Association Rule Mining

Yannis Kotidis

kotidis@aueb.gr

Professor, Department of Informatics

Athens University of Economics and Business

Suggested Reading

- **Data Mining: Concepts and Techniques**, 3rd Edition (The Morgan Kaufmann Series in Data Management Systems) 3rd Edition, by Jiawei Han, Micheline Kamber, Jian Pei (Chapter 6)

- **Mining of Massive Datasets**, 2nd Edition, by Jure Leskovec, Anand Rajaraman, Jeffrey David Ullman, Stanford University (Chapter 6)

Data Mining

- The process of analyzing data to identify patterns or relationships
- Has become a well-established discipline related to Artificial Intelligence and Statistical Analysis
  - Led by advances in computer hardware and our ability to analyze big datasets
    - Data warehousing, BI, Cloud Computing

Association Rule Mining

- Finding frequent patterns (associations) among sets of items in transactional databases
  - Basket data analysis, catalog design, direct mailing,…
- *Basic question:* ***"Which groups or sets of items are customers likely to purchase on a given trip to the store?"***
- Learned patterns are used to construct rules
  - buys(x, "diapers") $\rightarrow$ buys(x, "beers") [5%, 60%]

What to do with rule Diapers→Beers ?

- Enhance observed behavior
  - Place products in proximity to further encourage the combined sale
  - Increase the price of diapers but put beer in discount for a combined sale

- Put products at opposite ends of the store to make customers spend more time (and buy more products) at the store

More ideas

- Assume laptops and printers are frequently sold together
  - Place a higher-margin printer near the laptop section
  - Take a soon to be updated software suite and bundle it in an offer with laptops and printers

- See https://www.kdnuggets.com/news/98/n01.html
  - What Wal-Mart might do with Barbie doll → Candy bars association rule

Basic Concepts

- Each transaction is a set of items (e.g. purchased by a customer in a visit)
- Example: Basket Data analysis

T1: Milk, Diaper, Chocolate

T2: Diaper, Beer, Meat

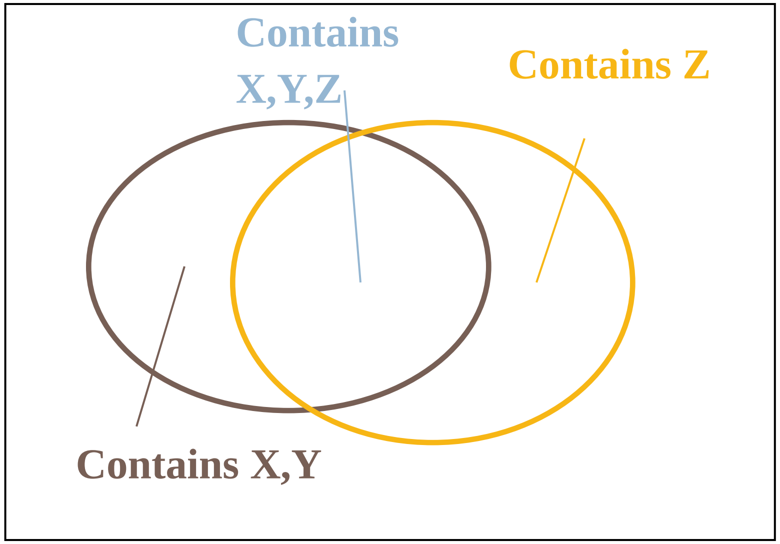
T3: Sugar, Beer, Diaper

…

Inferred rule:

buys(x, "Diaper") → buys(x, "Beer") [5%, 67%]

Support and Confidence



TID	Items
T1	A,C
T2	A,C,D
T3	A,E
T4	D,E,F,G

- Given rule $X,Y \Rightarrow Z$
- **Support**: probability that a transaction contains {X,Y,Z}
  - s=P[X and Y and Z]
- **Confidence**: probability that a transaction having {X,Y} also contains Z
  - c=P[Z|X,Y]

Let minimum support 50%, and minimum confidence 50%, we have

$$A \Rightarrow C \ (\qquad\qquad)$$
$$C \Rightarrow A \ (\qquad\qquad)$$

Problem formulation

- **Given**
  - a set of 'market baskets' (=binary matrix, of N rows/baskets and M columns/products)
  - min-support 's' and
  - min-confidence 'c'

Tid	Diaper	Meat	Milk	Beer
1	1	0	1	1
2	1	1	0	0
3	1	1	0	0
4	0	1	1	0

- **Find**
  - all the rules with:

support $\geq$ s & confidence $\geq$ c

From rules to itemsets

- Find frequent itemsets
  - e.g. {X,Y,Z}
  - "Frequent" means support>= s (min-support)
- Once we have a 'frequent itemset', we can find out the qualifying rules easily (how?)

$$\text{Support}(X,Y \rightarrow Z) = \text{Freq}(\{X,Y,Z\})$$

$$\begin{aligned}\text{Conf}(X,Y \rightarrow Z) &= P[Z|X,Y] = P[X,Y,Z]/P[X,Y] \\ &= \text{Freq}(\{X,Y,Z\}) \,/\, \text{Freq}(\{X,Y\})\end{aligned}$$

- Thus, let's focus on how to find frequent itemsets

Brute-force Frequent Itemsets Counting

- Scan database once; keep 2^M-1 counters
  - One counter for each of {A}, {B}, {C}, …., {A,B}, {A,C}, {A,D}, … {B,C}, {B,D}, {B,E},… {A,B,C}, …
- Example (M=3)

Itemset	Counter
{A}	0 +1
{B}	0 +1
{C}	0
{A,B}	0 +1
{A,C}	0
{B,C}	0
{A,B,C}	0

Basket 1: A,B

Brute-force Frequent Itemsets Counting

- Scan database once; keep 2^M-1 counters
  - One counter for each of {A}, {B}, {C}, …., {A,B}, {A,C}, {A,D}, … {B,C}, {B,D}, {B,E},… {A,B,C}, …
- Example (M=3)

Itemset	Counter
{A}	1
{B}	1 +1
{C}	0
{A,B}	1
{A,C}	0
{B,C}	0
{A,B,C}	0

Basket 1: A,B
Basket 2: B

Brute-force Frequent Itemsets Counting

- Scan database once; keep 2^M-1 counters
  - One counter for each of {A}, {B}, {C}, …., {A,B}, {A,C}, {A,D}, … {B,C}, {B,D}, {B,E},… {A,B,C}, …
- Drawback?
  - 2^{1000} is prohibitive...
  - E.g. 16GB RAM ($=2^{34}$ bits) stores 2^{29} counters using $32=2^5$ bit integers
- Improvement?
  - Scan the db M times, looking for 1-, 2-, etc itemsets

