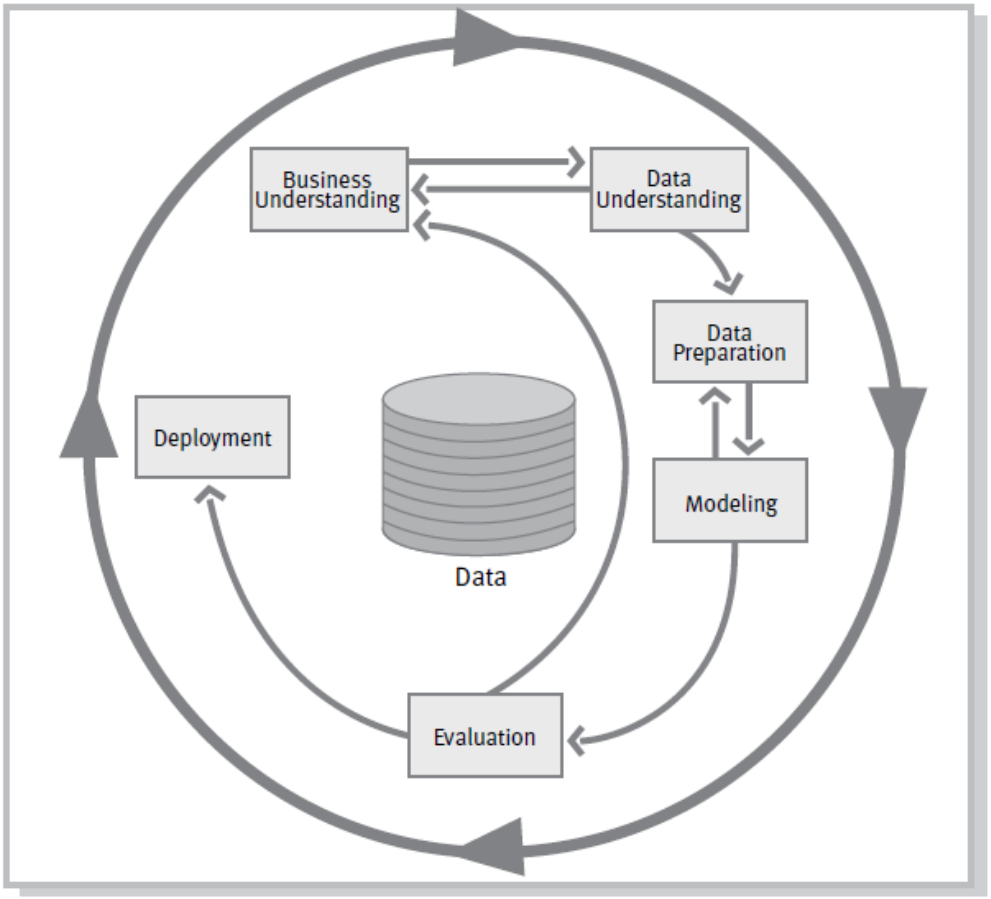
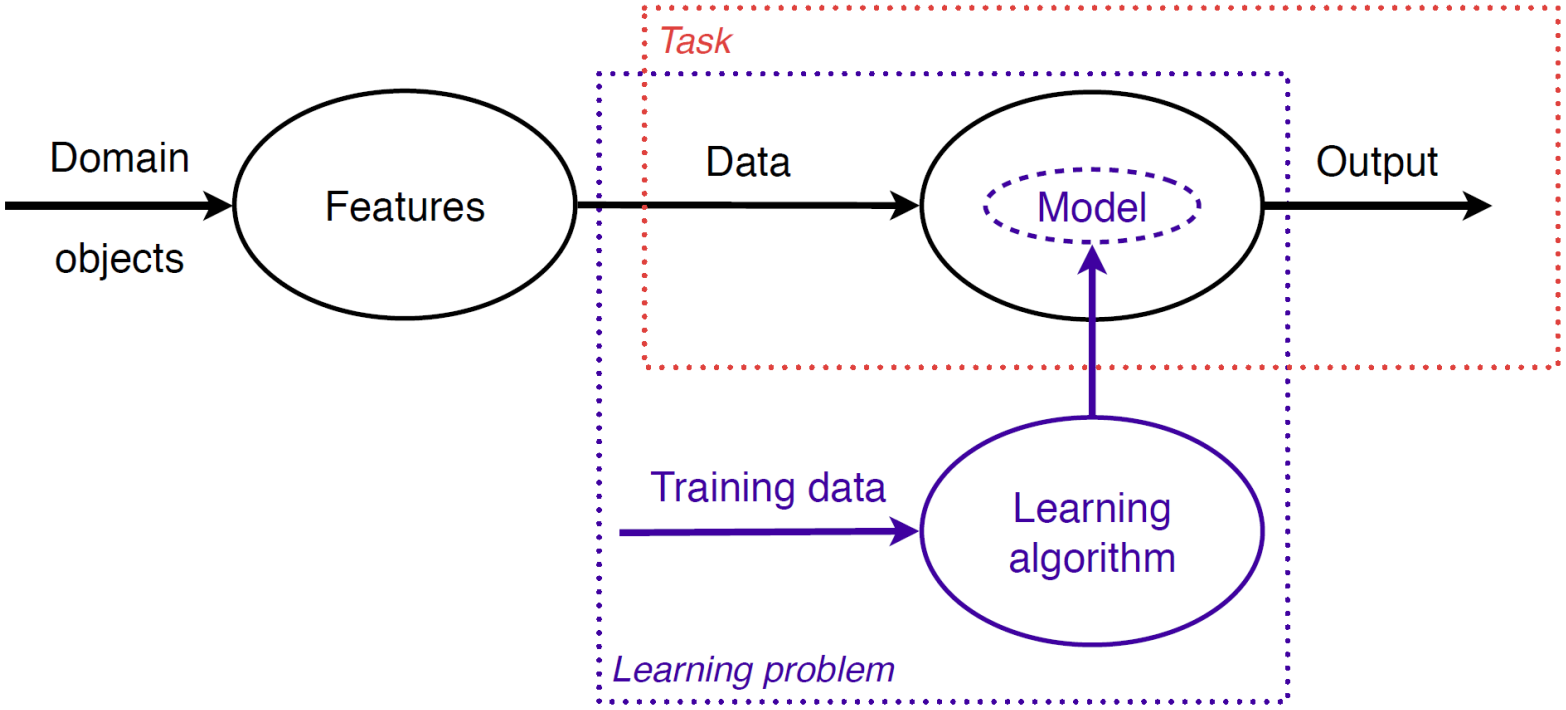
Lesson B-1

Introduction to Data Mining Modeling and Algorithms

The life cycle of a data mining project




Task
Domain
objects
Features
Data
Model
Output
Training data
Learning
algorithm
Learning problem

Learning Styles (1)

- Supervised learning
  - The training data (observations, measurements, etc.) are accompanied by labels indicating the class of the observations and new data is classified based on the training set
    - Classification, regression, neural networks
- Unsupervised learning
  - The class labels of training data is unknown and the process of grouping a set of objects into classes of similar objects
    - Clustering, association rules

Learning Styles (2)

- Semi-supervised learning
  - Some algorithms can deal with partially labeled training data, usually a lot of unlabeled data and a little bit of labeled data.
    - RNN, DBN, RMS
- Reinforced learning
  - The learning system, called an agent in this context, can observe the environment, select and perform actions, and get rewards in return or penalties in the form of negative rewards
    - DeepMind's AlphaGo

Types of Algorithms (1)

- Classification
  - algorithms can be used to build a model that predicts the outcome class for a given dataset
    - Decision tree, kNN, Bayesian, ensemble, etc.
- Regression
  - a statistical method for examining the relationship between two or more variables to predict numeric values
    - Regression (simple linear, nonlinear, polynomial), regularization, support vector machine, etc.

Types of Algorithms (2)

- Neural networks
  - an artificial model based on the human brain and it learns task without being told any specific rules
    - Perceptron, back-propagation, MLP etc.
- Deep learning
  - Special type of neural networks and deep-learning architectures such as deep neural networks, deep belief networks, deep reinforcement learning, recurrent neural networks and convolutional neural networks
    - RNN, CNN, DBN, LVQ, RBM, etc.

Types of Algorithms (3)

- Ensemble
  - meta-algorithms that combine several machine learning methods into a single predictive model to increase the overall performance
    - Random forest, bagging, AdaBoost, stacking etc.
- Clustering
  - Clustering is the practice of assigning labels to unlabeled data using the patterns that exist in it
    - Partitioning (k-means, k-medians, k-modes), hierarchical (agglomerative, divisive), density-based (DBSCAN, OPTICS)

Types of Algorithms (4)

- Association rule
  - uncover how items are associated with each other
    - Frequent pattern mining, Apriori algorithm
- Anomaly detection
  - anomaly detection is used to find rare occurrences or suspicious events in your data
    - Isolation forest, PCA-based anomaly detection, IQR-based, standard deviation

Lesson B-2

Frequent Pattern Mining

-frequent patterns, Apriori algorithms, mining association rules, and correlation rules

Frequent Patterns

- Frequent pattern mining searches for recurring relationships in a given data set
- Frequent pattern
  - patterns (e.g., itemset, sequences, or structures) that appear frequently in a dataset.
    - milk and bread, that appear frequently together in a transaction
    - buying first a smartphone, then a phone case, and then a memory card, if it occurs frequently in a shopping history database

Frequent Pattern Analysis

- Motivation: Finding inherent regularities in data
  - What products were often purchased together?— Beer and diapers?!
  - What are the subsequent purchases after buying a PC?
  - What kinds of DNA are sensitive to this new drug?
  - Can we automatically classify web documents?
- Applications
  - Basket data analysis, cross-marketing, catalog design, sale campaign analysis, Web log (click stream) analysis, recommender systems, DNA sequence analysis, etc.

