Data Mining

Practical Machine Learning Tools and Techniques

Slides for Chapter 4, Association Rules

of *Data Mining* by I. H. Witten, E. Frank,
M. A. Hall and C. J. Pal

A text document data set. Each document is treated as a “bag” of keywords

doc1:	Student, Teach, School
doc2:	Student, School
doc3:	Teach, School, City, Game
doc4:	Baseball, Basketball
doc5:	Basketball, Player, Spectator
doc6:	Baseball, Coach, Game, Team
doc7:	Basketball, Team, City, Game

A set of transactions

t1:	Beef, Chicken, Milk
t2:	Beef, Cheese
t3:	Cheese, Boots
t4:	Beef, Chicken, Cheese
t5:	Beef, Chicken, Clothes, Cheese, Milk
t6:	Chicken, Clothes, Milk
t7:	Chicken, Milk, Clothes

Mining association rules

- Naïve method for finding association rules:
  - Use separate-and-conquer method
  - Treat every possible combination of attribute values as a separate class
- Two problems:
  - Computational complexity
  - Resulting number of rules (which would have to be pruned on the basis of support and confidence)
- It turns out that we can look for association rules with high support and accuracy directly

Item sets: the basis for finding rules

- Support: number of instances correctly covered by association rule
  - The same as the number of instances covered by *all* tests in the rule (LHS and RHS!)
- *Item*: one test/attribute-value pair
- *Item set* : all items occurring in a rule
- Goal: find only rules that exceed pre-defined support
  - ▯ Do it by finding all item sets with the given minimum support and generating rules from them!

Weather data

Outlook	Temp	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No

Item sets for weather data

One-item sets	Two-item sets	Three-item sets	Four-item sets

Outlook = Sunny (5)	Outlook = Sunny
	Temperature = Hot (2)
		Outlook = Sunny
		Temperature = Hot
		Humidity = High (2)
			Outlook = Sunny
			Temperature = Hot
			Humidity = High
			Play = No (2)

Temperature = Cool (4)	Outlook = Sunny
	Humidity = High (3)
		Outlook = Sunny
		Humidity = High
		Windy = False (2)
			Outlook = Rainy
			Temperature = Mild
			Windy = False
			Play = Yes (2)

…	…	…	…

- Total number of item sets with a minimum support of at least two instances: 12 one-item sets, 47 two-item sets, 39 three-item sets, 6 four-item sets and 0 five-item sets

Generating rules from an item set

- Once all item sets with the required minimum support have been generated, we can turn them into rules
- Example 4-item set with a support of 4 instances:

```
Humidity = Normal, Windy = False, Play = Yes (4)
```

- Seven (2^N-1) potential rules:

```
If Humidity = Normal and Windy = False then Play = Yes     4/4
If Humidity = Normal and Play = Yes then Windy = False     4/6
If Windy = False and Play = Yes then Humidity = Normal     4/6
If Humidity = Normal then Windy = False and Play = Yes     4/7
If Windy = False then Humidity = Normal and Play = Yes     4/8
If Play = Yes then Humidity = Normal and Windy = False     4/9
If True then Humidity = Normal and Windy = False
   and Play = Yes                                          4/12
```

Rules for weather data

- All rules with support > 1 and confidence = 100%:

	Association rule			Sup.	Conf.
1	Humidity=Normal Windy=False	$\Rightarrow$	Play=Yes	4	100%
2	Temperature=Cool	$\Rightarrow$	Humidity=Normal	4	100%
3	Outlook=Overcast	$\Rightarrow$	Play=Yes	4	100%
4	Temperature=Cold Play=Yes	$\Rightarrow$	Humidity=Normal	3	100%
	...	...		...	...
58	Outlook=Sunny Temperature=Hot	$\Rightarrow$	Humidity=High	2	100%

- In total:
  3 rules with support four
  5 with support three
  50 with support two

Example rules from the same item set

- Item set:

```
Temperature = Cool, Humidity = Normal, Windy = False, Play = Yes (2)
```

- Resulting rules (all with 100% confidence):

```
Temperature = Cool, Windy = False ⇒ Humidity = Normal, Play = Yes
Temperature = Cool, Windy = False, Humidity = Normal ⇒ Play = Yes
Temperature = Cool, Windy = False, Play = Yes ⇒ Humidity = Normal
```