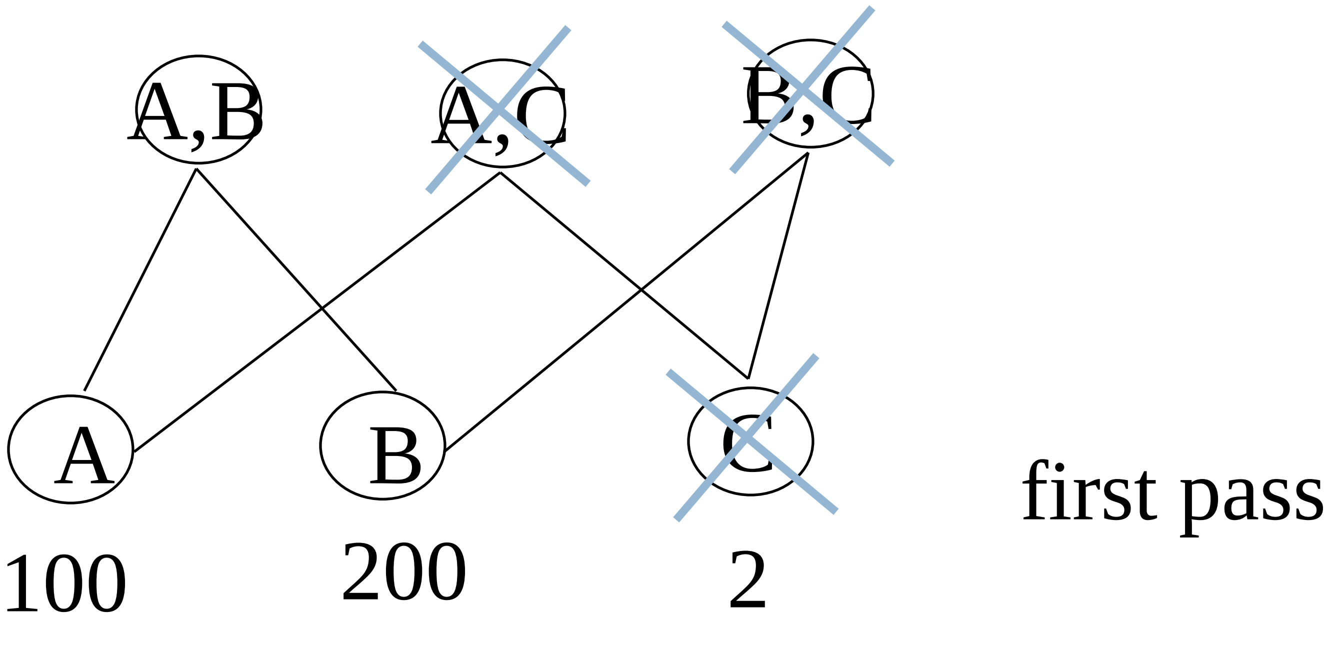
Assume {A},{B},{C} (M=3)

A B C first pass

100 200 2

min-sup:10

Move on



Anti-monotonicity property

- If an itemset fails to be frequent, so will every superset of it
  - hence all supersets can be pruned
- A subset of a frequent itemset must also be a frequent itemset
  - i.e., if $\{AB\}$ is a frequent itemset, both $\{A\}$ and $\{B\}$ should be a frequent itemset

- Sketch of the (famous!) 'a-priori' algorithm
  - Let $L(i\text{-}1)$ be the set of large (=frequent) itemsets with $i\text{-}1$ elements
  - Let $C(i)$ be the set of candidate itemsets (of size i)

The A-priori Algorithm

- Compute L(1), by scanning the database.
- repeat, for i=2,3...,
  - **'join'** L(i-1) with itself, to generate C(i)
    - two itemset can be joined, if they agree on their first *i-2* elements
  - **prune** the itemsets of C(i) (how?)
  - scan the db, finding the counts of the C(i) itemsets – those that reach or exceed threshold are placed in L(i)
  - unless L(i) is empty, repeat the loop

An Example

notation for itemset {a,c,e}

notation for itemset {b,c,d}

- L_3={*abc, abd, acd, ace, bcd*}
- Self-joining: L_3*L_3
  - *abcd* *produced* from abc and abd
  - *acde* *produced* from acd and ace
- Pruning:
  - *acde* is removed because *ade* is not in L_3
- C_4={*abcd*}

Note on Self-joining: $L_1 * L_1$

- *The result is essentially a Cartesian Product (**x**)*
- $L_1 = \{a, b, c, d, e\}$
- $C_2 = L_1$ **x** $L_1 = \{ab, ac, ad, ae, bc, bd, be, cd, ce, de\}$
- *No pruning possible (why?)*

Example 2

Min Support = 2 (50%)

Database D

TID	Items
100	A,C,D
200	B,C,E
300	A,B,C,E
400	B E

Scan D

C_1

itemset	sup.
{A}	2
{B}	3
{C}	3
{D}	1
{E}	3

L_1

itemset	sup.
{A}	2
{B}	3
{C}	3
{E}	3

C_2

itemset
{A,B}
{A,C}
{A,E}
{B,C}
{B,E}
{C,E}

Scan D

C_2

itemset	sup
{A,B}	1
{A,C}	2
{A,E}	1
{B,C}	2
{B,E}	3
{C,E}	2

L_2

itemset	sup
{A,C}	2
{B,C}	2
{B,E}	3
{C,E}	2

C_3

itemset
{B,C,E}

Scan D

L_3

itemset	sup
{B,C,E}	2

From Itemsets to Association Rules

- Itemset {B,C,E} is frequent (support=50%)
- Consider rule B,C →E
  - Support(B,C →E) = P[B,C,E] = 50%
  - Confidence(B,C →E) = P[B,C,E]/P[B,C]=2/2=100%
- Thus : B,C→E [50%,100%]
- More rules?
- Also look at L_2

MIN-SUPPORT = 50%
MIN-CONFIDENCE=90%

Exercise 3

- Frequent Itemsets
  - {A,B,C} support = 50%, {A,B} support = 50%, {A,C} support=80%, {B,C} support = 80%, {A}=90%, {B}=90%, {C}=90%
- A,B→C [50%, 100%] (OK, exceeds thresholds)
- Reject the following (confidence < 90%)
  - A,C→B [50%, 62.5%]
  - B,C→A [50%, 62.5%]
  - A→B [50% , 55.5%]
    - (also B→A, A→C, C→A, B→C, C→B)

Apache Spark MLlib Example

- Modified example from https://spark.apache.org/docs/latest/ml-frequent-pattern-mining.html

- In addition to *association rule mining*, library provides common learning algorithms such as classification, regression, clustering, and collaborative filtering, feature extraction, transformation, dimensionality reduction, and selection

Define input dataset, convert to DF


```
scala> val dataset = spark.createDataset(Seq(
     |   "A C D",
     |   "B C E",
     |   "A B C E",
     |   "B E")
     | ).map(t => t.split(" ")).toDF("items")

scala> dataset.show
```

Database D

TID	Items
100	A,C,D
200	B,C,E
300	A,B,C,E
400	B E

```
scala> dataset.show
+------------+
|       items|
+------------+
|   [A, C, D]|
|   [B, C, E]|
|[A, B, C, E]|
|      [B, E]|
+------------+
```

Execute FPGrowth Algorithm


```
val fpgrowth = new
FPGrowth().setItemsCol("items").setMinSupport(0.5).setMinConfidence(0.5)
val model = fpgrowth.fit(dataset)

// Display frequent itemsets.
model.freqItemsets.show()
```

Database D

TID	Items
100	A,C,D
200	B,C,E
300	A,B,C,E
400	B E

L_1

itemset	sup.
{A}	2
{B}	3
{C}	3
{E}	3

L_2

itemset	sup
{A,C}	2
{B,C}	2
{B,E}	3
{C,E}	2

L_3

itemset	sup
{B,C,E}	2

```
scala> model.freqItemsets.show()
+---------+----+
|    items|freq|
+---------+----+
|      [E]|   3|
|   [E, C]|   2|
|[E, C, B]|   2|
|   [E, B]|   3|
|      [B]|   3|
|      [C]|   3|
|   [C, B]|   2|
|      [A]|   2|
|   [A, C]|   2|
+---------+----+
```

List rules with their confidence


```
scala> model.associationRules.show()
```

```
scala> model.associationRules.show()
+----------+----------+------------------+
|antecedent|consequent|        confidence|
+----------+----------+------------------+
|       [E]|       [C]|0.6666666666666666|
|       [E]|       [B]|               1.0|
|       [B]|       [E]|               1.0|
|       [B]|       [C]|0.6666666666666666|
|    [E, B]|       [C]|0.6666666666666666|
|       [A]|       [C]|               1.0|
|    [C, B]|       [E]|               1.0|
|       [C]|       [E]|0.6666666666666666|
|       [C]|       [B]|0.6666666666666666|
|       [C]|       [A]|0.6666666666666666|
|    [E, C]|       [B]|               1.0|
+----------+----------+------------------+


scala>
```