Basic Concepts of Frequent Patterns

- Itemset
- k-itemset
- the occurrence frequency of an itemset (frequency, support count or count of the itemset)
- frequent itemset
- closed frequent itemset
- maximal frequent itemset
- (relative) support
- (absolute) support
- minimum support threshold (count)
- association rules
- confidence
- minimum confidence threshold (count)
- strong rules

Item, Itemset, k-itemset, a set of itemset, a transaction database

TID	Items bought
1	A, B, C, E
2	A, C, D, E
3	B, C, E
4	A, C, D, E
5	C, D, E
6	A, D, E

The occurrence frequency of an itemset

TID	Items bought
1	A, B, C, E
2	A, C, D, E
3	B, C, E
4	A, C, D, E
5	C, D, E
6	A, D, E

{A}
{B}
{C}
{D}
{E}

{A,B}
{A,C}
{A,D}
{A,E}
{B,C}
{B,D}
{B,E}
{C,D}
{C,E}
{D,E}

{A,B,C}
{A,B,D}
{A,B,E}
{A,C,D}
{A,C,E}
{A,D,E}
{B,C,D}
{B,C,E}
{C,D,E}

{A,B,C,D}
{A,B,C,E}
{B,C,D,E}

Minimum support count and frequent itemset

minimum support count = 3

TID	Items bought
1	A, B, C, E
2	A, C, D, E
3	B, C, E
4	A, C, D, E
5	C, D, E
6	A, D, E

{A} = 4
{B} = 2
{C} = 5
{D} = 4
{E} = 6

{A,B} = 1
{A,C} = 3
{A,D} = 3
{A,E} = 4
{B,C} = 2
{B,D} = 0
{B,E} = 2
{C,D} = 3
{C,E} = 5
{D,E} = 4

{A,B,C} = 1
{A,B,D} = 0
{A,B,E} = 1
{A,C,D} = 2
{A,C,E} = 3
{A,D,E} = 3
{B,C,D} = 0
{B,C,E} = 2
{C,D,E} = 3

{A,B,C,D} = 0
{A,B,C,E} = 1
{B,C,D,E} = 0

Closed frequent itemset and maximal frequent itemset (max itemset)

Frequent itemset X ∈ D is closed if it has no superset with the same frequency.
Frequent itemset X ∈ D is maximal if it does not have any frequent supersets.

TID	Items bought
1	A, B, C, E
2	A, C, D, E
3	B, C, E
4	A, C, D, E
5	C, D, E
6	A, D, E

{A} = 4
~~{B} = 2~~
{C} = 5
{D} = 4
{E} = 6

~~{A,B} = 1~~
{A,C} = 3
{A,D} = 3
{A,E} = 4
~~{B,C} = 2~~
~~{B,D} = 0~~
~~{B,E} = 2~~
{C,D} = 3
{C,E} = 5
{D,E} = 4

~~{A,B,C} = 1~~
~~{A,B,D} = 0~~
~~{A,B,E} = 1~~
~~{A,C,D} = 2~~
{A,C,E} = 3
{A,D,E} = 3
~~{B,C,D} = 0~~
~~{B,C,E} = 2~~
{C,D,E} = 3

~~{A,B,C,D} = 0~~
~~{A,B,C,E} = 1~~
~~{B,C,D,E} = 0~~

How to Generate Frequent Itemset?

- Let $I = \{I_1, I_2, \ldots, I_m\}$ be a set of items
- Let *D*, the task-relevant data, be a set of database transactions where each transaction *T* is a set of items such that $T \subseteq I$
- Each transaction is associated with an identifier, called *TID*.
- Let *A* be a set of items
- A transaction *T* is said to contain *A* if and only if $A \subseteq T$

How to Generate Frequent Itemset?

- Suppose the items in L_{k-1} are listed in an order
- **The join step**:
  - To find L_k, a set of candidate k-itemsets, C_k, is generated by joining L_{k-1} with itself.
    - Let l_1 and l_2 be itemsets in L_{k-1}.
    - The resulting itemset formed by joining l_1 and l_2 is $l_1[1], l_1[2], \ldots, l_1[k-2], l_1[k-1], l_2[k-1]$
- **The prune step**:
  - Scan data set D and compare candidate support count of C_k with minimum support count.
  - Remove candidate itemsets that whose support count is less than minimum support count, resulting in L_k.

Apriori Algorithm

- Initially, scan DB once to get frequent 1-itemset
- Generate length *(k+1)* candidate itemsets by joining length *k* frequent itemsets
- Prune length *(k+1)* candidate itemsets **with Apriori property**
  - **Apriori property: All nonempty subsets of a frequent itemset must also be frequent**
- Test the candidates against DB
- Terminate when no frequent or candidate set can be generated

Algorithm: Apriori. Find frequent itemsets using an iterative level-wise approach based on candidate generation.

Input:

- D, a database of transactions;
- *min_sup*, the minimum support count threshold.

Output: L, frequent itemsets in D.

Method:

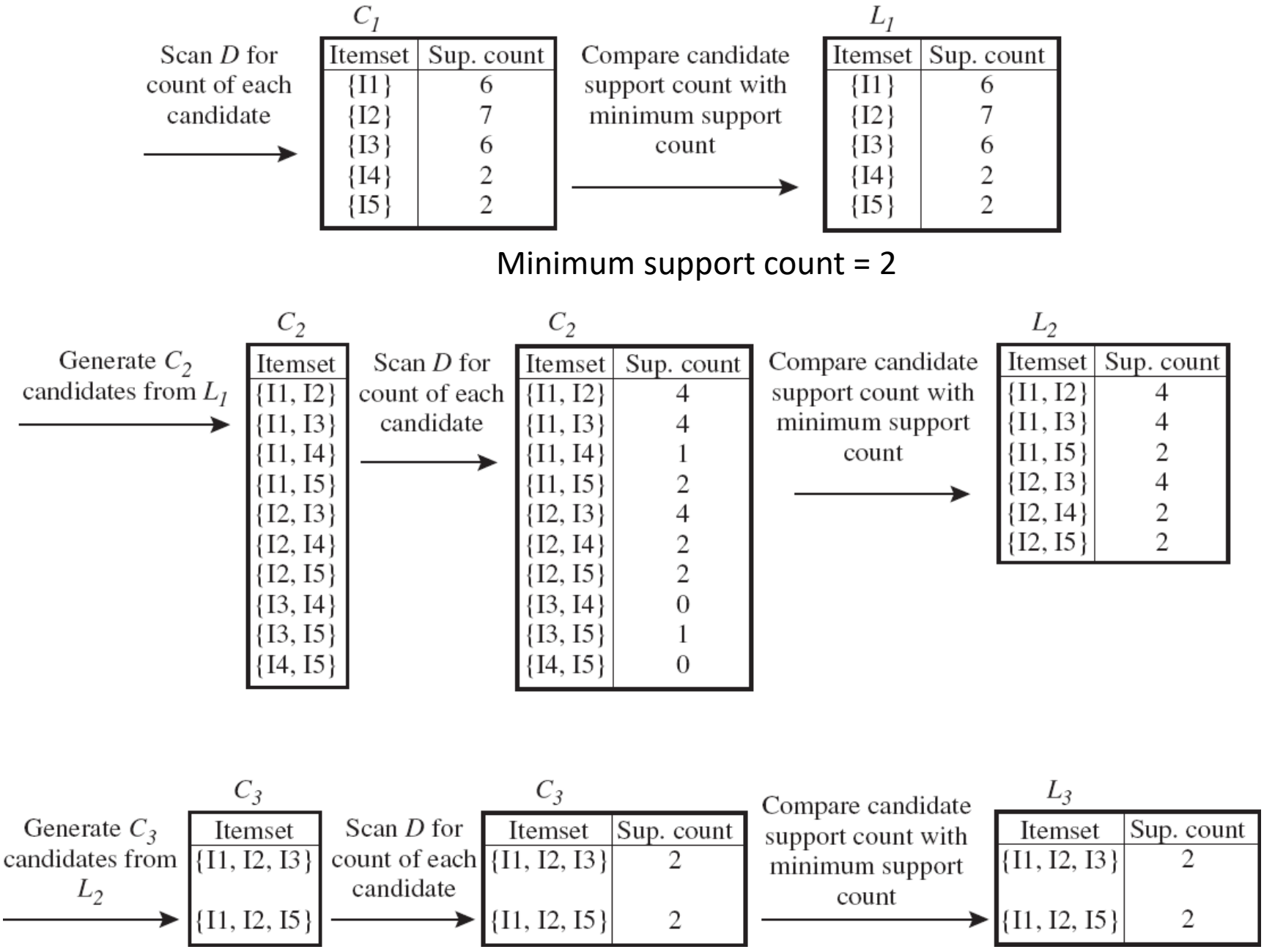
```
(1)   L1 = find_frequent_1-itemsets(D);
(2)   for (k = 2; Lk−1 ≠ φ; k++) {
(3)       Ck = apriori_gen(Lk−1);
(4)       for each transaction t ∈ D { // scan D for counts
(5)           Ct = subset(Ck, t); // get the subsets of t that are candidates
(6)           for each candidate c ∈ Ct
(7)               c.count++;
(8)       }
(9)       Lk = {c ∈ Ck|c.count ≥ min_sup}
(10)  }
(11)  return L = ∪k Lk;

procedure apriori_gen(Lk−1:frequent (k − 1)-itemsets)
(1)   for each itemset l1 ∈ Lk−1
(2)       for each itemset l2 ∈ Lk−1
(3)           if (l1[1] = l2[1]) ∧ (l1[2] = l2[2])
                  ∧... ∧ (l1[k − 2] = l2[k − 2]) ∧ (l1[k − 1] < l2[k − 1]) then {
(4)               c = l1 ⋈ l2; // join step: generate candidates
(5)               if has_infrequent_subset(c, Lk−1) then
(6)                   delete c; // prune step: remove unfruitful candidate
(7)               else add c to Ck;
(8)           }
(9)   return Ck;

procedure has_infrequent_subset(c: candidate k-itemset;
          Lk−1: frequent (k − 1)-itemsets); // use prior knowledge
(1)   for each (k − 1)-subset s of c
(2)       if s ∉ Lk−1 then
(3)           return TRUE;
(4)   return FALSE;
```

Apriori algorithm for discovering frequent itemsets for mining Boolean association rules.

Transactional Database

TID	*List of item_IDs*
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

Transactional data for an AllElectronics branch.



Generation of candidate itemsets and frequent itemsets, where the minimum support count is 2.

Rules

- Rule-based expert system
  - *If-then* rules
  - *X => Y*

Association Rules

- In data mining, association rules are "if-then" statements, that help to show the probability of relationships between data items or attributes within large data sets.
  - Discover relations between attributes or data items within large dataset.

Association Rules

- The rule *X => Y is called a strong association rule*
  - *when it satisfies a prespecified* ***minimum support threshold and*** *a prespecified* ***minimum confidence threshold.***

Association Rules

- The association rule X => Y holds in the transaction set D with support s,
  - where s is the percentage of transactions in D that contain X U Y (i.e., the union of sets X and Y say, or, both X and Y).

