemsets is to define a distance metric between two itemsets X_i and X_j. A good choice is the Jaccard distance defined as :

$$d(X_i, X_j) = \frac{|X_i \cap X_j|}{|X_i \cup X_j|}$$

The distance simply is the number of items that Xi and Xj have in common divided by the number of unique items in both sets.

Affinity dissimilarity

TO DO

Example on tennis data on R

We use a dataset from the Wimbledon tennis tournament for Women in 2013. We will predict the result for player 1 (win=1 or loose=0) based on : the number of aces won by each player, and, the number of unforced errors commited by both players. The data set is a subset of a data set from https://archive.ics.uci.edu/ml/datasets/Tennis+Major+Tournament+Match+Statistics.

```
##         Player1      Player2 Result ACE.1 UFE.1 ACE.2 UFE.2
## 1     M.Koehler   V.Azarenka      0     2    18     3    14
## 2    E.Baltacha   F.Pennetta      0     0    10     4    14
## 3     S-W.Hsieh      T.Maria      1     1    13     2    29
## 4      A.Cornet       V.King      1     4    30     0    45
## 5  Y.Putintseva   K.Flipkens      0     2    28     6    19
## 6 A.Tomljanovic B.Jovanovski      0     6    42    11    40
```

We can transform the tennis data set into a transaction data set.

We can look at the 3 first transactions.

```
##     items                                                 transactionID
## [1] {Result=0,ACE.1=Low,UFE.1=Low,ACE.2=Low,UFE.2=Low}   1
## [2] {Result=0,ACE.1=None,UFE.1=Low,ACE.2=High,UFE.2=Low} 2
## [3] {Result=1,ACE.1=Low,UFE.1=Low,ACE.2=Low,UFE.2=High}  3
```

We can restrict the rules to the result rhs="Result=1" which means Player-1 winner.

The associations rules for Player-1 winning are :

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.3    0.1    1 none FALSE            TRUE       5    0.15      1
##  maxlen target  ext
##      10  rules TRUE
##
```

```
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 17
##
## set item appearances ...[1 item(s)] done [0.00s].
## set transactions ...[12 item(s), 118 transaction(s)] done [0.00s].
## sorting and recoding items ... [11 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [13 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
## set of 13 rules
```

```
##     lhs                         rhs        support   confidence coverage
## [1] {ACE.1=High,UFE.1=Low} => {Result=1} 0.1525424 0.8181818  0.1864407
## [2] {ACE.1=High}           => {Result=1} 0.2881356 0.6938776  0.4152542
## [3] {ACE.1=High,UFE.2=Low} => {Result=1} 0.1694915 0.6451613  0.2627119
## [4] {ACE.2=Low}            => {Result=1} 0.2457627 0.6170213  0.3983051
## [5] {UFE.1=Low}            => {Result=1} 0.3220339 0.6129032  0.5254237
##     lift     count
## [1] 1.532468 18
## [2] 1.299644 34
## [3] 1.208397 20
## [4] 1.155691 29
## [5] 1.147977 38
```

These rules look correct : either a player-1 winning make a lot of aces and few unforced errors or the player-2 make few aces.

We can also restrict the rules to the result rhs="Result=0" which means Player-1 loosing.

The associations rules for Player-1 loosing :

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.3    0.1    1 none FALSE            TRUE       5    0.15      1
##  maxlen target  ext
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 17
##
## set item appearances ...[1 item(s)] done [0.00s].
## set transactions ...[12 item(s), 118 transaction(s)] done [0.00s].
## sorting and recoding items ... [11 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [10 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
## set of 10 rules
```

```
##     lhs             rhs        support   confidence coverage  lift     count
## [1] {ACE.2=High} => {Result=0} 0.2372881 0.6086957  0.3898305 1.305929 28
## [2] {UFE.1=High} => {Result=0} 0.2627119 0.5535714  0.4745763 1.187662 31
## [3] {ACE.1=Low}  => {Result=0} 0.2372881 0.5283019  0.4491525 1.133448 28
```

These rules look correct : either player-1 is loosing because player-2 makes a lot of aces or because he does a lot of unforced errors or player-2 makes a lot of aces.