Database D

TID	Items
100	A,C,D
200	B,C,E
300	A,B,C,E
400	B E

Use rules to predict new purchases


```
scala> val newCustomer = spark.createDataset(Seq("A","B C")).map(t => t.split(" ")).toDF("items")
newCustomer: org.apache.spark.sql.DataFrame = [items: array<string>]

scala> newCustomer.show
+------+
| items|
+------+
|   [A]|
|[B, C]|
+------+


scala> model.transform(newCustomer).show()
+------+----------+
| items|prediction|
+------+----------+
|   [A]|       [C]|
|[B, C]|    [E, A]|
+------+----------+


scala>
```

```
scala> model.associationRules.show()
+----------+----------+------------------+
|antecedent|consequent|        confidence|
+----------+----------+------------------+
|       [E]|       [C]|0.6666666666666666|
|       [E]|       [B]|               1.0|
|       [B]|       [E]|               1.0|
|       [B]|       [C]|0.6666666666666666|
|    [E, B]|       [C]|0.6666666666666666|
|       [A]|       [C]|               1.0|
|    [C, B]|       [E]|               1.0|
|       [C]|       [E]|0.6666666666666666|
|       [C]|       [B]|0.6666666666666666|
|       [C]|       [A]|0.6666666666666666|
|    [E, C]|       [B]|               1.0|
+----------+----------+------------------+


scala>
```

More uses of Association Rules

(MMDS book)

- Related concepts: Let items be words, and let baskets be documents (e.g., Web pages, blogs, tweets).
  - Brad and Angelina appear together
- Plagiarism: Let the items be documents and the baskets be sentences. An item (doc) is “in” a basket (sentence) if the sentence is in the document.
  - Look for pairs of items (docs) that appear together in baskets (sentences).
- Biomarkers: Let the items be biomarkers such as genes or blood proteins, and diseases. Each basket is the set of data about a patient: list of biomarkers and deseses
  - A frequent itemset that consists of one disease and one or more biomarkers suggests a test for the disease.

Hot vs Not-so-hot items

- Most people buy milk, vegetabples, soda, snacs etc. in their trip to the store
- Other products are not that common (e.g. windscreen cleaners, sushi
- How to choose a good min-support threshold?

- A global, low threshold will generate many rules from the frequent items

Idea 1: Separate hot from cold

- Partition the data into several subsets, each of which contains only items of similar frequencies.
- Perform association rule mining for each subset using a different min-support threshold.

- Caveat: can not generate rules spanning items from different subsets (e.g. Milk → Sushi)

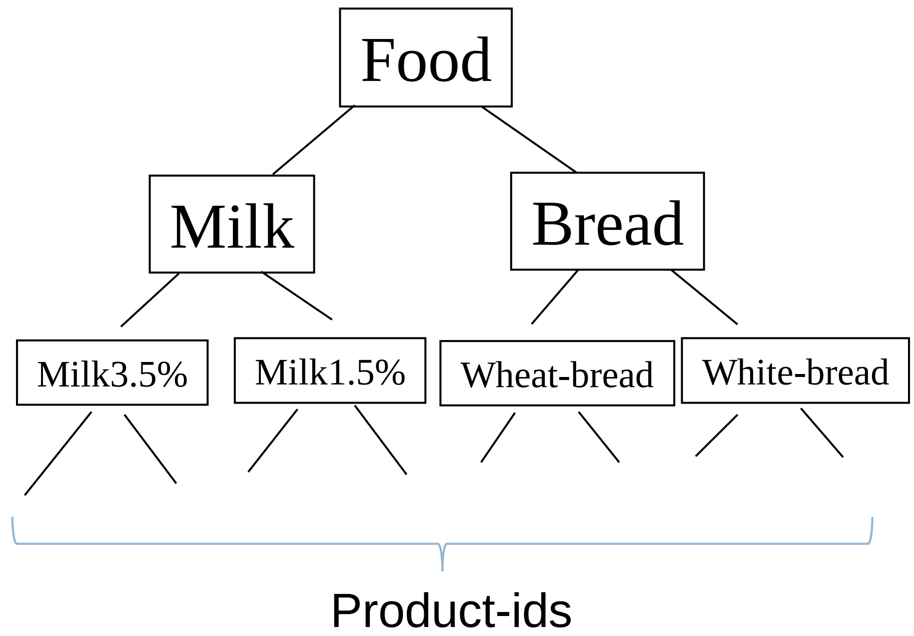
Idea 2: Use multiple thresholds

- Assign a different minimum support threshold per item (or group of items based on their frequencies)
  - E.g. min-sup(Milk) = 10%, min-sup(Sushi) = 5%
- When considering an itemset use the minimum min-sup() value of its elements
  - E.g. min-sup({Mik,Sushi}) = min(5%,10%)=5%
- Thus, rules need to satisfy different minimum supports depending on what items are in the rules

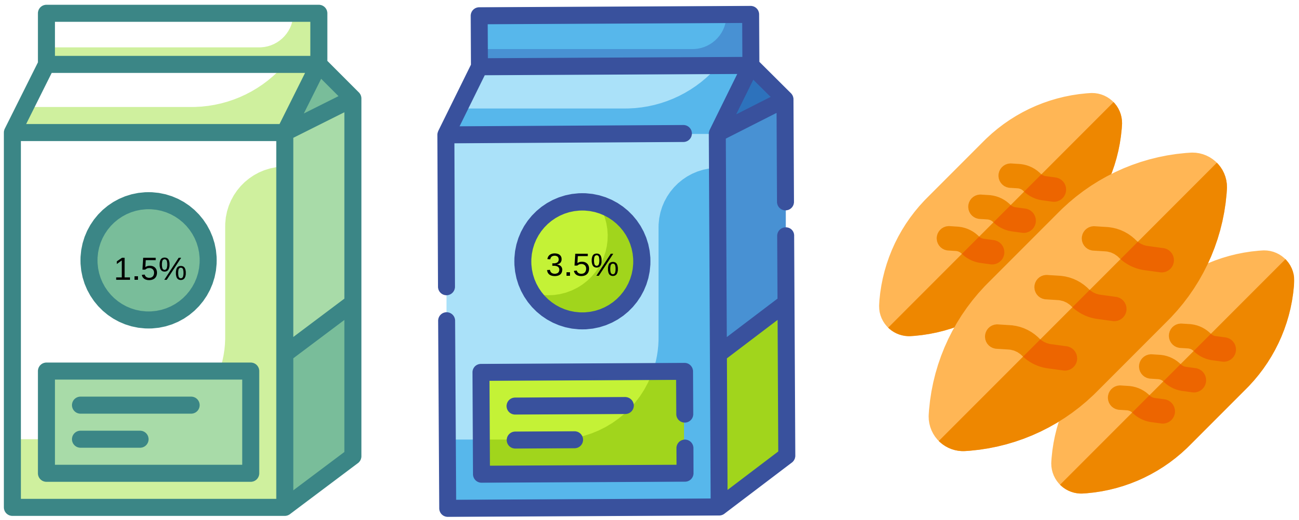
Multiple-Level Association Rules

- Items often form hierarchy
  - Recall dimension hierarchies in data warehousing
- Rules regarding itemsets at appropriate levels could be quite useful:

1.5% Milk $\Rightarrow$ Wheat bread



Shopping Cart → Itemset



P144 P2157 P11

{P144, Milk1.5%, Milk, P2157, Milk3.5%, P11, White-bread, Bread}

Performance considerations?