Support

- The rule $A \Rightarrow B$ holds in the transaction set D with *support* ***s***
  - *support*, ***s***, probability that a transaction contains A and B
  - *support* $(A \Rightarrow B)$ = *support* $(A \cup B)$ = $P(A \cup B)$, *range:[0,1]*

Confidence

- The rule $A \Rightarrow B$ has **confidence *c*** in the transaction set *D*,
  - where *c* is the percentage of transactions in *D* containing *A* that also contain *Y*.
  - *confidence*, ***c***, conditional probability that a transaction having *A* also contains *Y*
  - *confidence* $(A \Rightarrow B) = P(B \mid A)$, *range: [0,1]*
    - $confidence\ (A \Rightarrow B) = P(B|A) = P(A \cup B) / P(A)$

      $= support\ (A \cup B) / support(A)$

      $= support\_count(A \cup B) / support\_count(A)$

Mining Association Rules

- The ***association rule mining*** can be viewed as a two-step process
  - **Find all frequent itemsets:**
    - By definition, each of these itemsets will occur at least as frequently as a predetermined minimum support count, *min_sup*.
  - **Generate strong association rules from the frequent itemsets:**
    - By definition, these rules must satisfy minimum support and minimum confidence.

Association Rules

- Generating Association Rules from Frequent Itemsets
  - *for each frequent itemset l, generate all nonempty subset of l*
  - *For every nonempty subset s of l,*

    Output the rule "s ⇒ (l – s)" If support_count(l) / support_count(s) ≥ min_confidence, where min_confidence is the minimum confidence threshold

- Rules that satisfy both a minimum support threshold and a minimum confidence threshold are called strong

Generating Association Rules from Frequent Itemsets

- Suppose the data contain the frequent itemset *l* = {*I1, I2, I5*}. What are the association rules that can be generated from *l*? If the minimum confidence threshold is 70%, then which rules are strong?

Frequent Patterns -- Apriori Agorithm

```
dataset = [['Milk', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
           ['Dill', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
           ['Milk', 'Apple', 'Kidney Beans', 'Eggs'],
           ['Milk', 'Unicorn', 'Corn', 'Kidney Beans', 'Yogurt'],
           ['Corn', 'Onion', 'Onion', 'Kidney Beans', 'Ice cream', 'Eggs']]
```

```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
te = TransactionEncoder()
te_ary = te.fit(dataset).transform(dataset)
te_ary
```

```
array([[False, False, False,  True, False,  True,  True,  True,  True,
         False,  True],
       [False, False,  True,  True, False,  True, False,  True,  True,
         False,  True],
       [ True, False, False,  True, False,  True,  True, False, False,
         False, False],
       [False,  True, False, False, False,  True,  True, False, False,
          True,  True],
       [False,  True, False,  True,  True,  True, False, False,  True,
         False, False]])
```

```
te.columns_
```

```
['Apple',
 'Corn',
 'Dill',
 'Eggs',
 'Ice cream',
 'Kidney Beans',
 'Milk',
 'Nutmeg',
 'Onion',
 'Unicorn',
 'Yogurt']
```

```
te_ary.astype("int")
```

```
array([[0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1],
       [0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1],
       [1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0],
       [0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1],
       [0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0]])
```

```
1 te.columns_
```

```
['Apple',
 'Corn',
 'Dill',
 'Eggs',
 'Ice cream',
 'Kidney Beans',
 'Milk',
 'Nutmeg',
 'Onion',
 'Unicorn',
 'Yogurt']
```

```
1 df = pd.DataFrame(te_ary, columns=te.columns_)
2 df
```

	Apple	Corn	Dill	Eggs	Ice cream	Kidney Beans	Milk	Nutmeg	Onion	Unicorn	Yogurt
0	False	False	False	True	False	True	True	True	True	False	True
1	False	False	True	True	False	True	False	True	True	False	True
2	True	False	False	True	False	True	True	False	False	False	False
3	False	True	False	False	False	True	True	False	False	True	True
4	False	True	False	True	True	True	False	False	True	False	False

```
1 first4 = te_ary[:4]
2 te.inverse_transform(first4)
```

```
[['Eggs', 'Kidney Beans', 'Milk', 'Nutmeg', 'Onion', 'Yogurt'],
 ['Dill', 'Eggs', 'Kidney Beans', 'Nutmeg', 'Onion', 'Yogurt'],
 ['Apple', 'Eggs', 'Kidney Beans', 'Milk'],
 ['Corn', 'Kidney Beans', 'Milk', 'Unicorn', 'Yogurt']]
```

```
from mlxtend.frequent_patterns import apriori

apriori(df, min_support=0.6)
```

]:

	support	itemsets
0	0.8	(3)
1	1.0	(5)
2	0.6	(6)
3	0.6	(8)
4	0.6	(10)
5	0.8	(3, 5)
6	0.6	(8, 3)
7	0.6	(5, 6)
8	0.6	(8, 5)
9	0.6	(10, 5)
10	0.6	(8, 3, 5)

```
apriori(df, min_support=0.6, use_colnames=True)
```

]:

	support	itemsets
0	0.8	(Eggs)
1	1.0	(Kidney Beans)
2	0.6	(Milk)
3	0.6	(Onion)
4	0.6	(Yogurt)
5	0.8	(Eggs, Kidney Beans)
6	0.6	(Eggs, Onion)
7	0.6	(Milk, Kidney Beans)
8	0.6	(Onion, Kidney Beans)
9	0.6	(Yogurt, Kidney Beans)
10	0.6	(Eggs, Onion, Kidney Beans)

```
frequent_itemsets = apriori(df, min_support=0.6, use_colnames=True)
frequent_itemsets['length'] = frequent_itemsets['itemsets'].apply(lambda x: len(x))
frequent_itemsets
```

:

	support	itemsets	length
0	0.8	(Eggs)	1
1	1.0	(Kidney Beans)	1
2	0.6	(Milk)	1
3	0.6	(Onion)	1
4	0.6	(Yogurt)	1
5	0.8	(Eggs, Kidney Beans)	2
6	0.6	(Eggs, Onion)	2
7	0.6	(Milk, Kidney Beans)	2
8	0.6	(Onion, Kidney Beans)	2
9	0.6	(Yogurt, Kidney Beans)	2
10	0.6	(Eggs, Onion, Kidney Beans)	3

```
frequent_itemsets[ (frequent_itemsets['length'] == 2) &
                   (frequent_itemsets['support'] >= 0.8) ]
```

:

	support	itemsets	length
5	0.8	(Eggs, Kidney Beans)	2

```
frequent_itemsets[ (frequent_itemsets['length'] == 3) &
                   (frequent_itemsets['support'] >= 0.6) ]
```

:

	support	itemsets	length
10	0.6	(Eggs, Onion, Kidney Beans)	3

Association Rules

```
1 from mlxtend.frequent_patterns import association_rules
2
3 association_rules(frequent_itemsets, metric="confidence", min_threshold=0.7)
```

	antecedents	consequents	antecedent support	consequent support	support	confidence
0	(Eggs)	(Kidney Beans)	0.8	1.0	0.8	1.00
1	(Kidney Beans)	(Eggs)	1.0	0.8	0.8	0.80
2	(Eggs)	(Onion)	0.8	0.6	0.6	0.75
3	(Onion)	(Eggs)	0.6	0.8	0.6	1.00
4	(Milk)	(Kidney Beans)	0.6	1.0	0.6	1.00
5	(Onion)	(Kidney Beans)	0.6	1.0	0.6	1.00
6	(Yogurt)	(Kidney Beans)	0.6	1.0	0.6	1.00
7	(Eggs, Onion)	(Kidney Beans)	0.6	1.0	0.6	1.00
8	(Eggs, Kidney Beans)	(Onion)	0.8	0.6	0.6	0.75
9	(Onion, Kidney Beans)	(Eggs)	0.6	0.8	0.6	1.00
10	(Eggs)	(Onion, Kidney Beans)	0.8	0.6	0.6	0.75
11	(Onion)	(Eggs, Kidney Beans)	0.6	0.8	0.6	1.00

Misleading “strong” association rule

- Analyzing transactions at *AllElectronics* with respect to the purchase of computer games and videos.
  - 10,000 transactions analyzed
  - 6000 of the customer transactions included computer games
  - 7500 included videos
  - 4000 included both computer games and videos
  - minimum support: 30%
  - minimum confidence: 60%
  - *buys(X, “computer games” => buys (X, “videos”)*
    - *support = 40%, confidence = 66%*

From Association Analysis to Correlation Analysis

- A correlation measure can be used to augment the support–confidence framework for association rules
- *Correlation rules*
  - *A=>B* [*support*, *confidence*, *correlation (lift, leverage, conviction)]*
  - A correlation rule is measured not only by its support and confidence but also by the correlation between itemsets *A* and *B*

Lift

- **Lift** is a simple correlation measure
- The occurrence of itemset A is independent of the occurrence of itemset B if *P(A U B) = P(A)P(B)*
  - otherwise, itemsets A and B are dependent and correlated as events.
- The Lift between the occurrence of A and B can be measured by computing
  - *lift (A=>B) = P(A U B) / P(A)P(B) = P(B|A)/P(B) = confidence(A=>B)/support(B), range: [0, ∞]*
    - *If the resulting value is greater than 1, then A and B are positively correlated,*
    - *If the resulting value is less than 1, then A and B are negatively correlated,*
    - *If the resulting value is equal to 1, then A and B are independent and there is no correlation between them.*

```
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1.2)
rules
```

:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(Eggs)	(Onion)	0.8	0.6	0.6	0.75	1.25	0.12	1.6
1	(Onion)	(Eggs)	0.6	0.8	0.6	1.00	1.25	0.12	inf
2	(Eggs, Kidney Beans)	(Onion)	0.8	0.6	0.6	0.75	1.25	0.12	1.6
3	(Onion, Kidney Beans)	(Eggs)	0.6	0.8	0.6	1.00	1.25	0.12	inf
4	(Eggs)	(Onion, Kidney Beans)	0.8	0.6	0.6	0.75	1.25	0.12	1.6
5	(Onion)	(Eggs, Kidney Beans)	0.6	0.8	0.6	1.00	1.25	0.12	inf

Leverage

- Leverage computes the difference between the observed frequency of A and B appearing together and the frequency that would be expected if A and B were independent.
  - *leverage (A=>B) = support(A=>B) – support(A) x support(B),*
  - *range: [-1, +1]*
  - leverage value of 0 indicates independence

```
rules = association_rules(frequent_itemsets, metric="leverage", min_threshold=0.12)
rules
```

:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(Eggs)	(Onion)	0.8	0.6	0.6	0.75	1.25	0.12	1.6
1	(Onion)	(Eggs)	0.6	0.8	0.6	1.00	1.25	0.12	inf
2	(Eggs, Kidney Beans)	(Onion)	0.8	0.6	0.6	0.75	1.25	0.12	1.6
3	(Onion, Kidney Beans)	(Eggs)	0.6	0.8	0.6	1.00	1.25	0.12	inf
4	(Eggs)	(Onion, Kidney Beans)	0.8	0.6	0.6	0.75	1.25	0.12	1.6
5	(Onion)	(Eggs, Kidney Beans)	0.6	0.8	0.6	1.00	1.25	0.12	inf