Now let's look at all the associations rules leading to "Result".
All the rules with Result as association :

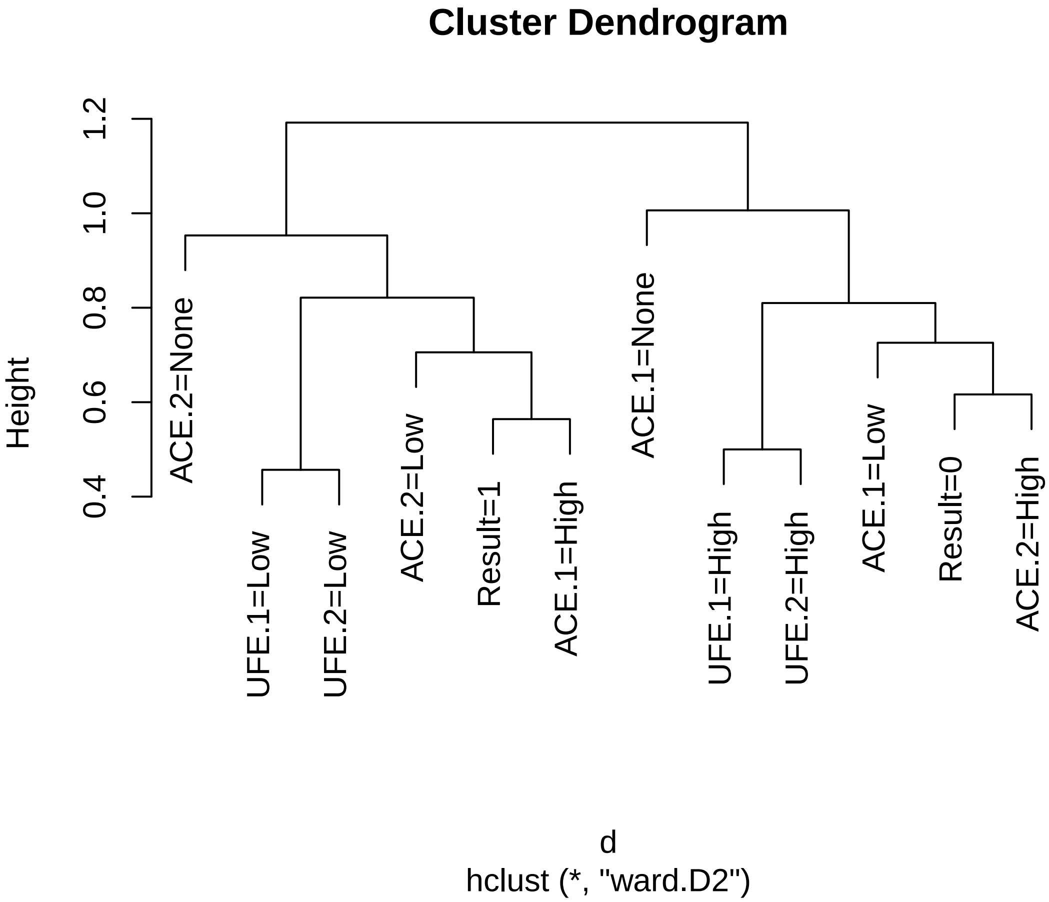
```
## Apriori
## 
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.4    0.1    1 none FALSE            TRUE       5     0.1      1
##  maxlen target  ext
##      10  rules TRUE
## 
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
## 
## Absolute minimum support count: 11 
## 
## set item appearances ...[2 item(s)] done [0.00s].
## set transactions ...[12 item(s), 118 transaction(s)] done [0.00s].
## sorting and recoding items ... [12 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [46 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
## set of 46 rules
```

```
##     lhs                                 rhs        support   confidence
## [1] {ACE.1=None}                     => {Result=0} 0.1016949 0.7500000 
## [2] {ACE.1=High,UFE.1=Low}           => {Result=1} 0.1525424 0.8181818 
## [3] {UFE.1=High,ACE.2=High}          => {Result=0} 0.1186441 0.7000000 
## [4] {ACE.2=High,UFE.2=Low}           => {Result=0} 0.1355932 0.6956522 
## [5] {ACE.1=High,UFE.1=Low,UFE.2=Low} => {Result=1} 0.1271186 0.7894737 
##     coverage  lift     count
## [1] 0.1355932 1.609091 12   
## [2] 0.1864407 1.532468 18   
## [3] 0.1694915 1.501818 14   
## [4] 0.1949153 1.492490 16   
## [5] 0.1610169 1.478697 15
```