Quantitative Association Rules

- Boolean rules (categorical values):

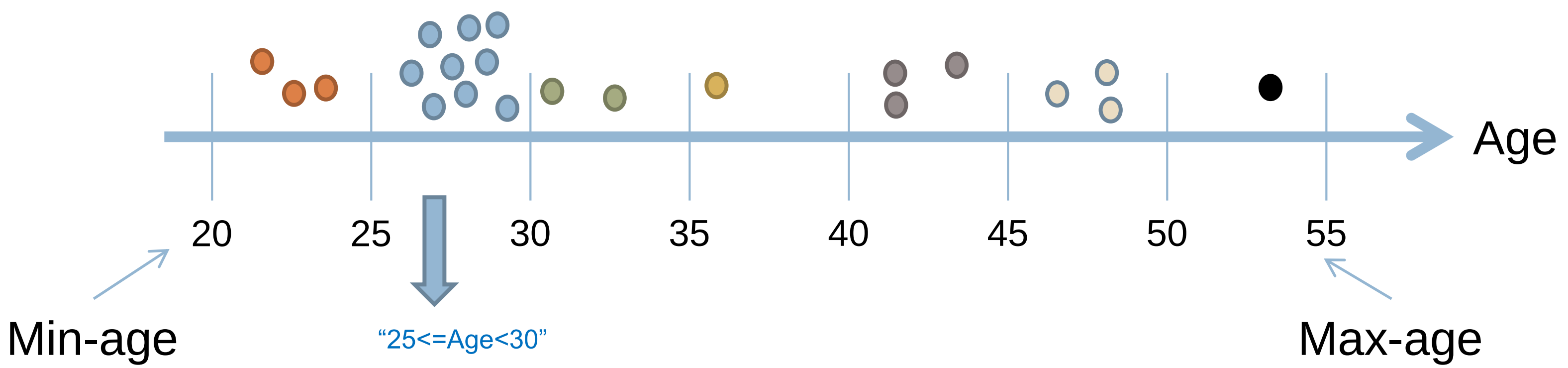
  buys(x, "Bread") ^ buys(x, "Diapers") → buys(x, "Beer") [20%, 60%]

- Quantitative rules (interval values):

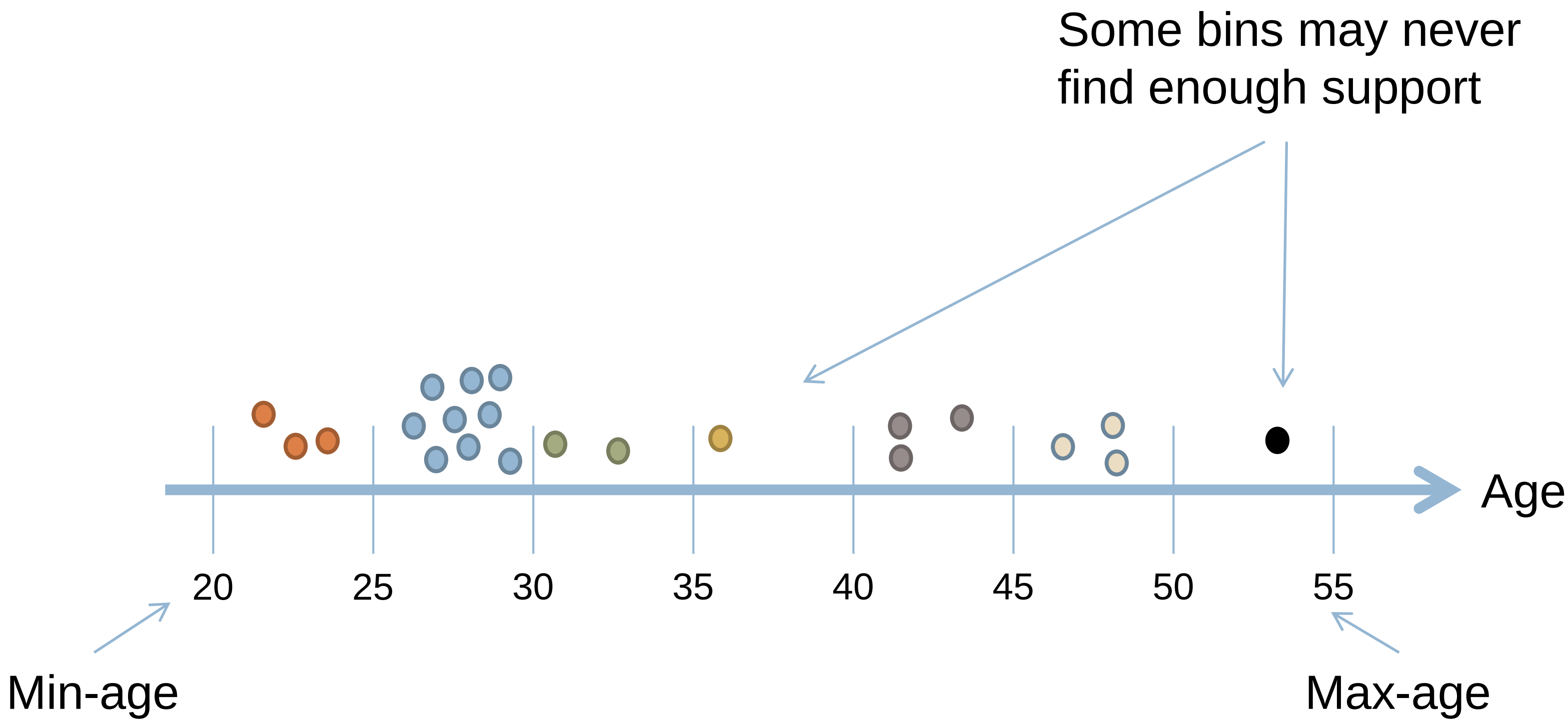
  age(x, "25..35") ^ income(x, "12..30K") → buys(x, "PC") [20%, 75%]