Conviction

- A high conviction value means that the consequent is highly depending on the antecedent.
  - *confidence (A=>B) = (1 – support(B)) / (1 – confidence (A,B))*
  - *range: [0, ∞]*
  - For instance, in the case of a perfect confidence score, the denominator becomes 0 (due to 1 - 1) for which the conviction score is defined as 'inf'
  - if items are independent, the conviction is 1.

```
rules = association_rules(frequent_itemsets, metric="conviction", min_threshold=2)
rules
```

:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(Eggs)	(Kidney Beans)	0.8	1.0	0.8	1.0	1.00	0.00	inf
1	(Onion)	(Eggs)	0.6	0.8	0.6	1.0	1.25	0.12	inf
2	(Milk)	(Kidney Beans)	0.6	1.0	0.6	1.0	1.00	0.00	inf
3	(Onion)	(Kidney Beans)	0.6	1.0	0.6	1.0	1.00	0.00	inf
4	(Yogurt)	(Kidney Beans)	0.6	1.0	0.6	1.0	1.00	0.00	inf
5	(Eggs, Onion)	(Kidney Beans)	0.6	1.0	0.6	1.0	1.00	0.00	inf
6	(Onion, Kidney Beans)	(Eggs)	0.6	0.8	0.6	1.0	1.25	0.12	inf
7	(Onion)	(Eggs, Kidney Beans)	0.6	0.8	0.6	1.0	1.25	0.12	inf

```
# at least 2 antecedents

rules["antecedent_len"] = rules["antecedents"].apply(lambda x: len(x))
rules
```

:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	antecedent_len
0	(Eggs)	(Kidney Beans)	0.8	1.0	0.8	1.0	1.00	0.00	inf	1
1	(Onion)	(Eggs)	0.6	0.8	0.6	1.0	1.25	0.12	inf	1
2	(Milk)	(Kidney Beans)	0.6	1.0	0.6	1.0	1.00	0.00	inf	1
3	(Onion)	(Kidney Beans)	0.6	1.0	0.6	1.0	1.00	0.00	inf	1
4	(Yogurt)	(Kidney Beans)	0.6	1.0	0.6	1.0	1.00	0.00	inf	1
5	(Eggs, Onion)	(Kidney Beans)	0.6	1.0	0.6	1.0	1.00	0.00	inf	2
6	(Onion, Kidney Beans)	(Eggs)	0.6	0.8	0.6	1.0	1.25	0.12	inf	2
7	(Onion)	(Eggs, Kidney Beans)	0.6	0.8	0.6	1.0	1.25	0.12	inf	1

```
# a confidence > 0.75
# a lift score > 1.2
# a leverage > 0.1
# a conviction > 2.0

rules[ (rules['antecedent_len'] >= 2) &
       (rules['confidence'] > 0.75) &
       (rules['lift'] > 1.2) &
       (rules['leverage'] > 0.1) &
       (rules['conviction'] > 2.0) ]
```

:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	antecedent_len
6	(Onion, Kidney Beans)	(Eggs)	0.6	0.8	0.6	1.0	1.25	0.12	inf	2

Measures of Correlation

- *"Buy walnuts $\Rightarrow$ buy milk* [1%, 80%]" is misleading if 85% of customers buy milk
- Support and confidence are not good to indicate correlations
- Over 20 interestingness measures have been proposed (see Tan, Kumar, Sritastava @KDD'02)
- Which are good ones?

symbol	measure	range	formula
ϕ	ϕ-coefficient	-1 . . . 1	$\frac{P(A,B)-P(A)P(B)}{\sqrt{P(A)P(B)(1-P(A))(1-P(B))}}$
Q	Yule's Q	-1 . . . 1	$\frac{P(A,B)P(\overline{A},\overline{B})-P(A,\overline{B})P(\overline{A},B)}{P(A,B)P(\overline{A},\overline{B})+P(A,\overline{B})P(\overline{A},B)}$
Y	Yule's Y	-1 . . . 1	$\frac{\sqrt{P(A,B)P(\overline{A},\overline{B})}-\sqrt{P(A,\overline{B})P(\overline{A},B)}}{\sqrt{P(A,B)P(\overline{A},\overline{B})}+\sqrt{P(A,\overline{B})P(\overline{A},B)}}$
k	Cohen's	-1 . . . 1	$\frac{P(A,B)+P(\overline{A},\overline{B})-P(A)P(B)-P(\overline{A})P(\overline{B})}{1-P(A)P(B)-P(\overline{A})P(\overline{B})}$
PS	Piatetsky-Shapiro's	-0.25 . . . 0.25	$P(A,B)-P(A)P(B)$
F	Certainty factor	-1 . . . 1	$\max(\frac{P(B\|A)-P(B)}{1-P(B)}, \frac{P(A\|B)-P(A)}{1-P(A)})$
AV	added value	-0.5 . . . 1	$\max(P(B\|A)-P(B), P(A\|B)-P(A))$
K	Klosgen's Q	-0.33 . . . 0.38	$\sqrt{P(A,B)}\max(P(B\|A)-P(B), P(A\|B)-P(A))$
g	Goodman-kruskal's	0 . . . 1	$\frac{\Sigma_j \max_k P(A_j,B_k)+\Sigma_k \max_j P(A_j,B_k)-\max_j P(A_j)-\max_k P(B_k)}{2-\max_j P(A_j)-max_k P(B_k)}$
M	Mutual Information	0 . . . 1	$\frac{\Sigma_i\Sigma_j P(A_i,B_j)\log\frac{P(A_i,B_j)}{P(A_i)P(B_J)}}{\min(-\Sigma_i P(A_i)\log P(A_i)\log P(A_i), -\Sigma_i P(B_i)\log P(B_i)\log P(B_i))}$
J	J-Measure	0 . . . 1	$\max(P(A,B)\log(\frac{P(B\|A)}{P(B)})+P(A\overline{B})\log(\frac{P(\overline{B}\|A)}{P(\overline{B})}))$ $P(A,B)\log(\frac{P(A\|B)}{P(A)})+P(\overline{A}B)\log(\frac{P(\overline{A}\|B)}{P(\overline{A})})$
G	Gini index	0 . . . 1	$\max(P(A)[P(B\|A)^2+P(\overline{B}\|A)^2]+P(\overline{A}[P(B\|\overline{A})^2+P(\overline{B}\|\overline{A})^2]-P(B)^2-P(\overline{B})^2,$ $P(B)[P(A\|B)^2+P(\overline{A}\|B)^2]+P(\overline{B}[P(A\|\overline{B})^2+P(\overline{A}\|\overline{B})^2]-P(A)^2-P(\overline{A})^2)$
s	support	0 . . . 1	$P(A,B)$
c	confidence	0 . . . 1	$max(P(B\|A), P(A\|B))$
L	Laplace	0 . . . 1	$\max(\frac{NP(A,B)+1}{NP(A)+2}, \frac{NP(A,B)+1}{NP(B)+2})$
IS	Cosine	0 . . . 1	$\frac{P(A,B)}{\sqrt{P(A)P(B)}}$
γ	coherence(Jaccard)	0 . . . 1	$\frac{P(A,B)}{P(A)+P(B)-P(A,B)}$
α	all_confidence	0 . . . 1	$\frac{P(A,B)}{\max(P(A),P(B))}$
o	odds ratio	0 . . . ∞	$\frac{P(A,B)P(\overline{A},\overline{B})}{P(\overline{A},B)P(A,\overline{B})}$
V	Conviction	0.5 . . . ∞	$\max(\frac{P(A)P(\overline{B})}{P(A\overline{B})}, \frac{P(B)P(\overline{A})}{P(B\overline{A})})$
λ	lift	0 . . . ∞	$\frac{P(A,B)}{P(A)P(B)}$
S	Collective strength	0 . . . ∞	$\frac{P(A,B)+P(\overline{AB})}{P(A)P(B)+P(\overline{A})P(\overline{B})} \times \frac{1-P(A)P(B)-P(\overline{A})P(\overline{B})}{1-P(A,B)-P(\overline{AB})}$
χ^2	χ^2	0 . . . ∞	$\Sigma_i \frac{(P(A_i)-E_i)^2}{E_i}$

Lesson B-3

Frequent Pattern Mining

-FP-Tree Algorithm, Mining Multilevel Association Rules, Mining Multidimensional Association Rules, Mining Rare Patterns, Mining Negative Patterns

FP-Growth Algorithm

- The Apriori Algorithm has a major shortfall
  - required multiple scans of the databases (datasets) to check the support count of each item and itemsets
- FP-growth is an improved version of the Apriori Algorithm
  - scans the database twice and used a tree structure (FP-tree) to store all the information