Firstly let's look at the clustering of items

Cluster the items



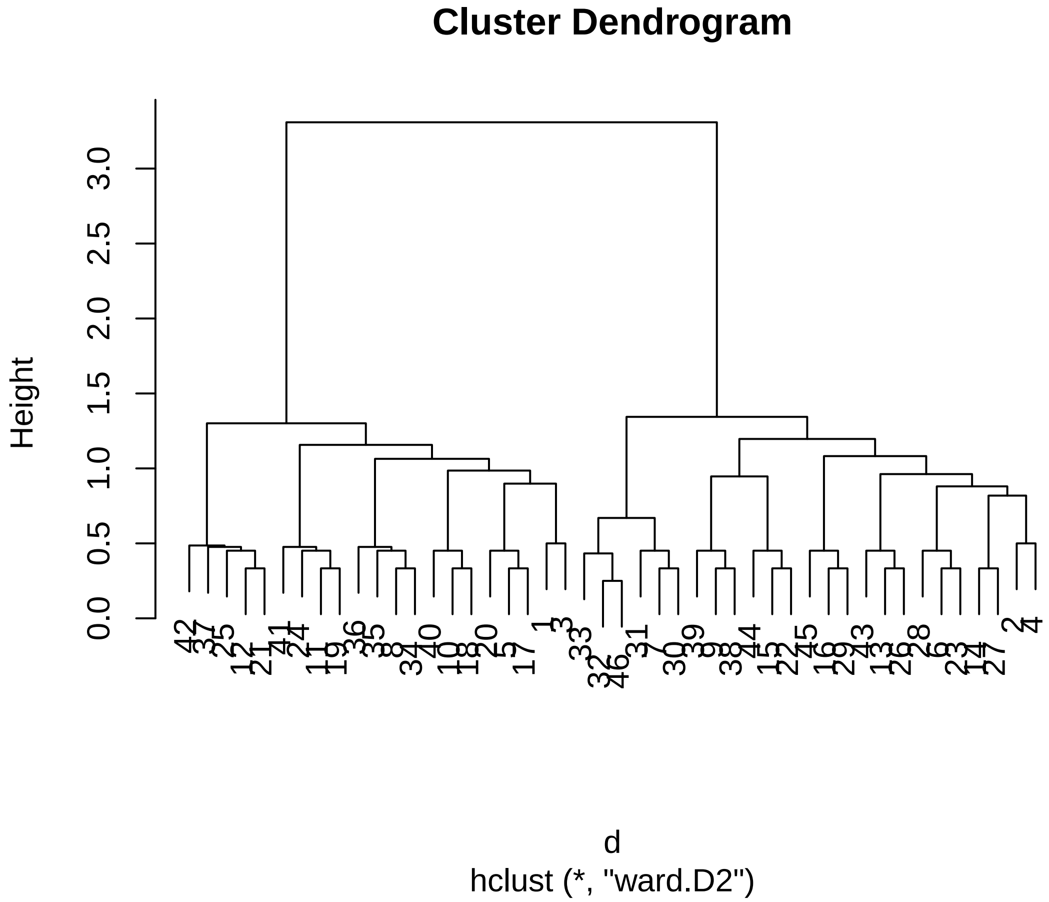
We can see two clusters on the dendogramm. One contains Result=1 and the other contains Result=0. Moreover in the cluster with the branch Result=1, we can also see that the number of aces made by player-1 is high and the number of unforced errors is low. Also in this cluster player-2 made few or none aces. In the other cluster, it is the opposite. Player-1 makes few or none aces whereas player-2 makes a lot.

So it seems that this clustering manage to cluster data linked to the result together.

Now let's try to cluster the rules.

Cluster the rules

- With Jaccard measure :



If we cut the dendogramm in two clusters. We can look at the first cluster.

And at the second cluster.

This clustering regroups Player-1 winner together and Player-2 together.

We used Jaccard distance but there is also the affinity measure. However it gave worst result than Jaccard dissimilarity.

Classification and association rules

CBA Algorithm

Build a classifier based on association rules mined for an input dataset.

Implementation the CBA algorithm with the M1 or M2 pruning strategy introduced by Liu, et al. (1998).

Candidate classification association rules (CARs) are mined with the standard APRIORI algorithm. Rules are ranked by confidence, support and size. Then either the M1 or M2 algorithm are used to perform database coverage pruning and to determin the number of rules to use and the default class.

TO DO DEFINITION

Example on tennis data on R

Recall from Homework 1 With Random Forest, the accuracy rate was 0.6931818.
With Logistic regression it was 0.7667.