Handling Numerical Attributes

- Want to discretize continuous domain (e.g. *age*)
- Idea 1: Equi-width binning

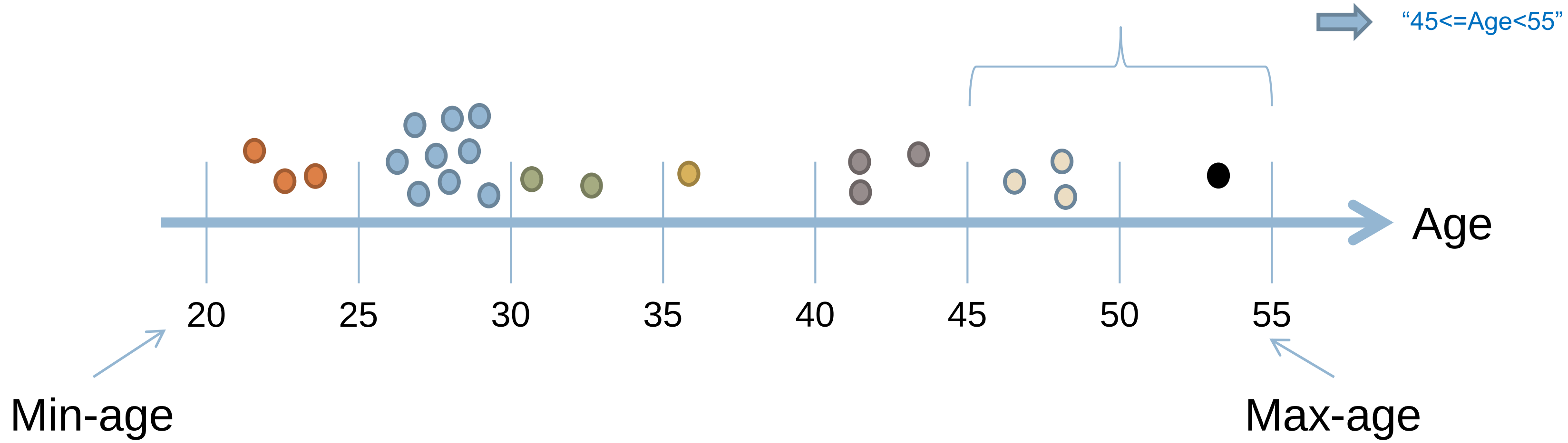


Equi-width binning problems



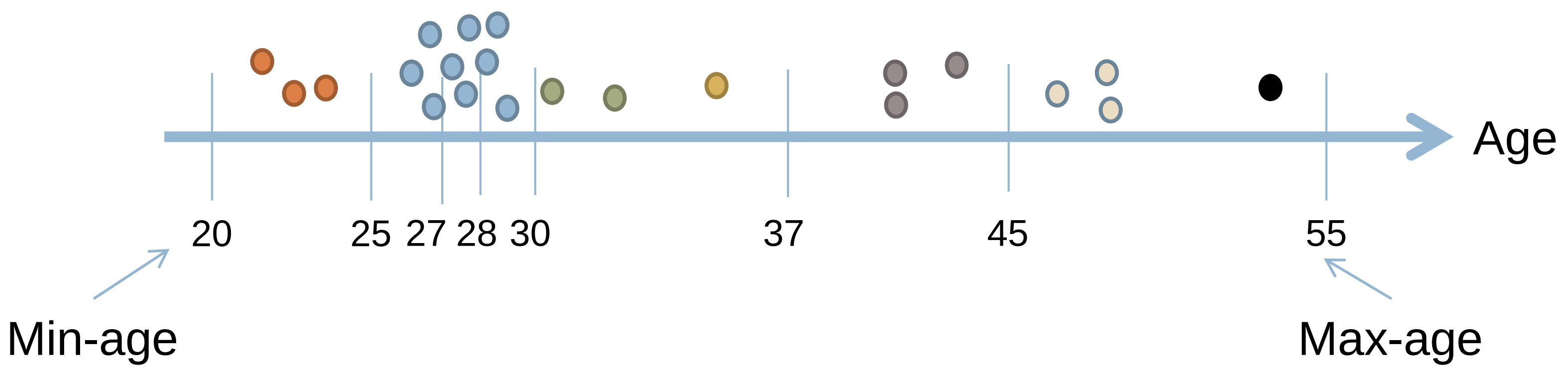
Bin-merging

Merge adjacent intervals when support < min-support



Equi-depth binning

- Sort objects, choose bins so as to equi-divide objects among them
  - Produced bins have (approximately) same freq



Example (python)


```
In [5]: df=pd.DataFrame([['john',21],['nick',22],['martha',23],['taylor',26],['tim',27],['jim',27],
                         ['nick',28],['mike',28],['kostas',28],['don',29],['mihaela',29],['jay',30],
                         ['donald',31],['josh',32],['george',35],['terry',39],['lisa',40],['dina',42],
                         ['pit',46],['nash',47],['scrooge McDuck',47]
                        ],columns=['name','age'])
        print(df)
```

```
              name  age
0             john   21
1             nick   22
2           martha   23
3           taylor   26
4              tim   27
5              jim   27
6             nick   28
7             mike   28
8           kostas   28
9              don   29
10         mihaela   29
11             jay   30
12          donald   31
13            josh   32
14          george   35
15           terry   39
16            lisa   40
17            dina   42
18             pit   46
19            nash   47
20  scrooge McDuck   47
```

Equi-width binning with cut()


```
In [2]: out = pd.cut(df.age,7,labels=['too young','very young','young','fine','kind of old','old','dinosaur'])
        df['equi_width']=out
        print(df)
```

```
              name  age   equi_width
0             john   21    too young
1             nick   22    too young
2           martha   23    too young
3           taylor   26   very young
4              tim   27   very young
5              jim   27   very young
6             nick   28   very young
7             mike   28   very young
8           kostas   28   very young
9              don   29        young
10         mihaela   29        young
11             jay   30        young
12          donald   31        young
13            josh   32        young
14          george   35         fine
15           terry   39  kind of old
16            lisa   40          old
17            dina   42          old
18             pit   46     dinosaur
19            nash   47     dinosaur
20  scrooge McDuck   47     dinosaur
```

Issue: some bins are too sparse

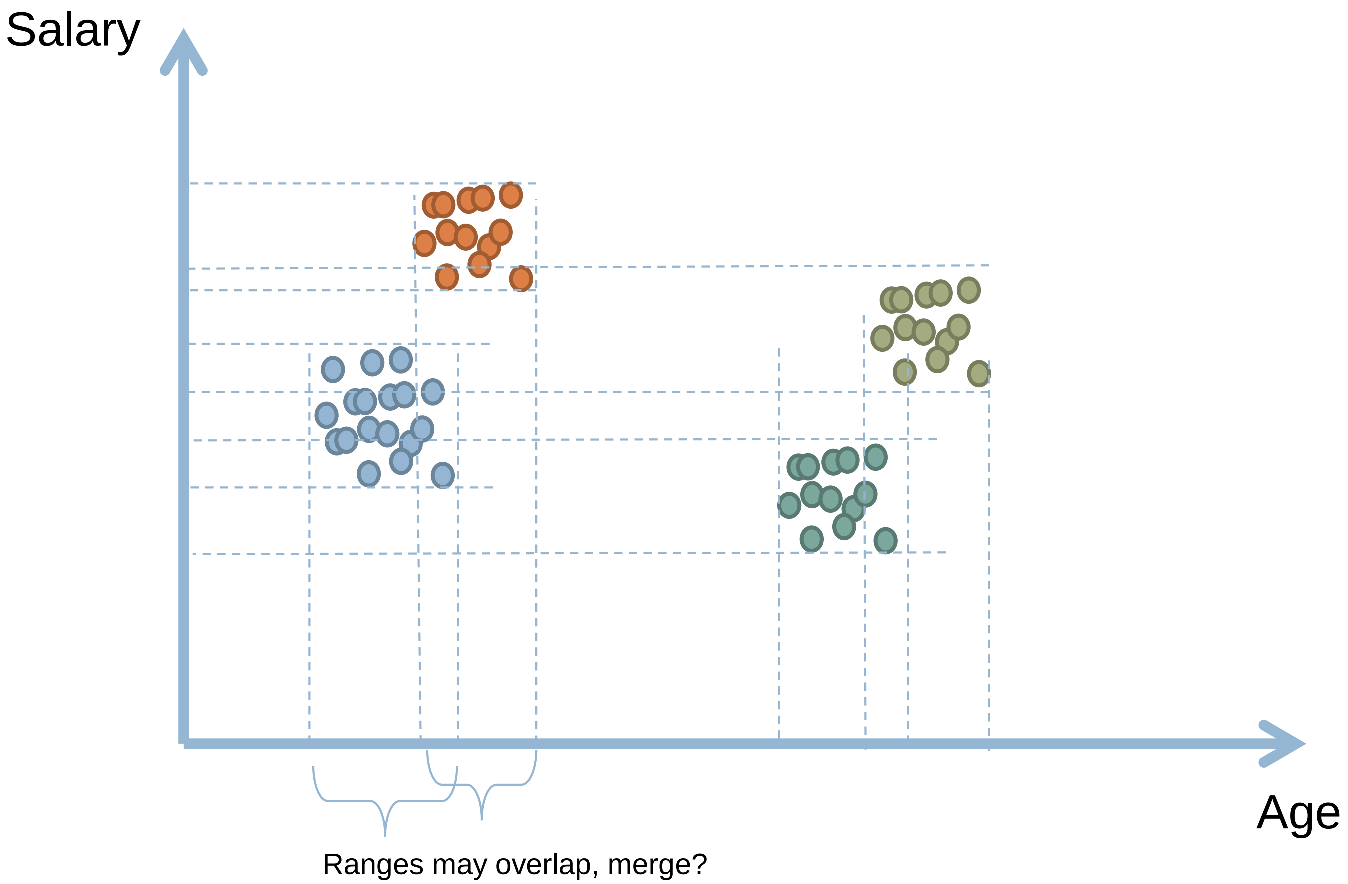

```
In [2]: out = pd.cut(df.age,7,labels=['too young','very young','young','fine','kind of old','old','dinosaur'])
        df['equi_width']=out
        print(df)

                 name  age   equi_width
0                john   21    too young
1                nick   22    too young
2              martha   23    too young
3              taylor   26   very young
4                 tim   27   very young
5                 jim   27   very young
6                nick   28   very young
7                mike   28   very young
8              kostas   28   very young
9                 don   29        young
10            mihaela   29        young
11                jay   30        young
12             donald   31        young
13               josh   32        young
14             george   35         fine
15              terry   39  kind of old
16               lisa   40          old
17               dina   42          old
18                pit   46     dinosaur
19               nash   47     dinosaur
20    scrooge McDuck   47     dinosaur

In [8]: df.groupby('equi_width').size()

Out[8]: equi_width
        too young      3
        very young     6
        young          5
        fine           1
        kind of old    1
        old            2
        dinosaur       3
```