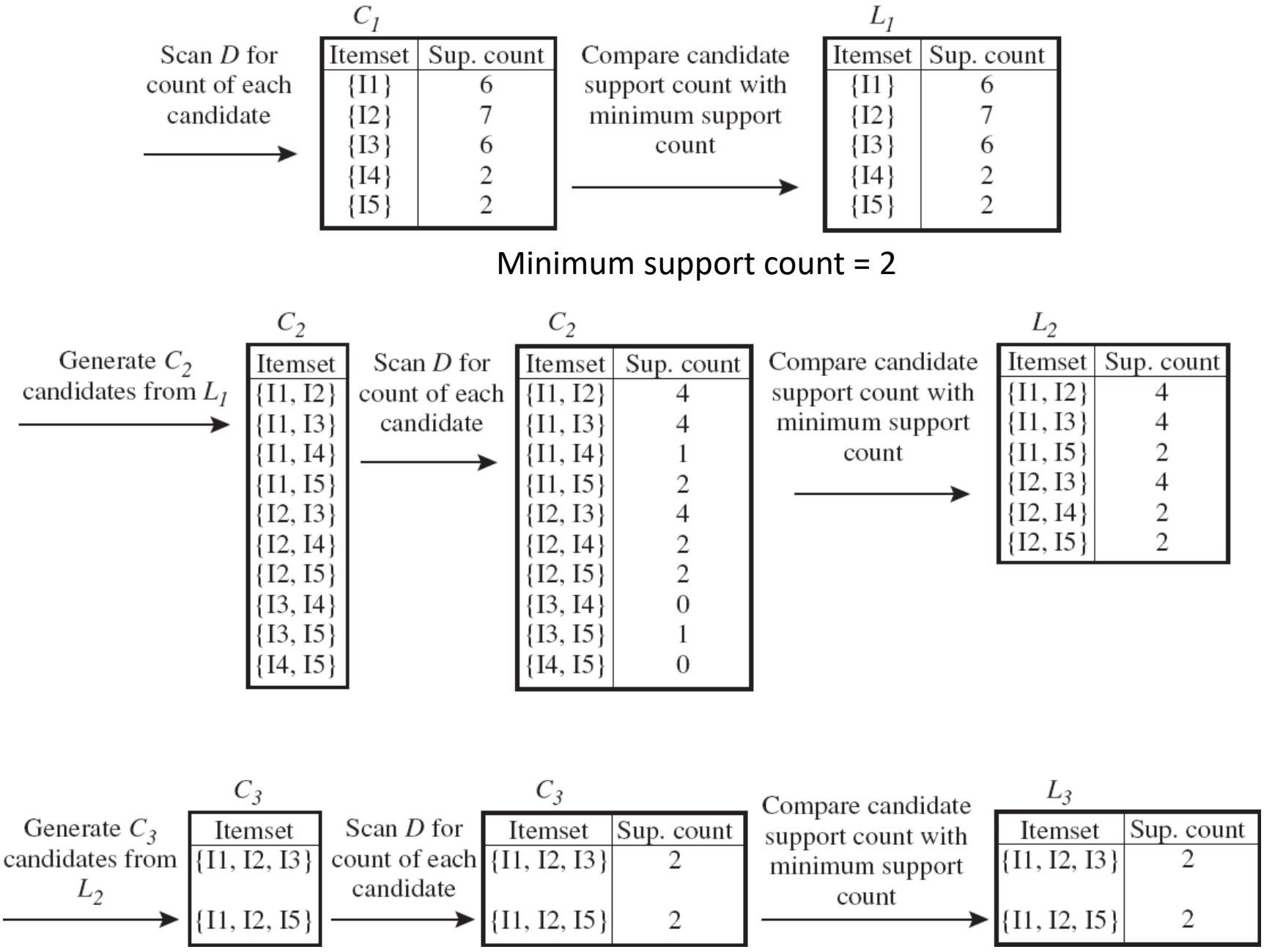
FP-Tree Algorithm

- Step 1: Deduce the ordered frequent items. For items with the same frequency, the order is given by the alphabetical order.
- Step 2: Construct the FP-tree from the above data
- Step 3: From the FP-tree above, construct the FP-conditional tree for each item (or itemset).
- Step 4: Determine the frequent patterns.

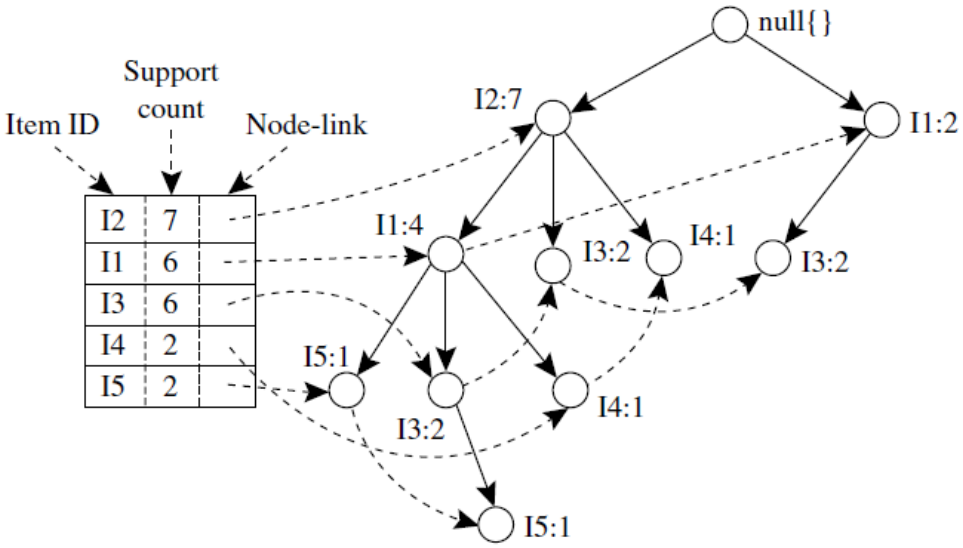
Transactional Database

TID	*List of item_IDs*
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

Transactional data for an AllElectronics branch.



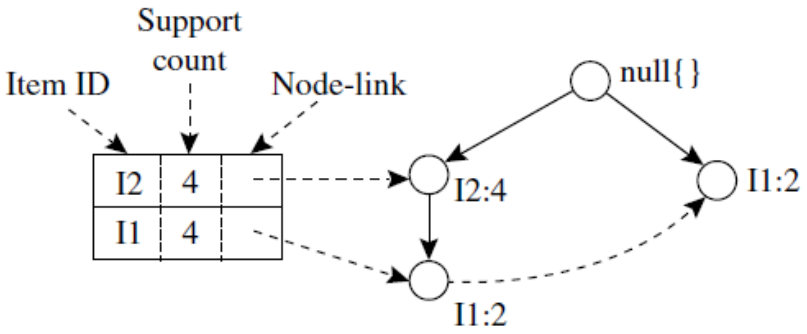
Generation of candidate itemsets and frequent itemsets, where the minimum support count is 2.



An FP-tree registers compressed, frequent pattern information.

Mining the FP-Tree by Creating Conditional (Sub-)Pattern Bases

Item	*Conditional Pattern Base*	*Conditional FP-tree*	*Frequent Patterns Generated*
I5	{{I2, I1: 1}, {I2, I1, I3: 1}}	⟨I2: 2, I1: 2⟩	{I2, I5: 2}, {I1, I5: 2}, {I2, I1, I5: 2}
I4	{{I2, I1: 1}, {I2: 1}}	⟨I2: 2⟩	{I2, I4: 2}
I3	{{I2, I1: 2}, {I2: 2}, {I1: 2}}	⟨I2: 4, I1: 2⟩, ⟨I1: 2⟩	{I2, I3: 4}, {I1, I3: 4}, {I2, I1, I3: 2}
I1	{{I2: 4}}	⟨I2: 4⟩	{I2, I1: 4}



The conditional FP-tree associated with the conditional node I3.

Algorithm: **FP_growth.** Mine frequent itemsets using an FP-tree by pattern fragment growth.

Input:

- D, a transaction database;
- *min_sup*, the minimum support count threshold.

Output: The complete set of frequent patterns.

Method:

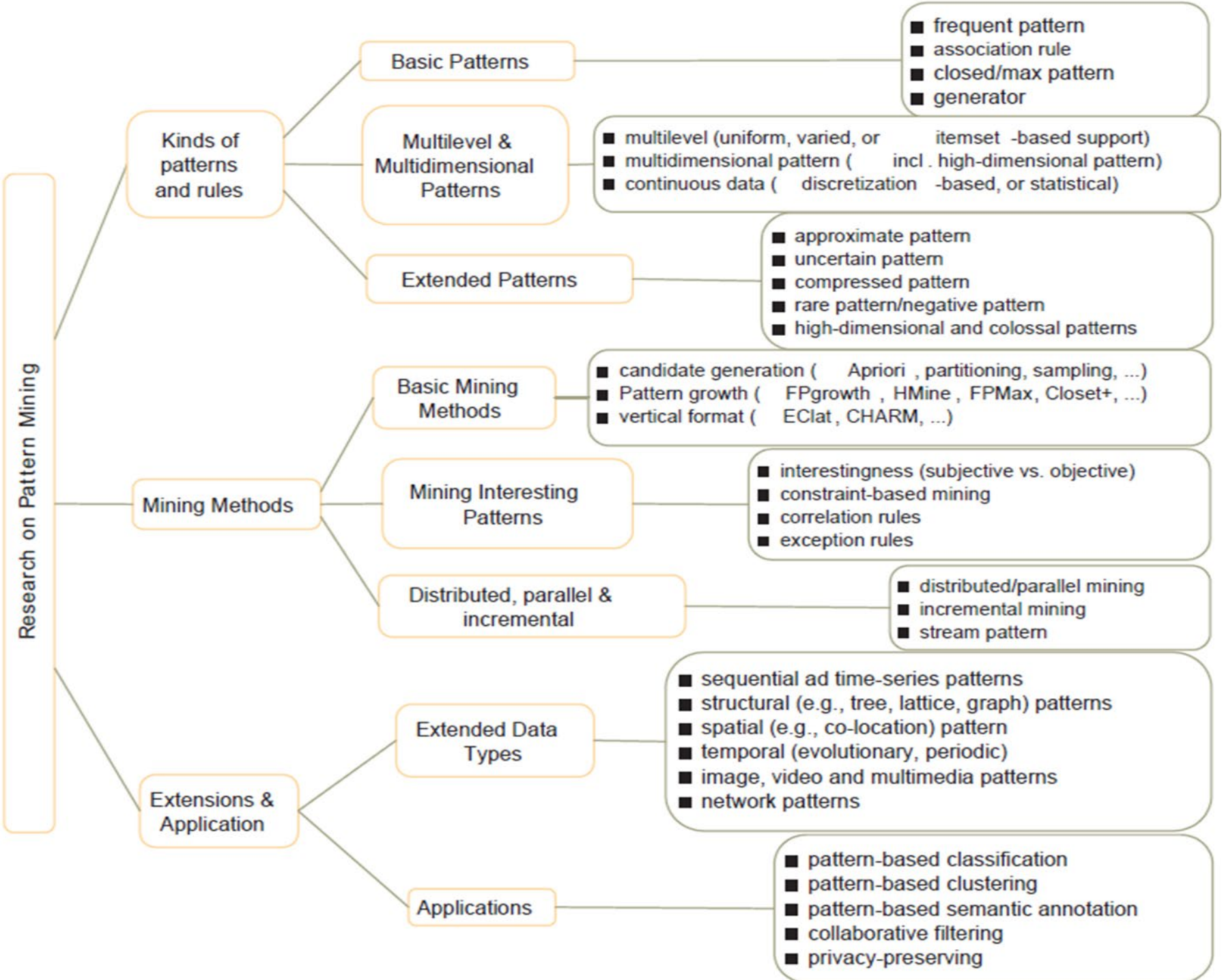
1. The FP-tree is constructed in the following steps:
   (a) Scan the transaction database D once. Collect F, the set of frequent items, and their support counts. Sort F in support count descending order as L, the *list* of frequent items.
   (b) Create the root of an FP-tree, and label it as "null." For each transaction *Trans* in D do the following.
   Select and sort the frequent items in *Trans* according to the order of L. Let the sorted frequent item list in *Trans* be $[p|P]$, where p is the first element and P is the remaining list. Call insert_tree($[p|P]$, T), which is performed as follows. If T has a child N such that $N.item\text{-}name = p.item\text{-}name$, then increment N's count by 1; else create a new node N, and let its count be 1, its parent link be linked to T, and its node-link to the nodes with the same *item-name* via the node-link structure. If P is nonempty, call insert_tree(P, N) recursively.
2. The FP-tree is mined by calling FP_growth(*FP_tree*, *null*), which is implemented as follows.

```
procedure FP_growth(Tree, α)
(1)  if Tree contains a single path P then
(2)     for each combination (denoted as β) of the nodes in the path P
(3)        generate pattern β ∪ α with support_count = minimum support count of nodes in β;
(4)  else for each a_i in the header of Tree {
(5)     generate pattern β = a_i ∪ α with support_count = a_i.support_count;
(6)     construct β's conditional pattern base and then β's conditional FP_tree Tree_β;
(7)     if Tree_β ≠ ∅ then
(8)        call FP_growth(Tree_β, β); }
```

FP-growth algorithm for discovering frequent itemsets without candidate generation.