Classification using chi2 discretization CBA can take as input a classic non-transaction dataset as tennis. We just have to choose the discretization method in parameter.

```
##      lhs                                    rhs        support confidence
## [1]  {ACE.1=[4.5,5.5)}                   => {Result=1} 0.0612  1.000
## [2]  {UFE.1=[12.5,13.5)}                 => {Result=1} 0.0612  1.000
## [3]  {UFE.1=[18.5, Inf],ACE.2=[1.5,2.5)}  => {Result=0} 0.0714  0.875
## [4]  {ACE.1=[1.5,2.5),UFE.1=[18.5, Inf]}  => {Result=0} 0.0714  0.875
## [5]  {ACE.1=[3.5,4.5)}                   => {Result=1} 0.0612  0.857
## [6]  {ACE.1=[5.5, Inf],ACE.2=[-Inf,1.5)}  => {Result=1} 0.0612  0.857
## [7]  {ACE.2=[-Inf,1.5)}                  => {Result=1} 0.2857  0.683
## [8]  {ACE.1=[5.5, Inf]}                  => {Result=1} 0.1224  0.667
## [9]  {UFE.1=[18.5, Inf],ACE.2=[3.5, Inf]} => {Result=0} 0.1020  0.667
## [10] {ACE.1=[-Inf,1.5),UFE.2=[14.5, Inf]} => {Result=1} 0.1633  0.640
## [11] {UFE.2=[14.5, Inf]}                 => {Result=1} 0.3776  0.627
## [12] {}                                  => {Result=0} 0.4388  0.439
##      coverage lift count size coveredTransactions totalErrors
## [1]  0.0612   1.78  6    2     6                  43
## [2]  0.0612   1.78  6    2     6                  43
## [3]  0.0816   1.99  7    3     8                  37
## [4]  0.0816   1.99  7    3     6                  33
## [5]  0.0714   1.53  6    2     7                  33
## [6]  0.0714   1.53  6    3     7                  33
## [7]  0.4184   1.22 28    2    19                  32
## [8]  0.1837   1.19 12    2     7                  31
## [9]  0.1531   1.52 10    3     4                  29
## [10] 0.2551   1.14 16    3    10                  23
## [11] 0.6020   1.12 37    2     5                  22
## [12]     NA   1.00 98    1    13                  22
```

```
##                      true
## classifier.prediction 0 1
##                     0 7 2
##                     1 5 6
```

The accuracy rate is :

```
## [1] 0.65
```

The sensitivity is the percentage of true output giving Player1-winner among the population of true Player1-winner :

```
## [1] 0.75
```

The specificity is the percentage of true output giving Player2-winner (= Player1-looser) among the population of true Player2-winner:

```
## [1] 0.583
```

The precision is the percentage of true output giving Player1-winner among all the outputs giving Player1-winner (even if not winner) :

```
## [1] 0.545
```

So the F_Mesure is :

```
## [1] 0.632
```

Direct classification from homemade transactions We can also use the transactions we created before.
ACE.1 and ACE.2 take for value either None, or Low or High.
UFE.1 and UFE.2 take for value either Low or High.

```
##     lhs             rhs        support confidence coverage lift count size coveredTransactions totalE
## [1] {ACE.1=High,
##      UFE.1=Low,
##      ACE.2=None} => {Result=1}  0.0510          1   0.0510 1.78     5    4                   5
## [2] {ACE.1=Low,
##      ACE.2=High,
##      UFE.2=Low}  => {Result=0}  0.0510          1   0.0510 2.28     5    4                   5
## [3] {ACE.1=Low,
##      UFE.1=Low,
##      ACE.2=High,
##      UFE.2=High} => {Result=1}  0.0510          1   0.0510 1.78     5    5                   5
## [4] {ACE.1=None,
##      ACE.2=Low,
##      UFE.2=Low}  => {Result=0}  0.0306          1   0.0306 2.28     3    4                   3
```

```
##
## classifier.prediction 0 1
##                     0 9 2
##                     1 3 6
```

The accuracy rate is :

```
## [1] 0.75
```

So the accuracy rate is better with our homemade transaction. It makes sense because discretization of integers is difficult to handle.

The sensitivity is the percentage of true output giving Player1-winner among the population of true Player1-winner :

```
## [1] 0.75
```

The specificity is the percentage of true output giving Player2-winner (= Player1-looser) among the population of true Player2-winner:

```
## [1] 0.75
```

The precision is the percentage of true output giving Player1-winner among all the outputs giving Player1-winner (even if not winner) :

```
## [1] 0.667
```

So the F_Mesure is :

```
## [1] 0.706
```

This classification is also as good as logistic regression. It gives very good result.