Equi-depth binning with qcut()


```
In [5]: out = pd.qcut(df.age,7,labels=['too young','very young','young','fine','kind of old','old','dinosaur'])
        df['equi_depth']=out
        print(df)
```

```
             name  age   equi_width   equi_depth
0            john   21    too young    too young
1            nick   22    too young    too young
2          martha   23    too young    too young
3          taylor   26   very young   very young
4             tim   27   very young   very young
5             jim   27   very young   very young
6            nick   28   very young        young
7            mike   28   very young        young
8          kostas   28   very young        young
9             don   29        young         fine
10        mihaela   29        young         fine
11            jay   30        young         fine
12         donald   31        young  kind of old
13           josh   32        young  kind of old
14         george   35         fine  kind of old
15          terry   39  kind of old          old
16           lisa   40          old          old
17           dina   42          old          old
18            pit   46     dinosaur     dinosaur
19           nash   47     dinosaur     dinosaur
20  scrooge McDuck   47     dinosaur     dinosaur
```

Discretization with clustering (several options)



Ratio Rules

- Example:

  Customer spends 1:2:5 $ on bread:milk:butter

- May answer questions of the form:
  - A customer who spends $10 on milk and $7 on butter how much is he willing to spend on diapers and beer?

- Ratio Rules derived using **eigenvector analysis**

All is not perfect with A-priori

- Performance considerations
- Usefulness of rules discovered

Tyranny of counting pairs

- Why counting supports of candidates is a problem?
  - The total number of candidates can be huge
  - One transaction may contain many candidates

  - Assume M items
  - How many itemsets of size 2?

$$M!/[(M-2)! * 2!] = M(M-1)/2$$

  - M=10,000 → 49,995,000 combinations

Many optimizations considered

- **Hash-based itemset counting: A *k*-itemset whose corresponding hashing bucket count is below the threshold cannot be frequent**
- **Transaction reduction**: A transaction that does not contain any frequent k-itemset is useless in subsequent scans.
- **Partitioning:** Any itemset that is potentially frequent in DB must be frequent in at least one of the partitions of DB
- **Sampling**: mining on a subset of given data, lower support threshold

Use hashing to expedite generation of C_2

- The PCY algorithm

J. Park, M. Chen, and P. Yu. An effective hash-based algorithm for mining association rules. In *SIGMOD'95*

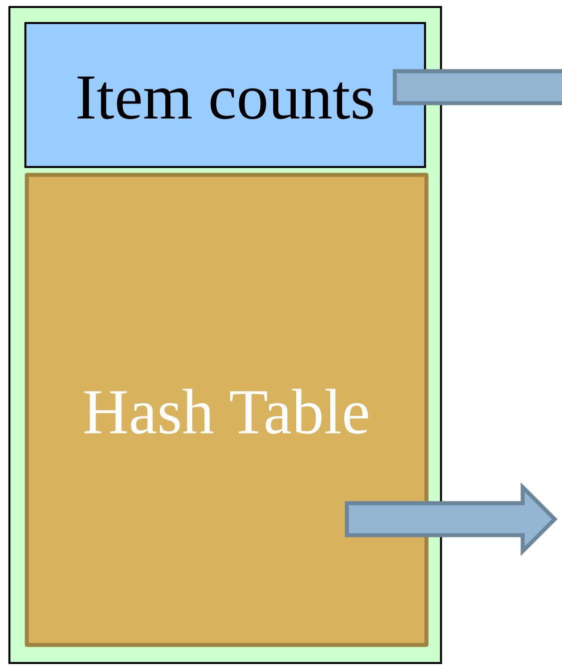
Key issue

- Counting pairs (second phase of a-priori) is too slow
  - Number of possible pairs is (often) much larger than main memory
- Wal-Mart sells 140,000 items and can store billions of baskets.
  - With 4-byte counters, need 36GB of RAM to store all pair counts in a triangular matrix
  - May also store only existing pairs in a list using 3x more space per pair

PCY Algorithm

- Hash-based improvement to A-Priori.
- During Pass 1 of A-priori, most memory is idle.
  - We only count frequent items
  - One counter (e.g. 4 bytes) per item suffices
    - For the Wal-Mart example ~0.6MB is enough
- Use extra memory for a hash table [0...B-1]
  - Each hash bucket stores a counter for that bin
  - Need B*4bytes

Hash Table Memory Usage: All-you-can-eat



Counters for itemsets with exactly 1 item used during fist pass

Will be used for counting groups of {x,y} pairs appearing in transactions

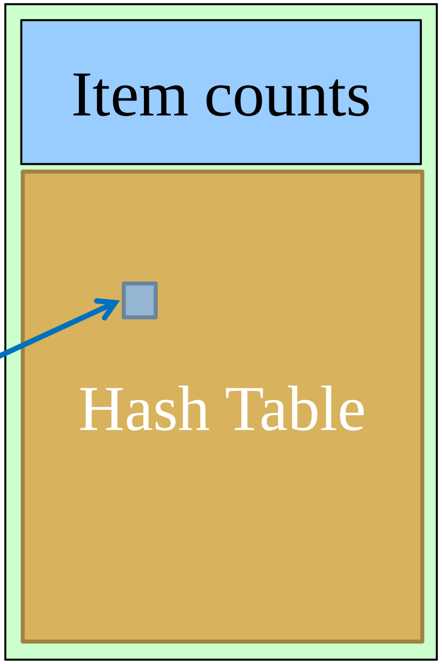
Hashing pairs

- Assume hash function h(x,y) that maps a pair of items x,y to a bucket in range 0..B-1
  - E.g. h(beer,diaper)=127

- While counting frequent items, upon seeing a transaction with x_1,… x_k items list all pairs x_i, x_j from this transaction
  - For each pair increase counter of corresponding bucket $h(x_i, x_j)$ by one

Notice: collisions

- Number of possible pairs is much larger than size of hash table
  - Collisions are inevitable!
- E.g. is may be that h(beep,diapers) = h(PC,Monitor) =
- Thus, a bucket k counts all pairs x,y for which h(x,y)=k



Observations About Buckets

- If a bucket contains a frequent pair, then the bucket is surely frequent.
  - We cannot use the hash table to eliminate any member of this bucket.
- Even without any frequent pair, a bucket can be frequent.
  - Again, nothing in the bucket can be eliminated.
- But in the best case, the count for a bucket is less than the support s.
  - Now, all pairs that hash to this bucket can be eliminated as candidates, even if the pair consists of two frequent items.