- We can establish their confidence due to the following "frequent" item sets:

```
Temperature = Cool, Windy = False                      (2)
Temperature = Cool, Humidity = Normal, Windy = False   (2)
Temperature = Cool, Windy = False, Play = Yes          (2)
```

Generating item sets efficiently

- How can we efficiently find all frequent item sets?
- Finding one-item sets easy
- Idea: use one-item sets to generate two-item sets, two-item sets to generate three-item sets, …
  - If (A B) is a frequent item set, then (A) and (B) have to be frequent item sets as well!
  - In general: if X is a frequent *k*-item set, then all (*k*-1)-item subsets of X are also frequent
  - ⍰ Compute *k*-item sets by merging (*k*-1)-item sets

Example

- Given: five frequent three-item sets

  (A B C), (A B D), (A C D), (A C E), (B C D)

- Lexicographically ordered!
- Candidate four-item sets:

(A B C D)	**OK because of (A C D) (B C D)**
(A C D E)	**Not OK because of (C D E)**

- To establish that these item sets are really frequent, we need to perform a final check by counting instances
- For fast look-up, the (*k* –1)-item sets are stored in a hash table

Algorithm for finding item sets

```
Set $k$ to 1
Find all $k$-item sets with sufficient coverage and store them in hash table #1
While some $k$-item sets with sufficient coverage have been found
    Increment k
    Find all pairs of ($k$-1)-item sets in hash table #($k$-1) that differ only in
    their last item
    Create a $k$-item set for each pair by combining the two ($k$-1)-item sets
    that are paired
    Remove all $k$-item sets containing any ($k$-1)-item sets that are not in the
    #($k$-1)hash table
    Scan the data and remove all remaining $k$-item sets that do not have
    sufficient coverage
    Store the remaining $k$-item sets and their coverage in hash table #$k$,
    sorting items in lexical order
```

Generating rules efficiently

- We are looking for all high-confidence rules
  - Support of antecedent can be obtained from item set hash table
  - But: brute-force method is (2^N-1) for an N-item set
- Better way: building (*c* + 1)-consequent rules from *c*-consequent ones
  - Observation: (*c* + 1)-consequent rule can only hold if all corresponding *c*-consequent rules also hold
- Resulting algorithm similar to procedure for large item sets

Example

- 1-consequent rules:

```
If Outlook = Sunny and Windy = False and Play = No
   then Humidity = High (2/2)
```

```
If Humidity = High and Windy = False and Play = No
   then Outlook = Sunny (2/2)
```

- Corresponding 2-consequent rule:

```
If Windy = False and Play = No
   then Outlook = Sunny and Humidity = High (2/2)
```

- Final check of antecedent against item set hash table is required to check that rule is actually sufficiently accurate

Algorithm for finding association rules

```
Set $n$ to 1
Find all sufficiently accurate n-consequent rules for the $k$-item set and
    store them in hash table #1, computing accuracy using the hash tables
    found for item sets
While some sufficiently accurate $n$-consequent rules have been found
    Increment $n$
    Find all pairs of ($n$-1)-consequent rules in hash table #($n$-1) whose
    consequents differ only in their last item
    Create an $n$-consequent rule for each pair by combining the two ($n$-1)-
    consequent rules that are paired
    Remove all $n$-consequent rules that are insufficiently accurate, computing
    accuracy using the hash tables found for item sets
    Store the remaining $n$-consequent rules and their accuracy in hash table
    #$k$, sorting items for each consequent in lexical order
```

Association rules: discussion

- Above method makes one pass through the data for each different item set size
  - Another possibility: generate (*k*+2)-item sets just after (*k*+1)-item sets have been generated
  - Result: more candidate (*k*+2)-item sets than necessary will be generated but this requires less passes through the data
  - Makes sense if data too large for main memory
- Practical issue: support level for generating a certain minimum number of rules for a particular dataset
  - This can be done by running the whole algorithm multiple times with different minimum support levels
  - Support level is decreased until a sufficient number of rules has been found

Other issues

- Standard ARFF format very inefficient for typical *market basket data*
  - Attributes represent items in a basket and most items are usually missing from any particular basket
  - Data should be represented in sparse format
- Note on terminology: instances are also called *transactions* in the literature on association rule mining
- Confidence is not necessarily the best measure
  - Example: milk occurs in almost every supermarket transaction
  - Other measures have been devised (e.g., lift)
- It is often quite difficult to find interesting patterns in the large number of association rules that can be generated