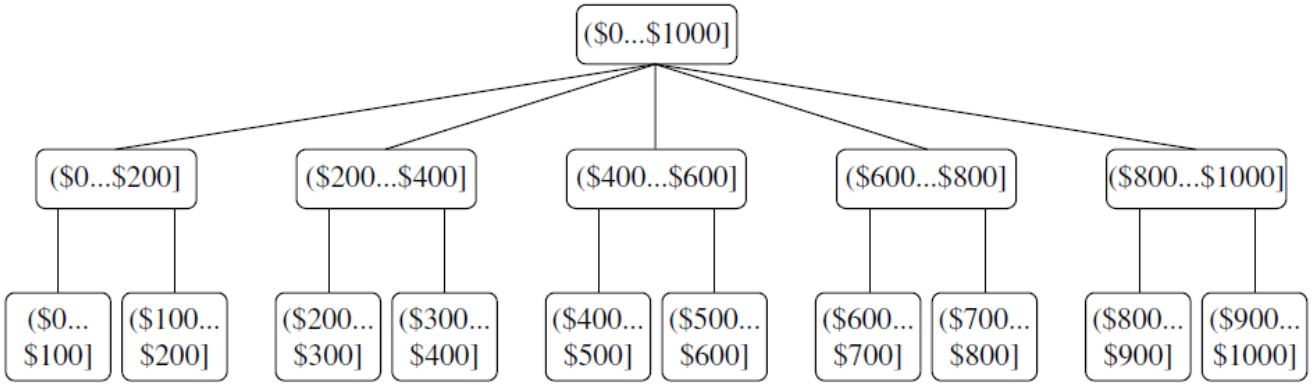
Benefits of the FP-tree

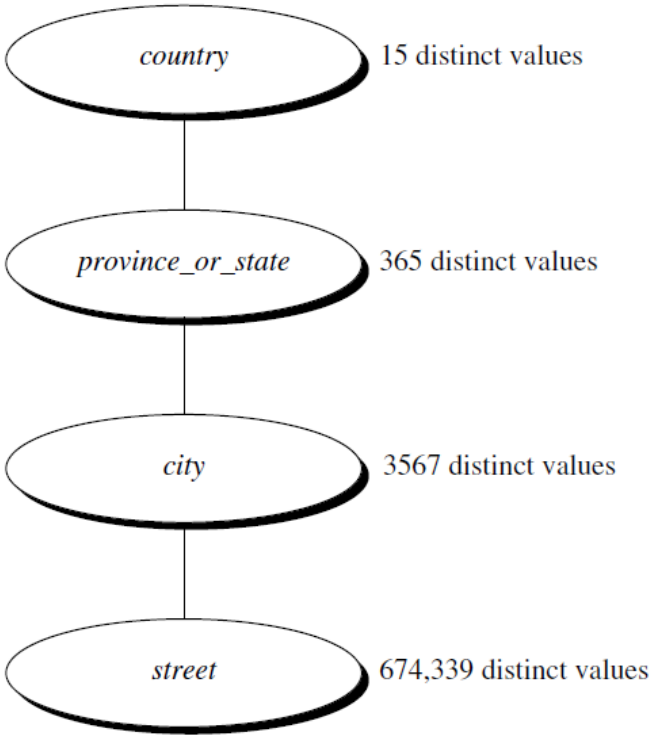
- Completeness
  - Preserve complete information for frequent pattern mining
  - Never break a long pattern of any transaction
- Compactness
  - Reduce irrelevant info—infrequent items are gone
  - Items in frequency descending order: the more frequently occurring, the more likely to be shared
  - Never be larger than the original database


Research on Pattern Mining
Kinds of patterns and rules
Basic Patterns
frequent pattern
association rule
closed/max pattern
generator
Multilevel & Multidimensional Patterns
multilevel (uniform, varied, or itemset -based support)
multidimensional pattern (incl . high-dimensional pattern)
continuous data (discretization -based, or statistical)
Extended Patterns
approximate pattern
uncertain pattern
compressed pattern
rare pattern/negative pattern
high-dimensional and colossal patterns
Mining Methods
Basic Mining Methods
candidate generation (Apriori , partitioning, sampling, ...)
Pattern growth (FPgrowth , HMine , FPMax, Closet+, ...)
vertical format (EClat , CHARM, ...)
Mining Interesting Patterns
interestingness (subjective vs. objective)
constraint-based mining
correlation rules
exception rules
Distributed, parallel & incremental
distributed/parallel mining
incremental mining
stream pattern
Extensions & Application
Extended Data Types
sequential ad time-series patterns
structural (e.g., tree, lattice, graph) patterns
spatial (e.g., co-location) pattern
temporal (evolutionary, periodic)
image, video and multimedia patterns
network patterns
Applications
pattern-based classification
pattern-based clustering
pattern-based semantic annotation
collaborative filtering
privacy-preserving

Concept Hierarchy Generation

- Concept Hierarchy Generation for Numeric Data
  - A concept hierarchy for a given numerical attribute defines a discretization of the attribute
  - Recursively reduce the data by collecting and replacing low level concepts by higher level concepts
- Concept Hierarchy Generation for Categorical Data
  - Specification of an ordering of attributes explicitly at the schema level by users or experts
    - street < city < state < country
  - Automatic Concept Hierarchy Generation
    - Based on the number of distinct values per attributes


($0...$1000]
($0...$200]
($200...$400]
($400...$600]
($600...$800]
($800...$1000]
($0... $100]
($100... $200]
($200... $300]
($300... $400]
($400... $500]
($500... $600]
($600... $700]
($700... $800]
($800... $900]
($900... $1000]

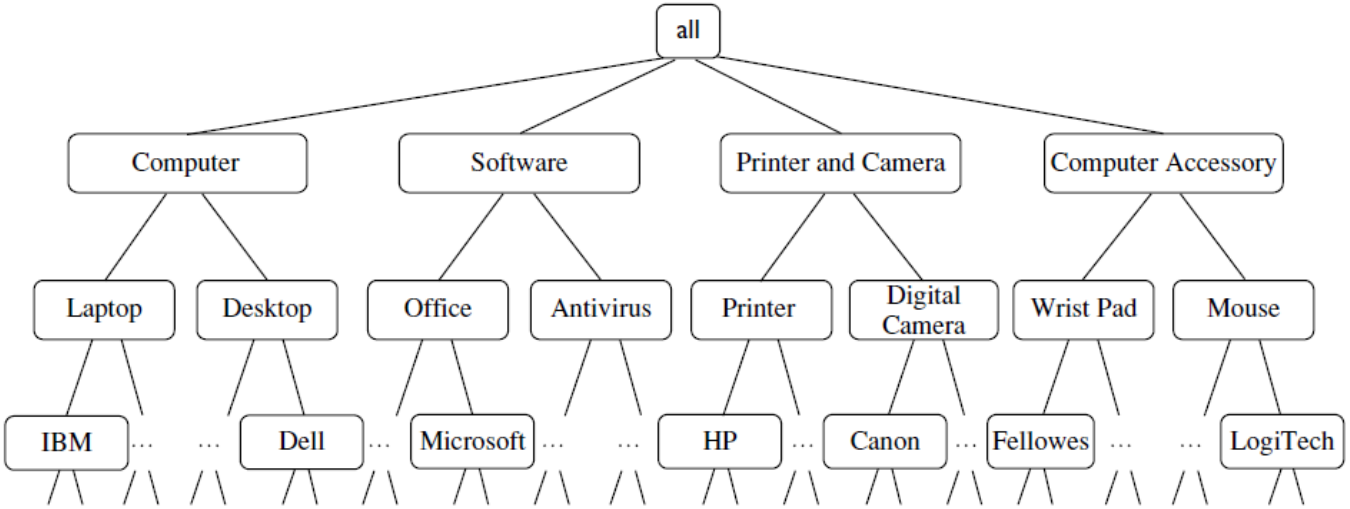

country
15 distinct values
province_or_state
365 distinct values
city
3567 distinct values
street
674,339 distinct values

Mining Multilevel Association Rules

- Data mining systems should provide capabilities for mining association rules at multiple levels of abstraction
- Items often form hierarchies
- Flexible support settings
  - Items at the lower level are expected to have lower support

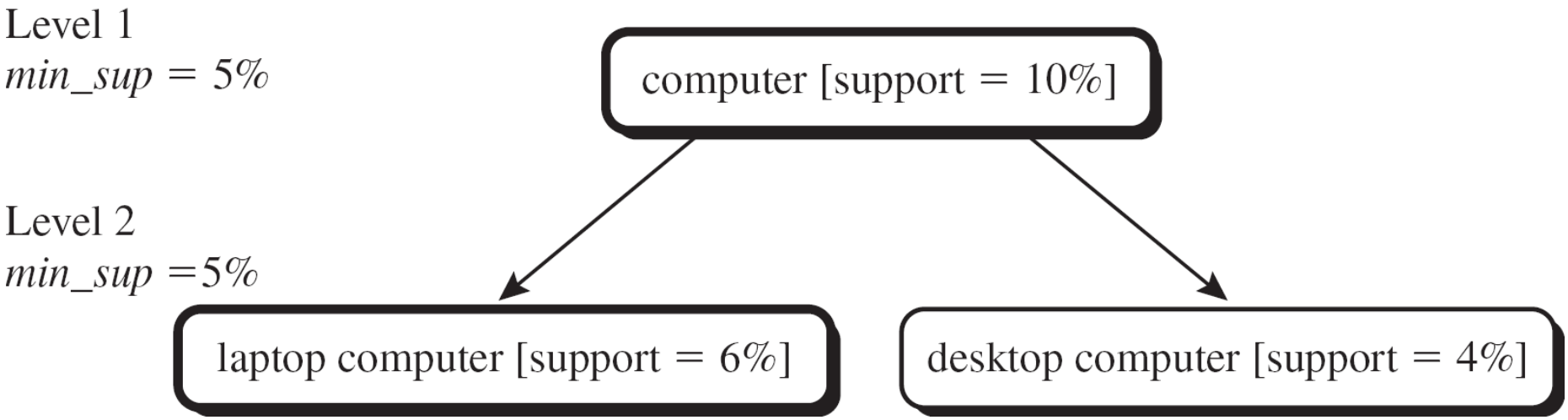
Concept Hierarchy for Computer Items

TID	Items Purchased
T100	Apple 17″ MacBook Pro Notebook, HP Photosmart Pro b9180
T200	Microsoft Office Professional 2010, Microsoft Wireless Optical Mouse 5000
T300	Logitech VX Nano Cordless Laser Mouse, Fellowes GEL Wrist Rest
T400	Dell Studio XPS 16 Notebook, Canon PowerShot SD1400
T500	Lenovo ThinkPad X200 Tablet PC, Symantec Norton Antivirus 2010
…	…