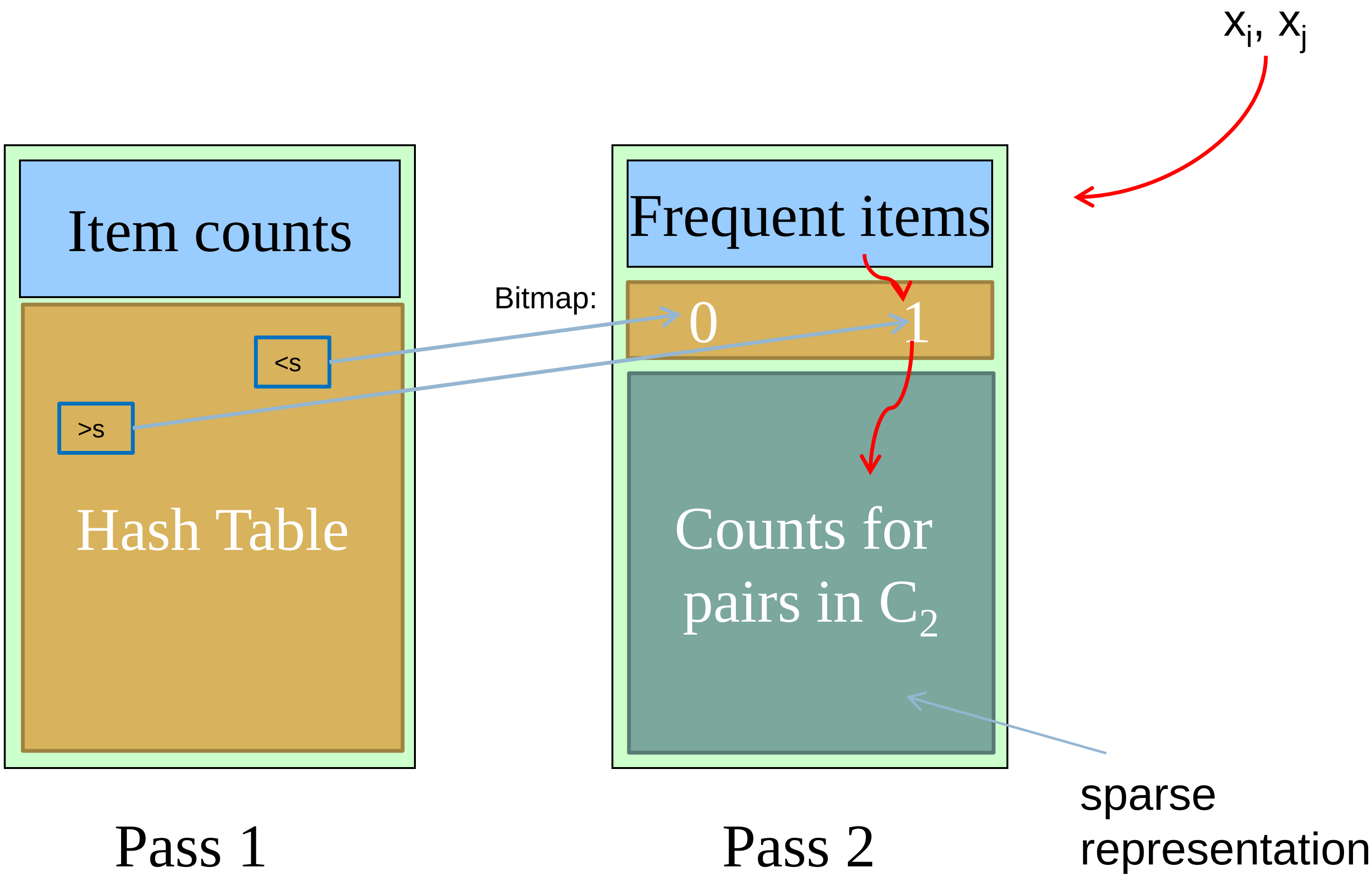
PCY Algorithm --- Pass 1


```
FOR (each basket) {
    FOR (each item)
      add 1 to item's count;
    FOR (each pair of items) {
      hash the pair to a bucket;
      add 1 to the count for that
    bucket
    }
}
```

PCY Algorithm: Between Passes

- Replace the buckets by a bit-vector:
  - 1 means the bucket count exceeds the support s (frequent bucket); 0 means it did not.
- Integers are replaced by bits, so the bit-vector requires little second-pass space.
- Also, decide which C_1 items are frequent and list them (create L_1) for the second pass.

Pass 2



x_i, x_j
Item counts
Frequent items
Bitmap:
0
1
<s
>s
Hash Table
Counts for pairs in C_2
Pass 1
Pass 2
sparse representation

PCY Algorithm --- Pass 2

- Count all pairs {*i*,*j* } that meet the conditions:
  1. Both *i* and *j* are frequent items.
  2. The pair {*i*,*j* }, hashes to a bucket number whose bit in the bit vector is 1.

- Notice all these conditions are necessary for the pair to have a chance of being frequent.

Strong Rules $\stackrel{?}{=}$ Interesting Rules

- Example 1: (Aggarwal & Yu, PODS98)
  - Among 5000 students
    - 3000 play basketball
    - 3750 eat cereal
    - 2000 both play basket ball and eat cereal
- *Compare the following two rules*
  - *play basketball* $\Rightarrow$ *eat cereal* [40%, 66.7]
  - *play basketball* $\Rightarrow$ *not eat cereal* [20%, 33.3%]

2000/5000

2000/3000

	basketball	not basketball	sum(row)
cereal	2000	1750	3750
not cereal	1000	250	1250
sum(col.)	3000	2000	5000

Strong Rules Are Not Necessarily Interesting

- *play basketball* $\Rightarrow$ *eat cereal* [40%, 66.7%] is misleading because the overall percentage of students eating cereal is 75% which is higher than 66.7%.
- *play basketball* $\Rightarrow$ *not eat cereal* [20%, 33.3%] is far more accurate, although with lower support and confidence

	basketball	not basketball	sum(row)
cereal	2000	1750	3750
not cereal	1000	250	1250
sum(col.)	3000	2000	5000

Criticism to Support and Confidence (Cont.)

- Example 2:
  - X and Y: positively correlated,
  - X and Z, negatively related
  - support and confidence of X→Z dominates
- We need a measure of dependent or correlated events

X	1	1	1	1	0	0	0	0
Y	1	1	0	0	0	0	0	0
Z	0	1	1	1	1	1	1	1

Rule	Support	Confidence
X=>Y	25%	50%
X=>Z	37,50%	75%

Lift of an Association Rule

- Lift(X→Y) = P(X and Y)/(P(X)*P(Y))
  - P(X and Y) = support observed in the dataset
  - P(X)*P(Y) = expected support if X and Y were independent
  - Lift(X→Y)>1 suggests that X&Y appear together more often that expected. Thus, the occurrence of X has a positive effect on the occurrence of Y

X	1	1	1	1	0	0	0	0
Y	1	1	0	0	0	0	0	0
Z	0	1	1	1	1	1	1	1

Itemset	Support	Lift
{X,Y}	25%	2.00
{X,Z}	37.5%	0.86
{Y,Z}	12.5%	0.57

- In some cases rare items may produce rules with very high values of lift

Back to the student's survey

- *play basketball* $\Rightarrow$ *eat cereal* [40%, 66.7%]
  - Lift = (2000/5000)/((3000/5000)*(3750/5000)) = 0.89 < 1

- *play basketball* $\Rightarrow$ *not eat cereal* [20%, 33.3%]
  - Lift = (1000/5000)/((3000/5000)*(1250/5000)) = 1.33 > 1

	basketball	not basketball	sum(row)
cereal	2000	1750	3750
not cereal	1000	250	1250
sum(col.)	3000	2000	5000

Recap (lift)

- Lift evaluates the mined rule against the expected response assuming independence
  - Lift(X→Y) = sup(X,Y)/(sup(X)*sup(Y))

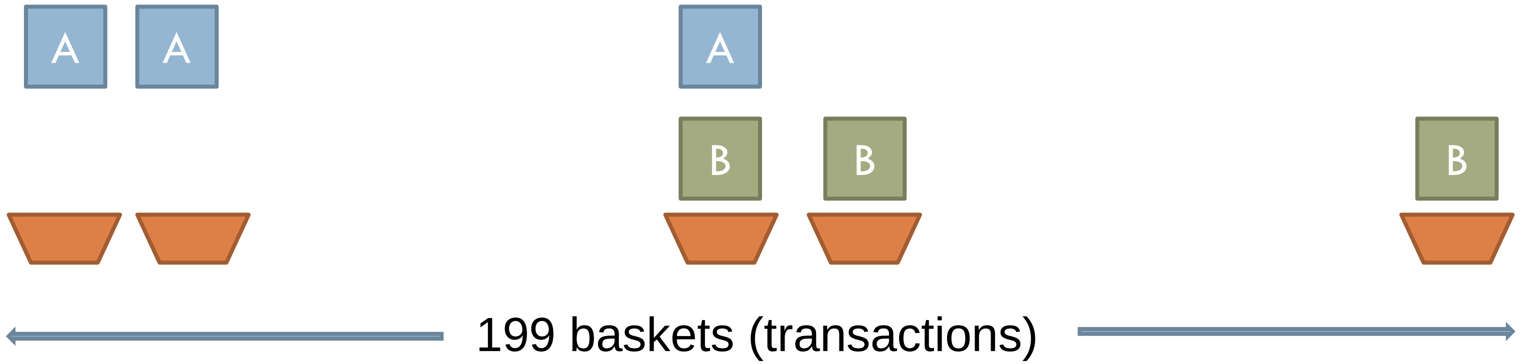
- Equiv. Lift = Confidence(rule)/expConfidence(Rule)
  - Confidence(X→Y)=P(X,Y)/P(X)=sup(X,Y)/sup(X)
  - expConfidence(X→Y)=P(X)(P(Y)/P(X)= P(Y)= sup(Y)
  - Thus, Lift(X→Y) = sup(X,Y)/(sup(X)*sup(Y))

Criticism on lift: effect of null transactions

- Assume itemset {A,B}
- A null transaction is a transaction that does not contain any of the itemsets being examined.
  - E.g T={D,F,G} is a null transaction for this itemset

Example

- Assume that store sold 100 packages of A and 100 packages of B
  - Only one of the above transactions contains both A,B
  - There are no null transactions for {A,B} in this example

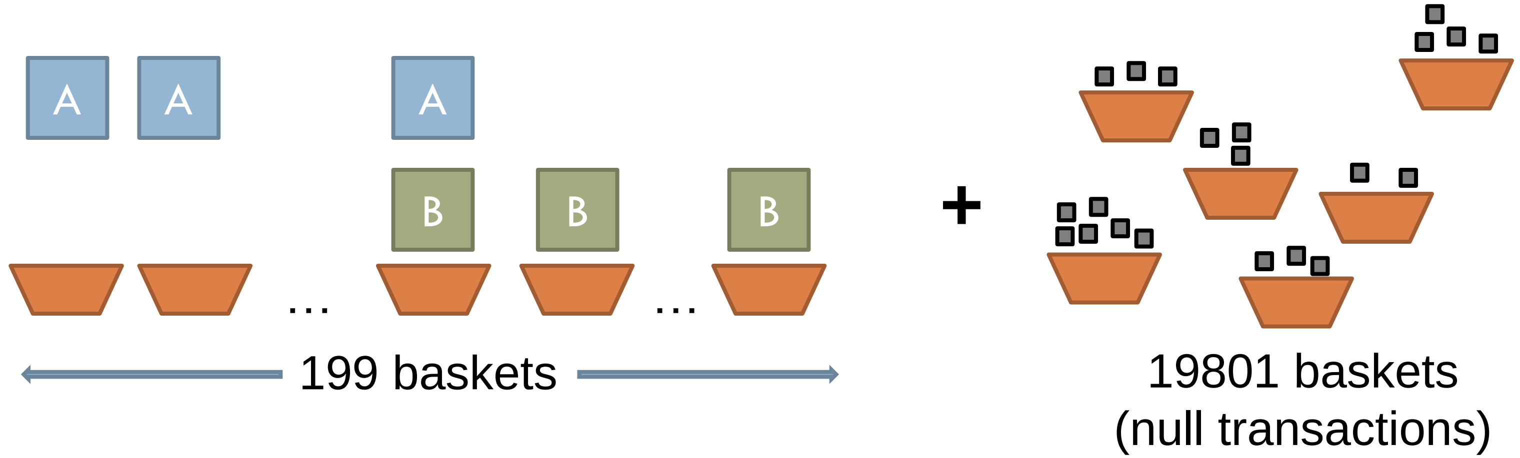


Example

- Assume that store sold 100 packages of A and 100 packages of B
  - Only one of the above transactions contains both A,B
- Thus, $P(A) = P(B) = 100/199$
- $P(A \text{ and } B) = 1/199$
- $\text{Lift} = 1/199 \,/ (100/199 * 100/199) \approx 0.02$
- Conclusion: A and B are *negatively* correlated

Effect of null transactions

- Now assume arrival of 19801 more transactions that do not contain A nor B
  - Total number of transactions is n=199+19801=20000
  - Thus, P(A) = P(B) =100/20000
  - P(A and B) = 1/20000
  - Lift = 1/20000/(100/20000 * 100/20000) =2
- Conclusion: A and B are *positively* correlated
  - Which is true. Neither A nor B appear in the 19801 null transactions we added!



Why is that?

- Lift = P(A and B) /(P(A)*P(B)) =

  $= |A \text{ and } B|/n \, / \, (|A|/n * |B|/n) =$

  $= n * |A \text{ and } B|/(|A|*|B|)$

- When more null transactions are added
  - n in increased
  - |A and B|, |A| and |B| stay constant
  - As a result, lift increases by adding more null transactions
- Thus, lift is not null invariant

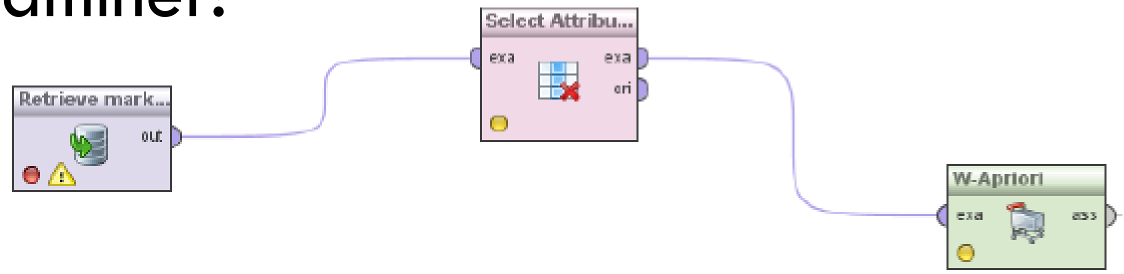
A solution: use cosine!

- Define cosine(A,B) = P(A and B)/sqrt(P(A)*P(B))
- Cosine takes values between 0 and 1
- Because of the sqrt(), cosine does not depend on n, thus, it is null invariant
- In this example cosine(A,B)= 0.01 in both examples

Many different implementations

- R: rules<-apriori(trans,parameter=list(supp=.02, conf=.5, target="rules"))

- Rapidminer:



Association rules - Conclusions

- An intuitive tool to find patterns
  - easy to understand its output
  - number of rules is a concern
  - fine-tuned algorithms exist

Technical Skills

- Many analysts rely on using simple low-level tools
  - Many tasks can be executed using Unix shell commands
  - Text manipulation languages (e.g. awk, perl) help you perform complex analytical tasks in a breeze

Text file with 10M customer sales data

SaleId	Date	Store_Location	Customer	Product_sold
1	1/5/2012	Athens	John	Sony Vaio Laptop
2	1/5/2012	Thessaloniki	Jim	iPad 2
3	2/5/2012	Larissa	John	LG TV
4	2/5/2012	Athens	Helen	Dell PC
5	2/5/2012	Athens	Mary	HP Printer
6	…	…	…	…

- What do the following commands compute?

$ cut –f4 sales.txt | sort | uniq –c | sort –nr | head -10

Same task using perl (top.pl script)


```
#!/usr/bin/perl -w
my $n = shift;
my $k = shift;
open (in_ , "<$ARGV[0]") or die;
my %counts;
while (<in_>) {
    chomp;
    my @val=(split(/\t/));
    $counts{$val[$n]}++;
}

foreach (sort {$counts{$a} <=> $counts{$b}} keys(%counts)) {
    print "$_\t$counts{$_}\n";
    last unless --$k;
}
```

Perform aggregation via hashing

In SQL


```
create database test;
use test;
create table sales (salesid int, sdate varchar(12),city
   varchar(50),cust varchar(50), prod varchar(50));
load data local infile "sales.txt" into table sales;
select cust,count(*) as Count from sales group by cust
   order by Count desc limit 10;
```

Comparisons

- Created text files with 10M/100M random sales
  - 400 MB / 4GB approximately
  - Notice something strange?

	Unix Shell	SQL Database	Perl
10M	1m 29s	Load table 1m 26s	26s
		Query table 11s	
100M	15m 19s	Load table 13m 6s	4m 27s
		Query table 2m 8s	