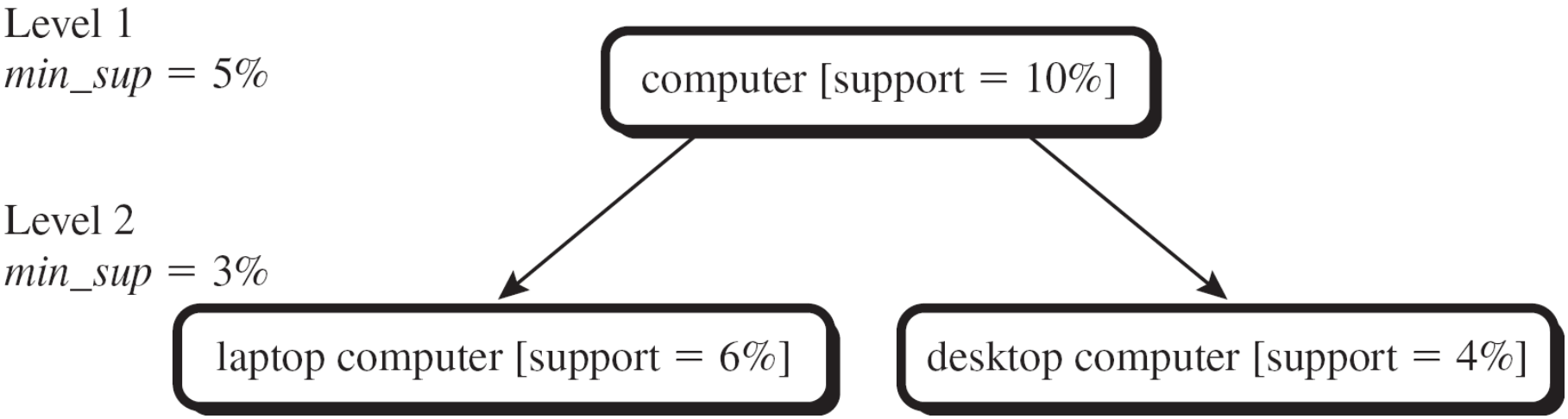


Mining Multilevel Association Rules

- For each level, any algorithm for discovering frequent itemsets may be used, such as Apriori or FP-tree
  - Using uniform minimum support for all levels (referred to as uniform support)
  - Using reduced minimum support at lower levels (referred to as reduced support)
  - Using item or group-based minimum support (referred to as group_based support)



Using uniform minimum support for all levels (referred to as **uniform support**)



Using reduced minimum support at lower levels (referred to as **reduced support**):

Flexible Min-Support Thresholds

- Using item or group-based minimum support (referred to as group-based support):
  - For example, a user could set up the minimum support thresholds based on product price or on items of interest
    - E.g., {diamond, watch, camera}: 0.05%; {bread, milk}: 5%; …

Mining Multilevel Association Rules

- The generation of many redundant rules across multiple levels of abstractions due to the ancestor relationships among items
- A rule is *redundant* if its support is close to the "expected" value, based on the rule's ancestor
  - *buys(X, "laptop computer") ⇒ buys(X, "HP printer")*

    [support = 8%, confidence = 70%]
  - *buys(X, "IBM laptop computer") ⇒ buys(X, "HP printer")*

    [support = 2%, confidence = 70%]

Mining Multidimensional Association Rules

- Single-dimensional rules:

  $buys(X, \text{“}milk\text{”}) \Rightarrow buys(X, \text{“}bread\text{”})$

- Multi-dimensional rules: $\geq$ 2 dimensions or predicates
  - Inter-dimension assoc. rules (*no repeated predicates*)

    $age(X,\text{”}19\text{-}25\text{”}) \wedge occupation(X,\text{“}student\text{”}) \Rightarrow buys(X, \text{“}coke\text{”})$
  - hybrid-dimension assoc. rules (*repeated predicates*)

    $age(X,\text{”}19\text{-}25\text{”}) \wedge buys(X, \text{“}popcorn\text{”}) \Rightarrow buys(X, \text{“}coke\text{”})$

Mining Rare Patterns

- In jewelry sales data, sales of diamond watches are rare; however, patterns involving the selling of diamond watches could be interesting.

Mining Negative Patterns

- In supermarket data, if we find that customers frequently buy Coca-Cola Classic or Diet Coke but not both, then buying Coca-Cola Classic and buying Diet Coke together is considered a negative (correlated) pattern

Lesson B-4

Bayesian Classification

Probability Basics for Data Mining

- The branch of mathematics that deals with measuring the likelihood (or uncertainty) around events to predict future events based on their likelihood
- Computing the likelihood of a future event
  - Use the relative frequency of the event in the past
  - In a predictive analytics context, the *relative frequency* is a standard approach

Terminology of Probability

- Probability
- Experiment
- Sample space
- Outcome
- Event
- Random variable
- Probability (mass/density) function

Map the Terminology of Probability into the Language of Predictive Analysis

- Random variables → features (attributes)
- Sample space → the set of all possible combinations of assignments of values to features
- An experiment whose outcome has been already been stored → a row in dataset
- An experiment whose outcome we do not yet know but would like to predict → the prediction task for which we are building a model
- An event is any subset of an experiment → an event may describe an assignment of values to all the features in the domain (a full row in the dataset) or an assignment to one or more features in the domain
- The value returned by a probability function for an event → the **relative frequency** of that event in the dataset

A Dataset of Instances from The Sample Space of Two Random Variables

ID	DICE 1	DICE2
1	3	4
2	1	5
3	6	5
4	3	3
5	1	1

Bayes' Theorem

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

- Where A and B are events and P(B) ≠ 0

Properties of Probability Mass Function

All probability should satisfy the following axioms:

- For any event A, $0 \leq P(A) \leq 1$
- If $A_1, A_2, \ldots, A_n$ is a finite collection of mutually exclusive events, then

$$P(A_1 \cup A_2 \cup \cdots \cup A_n) = \sum_{i=1}^{n} P(A_i)$$

Union, Intersection, Complement, Mutually Exclusive of Events

- The **union** of two events *A* and *B*, denoted by *A* ∪ *B* is the event consisting of all outcomes that either in *A* or in *B* or in both events.
- The **intersection** of two events *A* and *B*, denoted by *A* ∩ *B* is the event consisting of all outcomes that are in both *A* and *B*.
- The **complement** of an event, denoted by *A′*, is the set of all outcomes in *S* that are not contained in *A*.
- When *A* and *B* have no outcomes in common, they are said to be **mutually exclusive** or **disjoint** events.

Random Variables

- A **random variable** represents the outcome of a probabilistic experiment.
  - Each random variable has a range of possible values (outcomes).
- A random variable is said to be **discrete** if its set of possible values is discrete set.
  - Possible outcomes of a random variable `Mood`: *Happy* and *Sad*
  - Each outcome has a probability.
    - The probabilities for all possible outcomes must sum to 1.
  - For example:
    - P(Mood=*Happy*) = 0.7
    - P(Mood=*Sad*) = 0.3

Joint Probability

- Joint probability is the probability of two events happening together.
- The two events are usually designated event A and event B.
  - In probability terminology, it can be written as:
  - p(A and B) or
  - p(A ∩ B)
- Joint probability can also be described as the probability of the intersection of two (or more) events.
  - The intersection can be represented by a Venn diagram

Joint Probability

- The `Mood` can take 2 possible values: *happy, sad*.
- The `Weather` can take 3 possible vales: *sunny, rainy, cloudy.*
- Lets say we know:
  - P(Mood=*happy* ∩ Weather=*rainy*) = 0.25
  - P(Mood=*happy* ∩ Weather=*sunny*) = 0.4
  - P(Mood=*happy* ∩ Weather=*cloudy*) = 0.05

Joint Probabilities

	Sunny	Rainy	Cloudy
Happy	0.4	0.25	0.05
Sad	0.05	0.1	0.15

- P(Mood=*Happy*) = 0.25 + 0.4 + 0.05 = 0.7
- P(Mood=*Sad*) = ?
- Two random variables *A* and *B*
  - A has *m* possible outcomes $A_1, \ldots, A_m$
  - B has *n* possible outcomes $B_1, \ldots, B_n$

$$P(A = A_i) = \sum_{j=1}^{n} P((A = A_i) \bigcap (B = B_j))$$

Joint Probabilities

	Sunny	Rainy	Cloudy
Happy	0.4	0.25	0.05
Sad	0.05	0.1	0.15

- P(Weather=*Sunny*)=?
- P(Weather=*Rainy*)=?
- P(Weather=*Cloudy*)=?

$$P(B = B_i) = \sum_{j=1}^{m} P((A = A_j) \bigcap (B = B_i))$$

Joint Probabilities

ID	DICE 1	DICE2
1	3	4
2	1	5
3	6	5
4	3	3
5	1	1

- *In terms of rows in a dataset*, the probability of a joint event computation is simply the number of rows where the set of assignments listed in the joint event holds divided by the total number of rows in the dataset.

Conditional Probability

- In probability theory, conditional probability is a measure of the probability of an event occurring, given that another event (by assumption, presumption, assertion or evidence) has already occurred
- For any two events *A* and *B* with P(*B*) > 0, the conditional probability of *A* given that *B* has occurred is defined by

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$

Conditional Probability

- $P(A = A_i \mid B = B_j)$ represents the probability of $A = A_i$ given that we know $B = B_j$. This is called **conditional probability**.

$$P(A = A_i \mid B = B_j) = \frac{P((A = A_i) \bigcap (B = B_j))}{P(B = B_j)}$$

Conditional Probability

	Sunny	Rainy	Cloudy
Happy	0.4	0.25	0.05
Sad	0.05	0.1	0.15

- P(Mood=Happy|Weather=Sunny) = ?
- P(Mood=Happy|Weather=Rainy) = ?
- P(Mood=Happy|Weather=Cloudy) = ?
- P(Weather=Cloudy|Mood= Sad) = ?

Conditional Probabilities

ID	DICE 1	DICE2
1	3	4
2	1	5
3	6	5
4	3	3
5	1	1

- P (DICE1 = 6 | DICE2 = 5) =

Conditional Probability

- P(A | B) = 1 is equivalent to $B \Rightarrow A$.
  - Knowing the value of B exactly determines the value of A.
- For example, suppose my dog rarely howls:

  P(MyDogHowls) = 0.01
- But when there is a full moon, he always howls:

  P(MyDogHowls | FullMoon) = 1.0

Conditional Probability and Product Rule

Conditional probability:

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$

Product rule:

$$P(A \cap B) = P(A \mid B)P(B) = P(B \mid A)P(A)$$

Independence

- Two random variables *A* and *B* are said to be **independent** if and only if P(*A* ∩ *B*) = P(*A*)P(*B*).
- Conditional probabilities for independent A and B:

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)} = \frac{P(A)P(B)}{P(B)} = P(A)$$

$$P(B \mid A) = \frac{P(A \cap B)}{P(A)} = \frac{P(A)P(B)}{P(A)} = P(B)$$

- Product rule for independent events
  - If A and B are independent, P(A ∩ B)=P(A)P(B)

Bayes Theorem

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

Where A and B are events and P(B) ≠ 0

The Law of Total Probability

Let $A_1, A_2, \ldots, A_n$ be a collection of n mutually exclusive and exhaustive events with $P(A_i) > 0$ for $i = 1, \ldots, n$. Then for any other event B for which $P(B) > 0$

$$
\begin{aligned}
& P(B) \\
& = P(B \mid A_1)P(A_1) + \ldots. + P(B \mid A_n)P(A_n) \\
& = \sum_{i=1}^{n} P(B \mid A_i)P(A_i)
\end{aligned}
$$

Conditional Probability and The Law of Total Probability

Let $A_1, A_2, \ldots, A_n$ be a collection of n mutually exclusive and exhaustive events with $P(A_i) > 0$ for $i = 1, \ldots, n$. Then for any other event B for which $P(B) > 0$

$$P(A_k \mid B) = \frac{P(B \mid A_k)P(A_k)}{P(B)} = \frac{P(B \mid A_k)P(A_k)}{\sum_{i=1}^{n} P(B \mid A_i)P(A_i)} \quad k = 1, \ldots, n$$

Prior Probabilities and Posterior Probabilities

- The type of probabilities we have calculated so far are known as **prior probabilities** or **unconditional probabilities**.
- We want to know the probability of an event in the context where one or more other events are known to have happened.
  - This type of probability, where we take one or more events to already hold, is known as a **posterior probability**, because it is calculated *after* other events have happened.
  - It is also commonly known as a **conditional probability**, because the probability calculated is valid conditional on the given events (or evidence).

Bayesian Classification

- A statistical classifier: performs *probabilistic prediction, i.e.,* predicts class membership probabilities
- Foundation: Based on Bayes' Theorem.
- Performance: A simple Bayesian classifier, *naïve Bayesian classifier*, has comparable performance with decision tree and selected neural network classifiers
- Incremental: Each training example can incrementally increase/decrease the probability that a hypothesis is correct — prior knowledge can be combined with observed data

Bayes' Theorem in Data Analytics

- Bayes' Theorem:

$$P(H|\mathbf{X}) = \frac{P(\mathbf{X}|H)P(H)}{P(\mathbf{X})} = P(\mathbf{X}|H) \times P(H) / P(\mathbf{X})$$

  - Let **X** be a data sample ("*evidence*"): class label is unknown
  - Let H be a *hypothesis* that X belongs to class C
  - Classification is to determine P(H|**X**), (i.e., *posteriori probability):* the probability that the hypothesis holds given the observed data sample **X**
  - P(H) (*prior probability*): the initial probability
    - E.g., **X** will buy computer, regardless of age, income, …
  - P(**X**): probability that sample data is observed
  - P(**X**|H) (likelihood): the probability of observing the sample **X**, given that the hypothesis holds
    - e.g., Given that **X** will buy computer, the prob. that X is youth, medium income

Prediction Based on Bayes' Theorem

- Given training data **X**, *posteriori probability of a hypothesis* H, P(H|**X**), follows the Bayes' theorem

$$P(H|\mathbf{X}) = \frac{P(\mathbf{X}|H)P(H)}{P(\mathbf{X})} = P(\mathbf{X}|H) \times P(H) / P(\mathbf{X})$$

- Informally, this can be viewed as

  posteriori = likelihood x prior/evidence

- Predicts **X** belongs to C_i iff the probability $P(C_i|\mathbf{X})$ is the highest among all the $P(C_k|X)$ for all the *k* classes
- Practical difficulty: It requires initial knowledge of many probabilities, involving significant computational cost

Classification is to Derive the Maximum Posteriori

- Let D be a training set of tuples and their associated class labels, and each tuple is represented by an n-D attribute vector **X** = (x_1, x_2, …, x_n)

$$P(\mathbf{X} \mid C_i) = \prod_{k=1}^{n} P(x_k \mid C_i) = P(x_1 \mid C_i) \times P(x_2 \mid C_i) \times \ldots \times P(x_n \mid C_i)$$

- Suppose there are *m* classes C_1, C_2, …, C_m.
- Classification is to derive the maximum posteriori, i.e., the maximal $P(C_i \mid \mathbf{X})$
- This can be derived from Bayes' theorem

$$P(C_i \mid \mathbf{X}) = \frac{P(\mathbf{X} \mid C_i) P(C_i)}{P(\mathbf{X})}$$

- Since P(X) is constant for all classes, only

$$P(C_i \mid \mathbf{X}) = P(\mathbf{X} \mid C_i) P(C_i)$$

needs to be maximized

Naïve Bayes Classifier

- A simplified assumption: attributes are conditionally independent (i.e., no dependence relation between attributes):

$$P(\mathbf{X} \mid C_i) = \prod_{k=1}^{n} P(x_k \mid C_i) = P(x_1 \mid C_i) \times P(x_2 \mid C_i) \times \ldots \times P(x_n \mid C_i)$$

- This greatly reduces the computation cost: Only counts the class distribution
- If A_k is categorical, $P(x_k|C_i)$ is the # of tuples in C_i having value x_k for A_k divided by $|C_{i,D}|$ (# of tuples of C_i in D)
- If A_k is continuous-valued, $P(x_k|C_i)$ is usually computed based on Gaussian distribution with a mean μ and standard deviation σ

$$g(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

and $P(x_k|C_i)$ is

$$P(\mathbf{X} \mid C_i) = g(x_k, \mu_{C_i}, \sigma_{C_i})$$

Naïve Bayesian Classifier: Training Dataset

RID	*age*	*income*	*student*	*credit_rating*	*Class: buys_computer*
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle_aged	medium	no	excellent	yes
13	middle_aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

Class-labeled training tuples from AllElectronics customer database.

Naïve Bayesian Classifier: Training Dataset

- Class:

  C1:*buys_computer* = yes

  C2:*buys_computer* = no
- Data sample

  X = (*age =youth, income = medium, student = yes, credit_rating = fair*)

Naïve Bayesian Classifier: An Example

- $P(C_i)$: P(*buys_computer* = yes) = 9/14 = 0.643

  P(*buys_computer* = no) = 5/14= 0.357

- Compute $P(\mathbf{X}|C_i)$ for each class

P(*age* = youth | *buys_computer* = yes) = 2/9 = 0.222

P(*income* = medium | *buys_computer* = yes) = 4/9 = 0.444

P(*student* = yes | *buys_computer* = yes) = 6/9 = 0.667

P(*credit_rating* = fair | *buys_computer* = yes) = 6/9 = 0.667

P(*age* = youth | *buys_computer* = no) = 3/5 = 0.6

P(*income* = medium | *buys_computer* = no) = 2/5 = 0.4

P(*student* = yes | *buys_computer* = no) = 1/5 = 0.2

P(*credit_rating* = fair | *buys_computer* = no) = 2/5 = 0.4

Naïve Bayesian Classifier: An Example

X = (*age* = youth, *income* = medium, *student* = yes, *credit_rating* = fair)

P(**X**|C_i) : P(**X**|*buys_computer* = yes) = 0.222 x 0.444 x 0.667 x 0.667 = 0.044

P(**X**|*buys_computer* = no) = 0.6 x 0.4 x 0.2 x 0.4 = 0.019

P(**X**|C_i)xP(C_i) :

P(**X**|*buys_computer* = yes) x P(*buys_computer* = yes) = 0.044 x 0.643 = 0.028

P(**X**|*buys_computer* = no) x P(*buys_computer* = no) = 0.019 x 0.357 = 0.007

Therefore, **X** belongs to class (*buys_computer* = yes) !!

Naïve Bayesian Classifier: Pros and Cons

- Pros
  - The assumption that all features are independent makes naive bayes algorithm very fast compared to complicated algorithms.
    - In some cases, speed is preferred over higher accuracy. Good results obtained in most of the cases
  - It works well with high-dimensional data such as text classification, email spam detection.
- Cons
  - The assumption that all features are independent is not usually the case in real life so it makes naive bayes algorithm less accurate than complicated algorithms.

```
# split dataset
X = dataset.iloc[:,0:8]
y = dataset.iloc[:, 8]
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0, test_size=0.3)
```

```
from sklearn.model_selection import train_test_split
```

```
# hold-back
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,  random_state=7)
```

```
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
```

```
model.fit(X_train,y_train)
```

```
9]: GaussianNB()
```

```
result = model.score(X_test, y_test)
```

```
# Evaluate Model
cm = confusion_matrix(y_test, y_pred)
print (cm)
```

```
[[115  32]
 [ 30  54]]
```

```
print(f1_score(y_test, y_pred))
```

```
0.6352941176470588
```

```
print(accuracy_score(y_test, y_pred))
```

```
0.7316017316017316
```

```
print("Accuracy: %.3f%%" % (result*100.0))
```

```
Accuracy: 73.160%
```

```
# Pregnancies: 3,Glucose: 130, BloodPressure: 100, SkinThickness: 31;
# Insulin: 145, BMI: 25.3, DiabetesPedigreeFunction: 0.325, Age: 45
```

```
new_patient = [[3, 130, 100, 31, 145, 25.5, 0.325, 45]]
```

```
# Predict the new data result
y_pred = model.predict(new_patient)
y_pred
```

```
5]: array([0], dtype=int64)
```

```
new_patient = [[3, 330, 100, 31, 145, 56.5, 0.325, 45]]
```

```
# Predict the new data result
y_pred = model.predict(new_patient)
y_pred
```

```
7]: array([1], dtype=int64)
```