Overview

In this exercise, we briefly review concepts from itemset mining and association rule mining and then proceed to sequence mining. This part of the exercise is pen-and-paper. Then we move on to the bike rental prediction with ensemble methods. Have fun!

1 Closed and Maximal Itemsets

In the following table, which itemsets are frequent, frequent closed and frequent maximal for the min.support threshold 10.

Itemset	support	frequent	closed	maximal
A	15	x		
B	20	x	x	
C	33	x	x	
D	25	x		
AB	15	x	x	
AC	12	x		
AD	15	x	x	
BC	18	x	x	
BD	5			
CD	25	x	x	
ABC	10	x	x	x
ABD	2			
ACD	12	x	x	x
BCD	3			
ABCD	1			

2 Sequence mining

For the transactional database below:

1. Convert it to a customer-sequence database.
2. Mine the a-projected database and the c-projected database using FREESPAN with $minsup = 2$.
3. Perform the same task using PREFIXSPAN.

Date	CID	Items
01/12	1	a b
01/12	2	bh
01/12	3	i
01/12	4	b c
02/12	1	c
02/12	2	c
02/12	3	b g f
02/12	4	d
03/12	1	a d
03/12	2	a
03/12	3	i
03/12	4	c
04/12	1	b
04/12	2	g
04/12	3	j
04/12	4	f
05/12	1	g
05/12	2	f
05/12	3	c
05/12	4	a
06/12	1	f
06/12	2	a c
06/12	3	b g
06/12	4	b c
07/12	2	h
07/12	4	a
08/12	2	c d
08/12	4	d
09/12	4	c d

Sequence database

CID	Sequence
1	(ab)c(ad)bgf
2	(bh)cagf(ac)h(cd)
3	i(bgf)ijc(bg)
4	(bc)dcfa(bc)ad(cd)

FreeSpan

1. Compute frequencies of individual items (*f-list*): $c:4,\ b:4,\ f:4,\ a:3,\ d:3,\ g:3,\ h:1,\ i:1,\ j:1$

2. Remove infrequent items from the database:

Id	Seq
1	(ab)c(ad)bgf
2	bcagf(ac)(cd)
3	(bgf)c(bg)
4	(bc)dcfa(bc)ad(cd)

3. Mining *c-projection* – sequential patterns only containing c:

1’	c
2’	ccc
3’	c
4’	cccc

Generate candidate 2-sequences and count their support: $cc:2$

(a) *cc-projection* (only transactions that contain cc):

2”	ccc
4”	cccc

Candidate 3-sequences: $ccc : 2$

(b) *ccc-projection* (only transactions that contain ccc):

2”’	ccc
4”’	cccc

Candidate 4-sequences: ~~$cccc : 1$~~

There are no frequent candidate sequence $\Rightarrow$ terminate.

4. Mining *a-projection* – sequential patterns only containing a, and c, b, and f:

1’	(ab)cabf
2’	bcaf(ac)c
4’	(bc)cfa(bc)ac

Generate candidate 2-sequences and count their support:
$aa : 3,\ ac : 3,\ ca : 3,$ ~~$(ac) : 1$~~, $ab : 2,\ ba : 3,$ ~~$(ab) : 1$~~, $af : 2,\ fa : 2,$ ~~$(af) : 0$~~.

For each frequent 2-sequence, mine its projection, e.g.:

(a) *ab-projection*:

1”	(ab)cabf
4”	(bc)cfa(bc)ac

Candidate 3-sequences: $ab\mathbf{a}$, $ab\mathbf{b}$, $ab\mathbf{c}$, $ab\mathbf{f}$, $\mathbf{a}ab$, $\mathbf{b}ab$, $\mathbf{c}ab$, $\mathbf{f}ab$, $\mathbf{a(ab)}$…

(b) *ba-projection* (only transactions that contain ccc):

1”	(ab)cabf
2”	bcaf(ac)c
4”	(bc)cfa(bc)ac

Candidate 3-sequences: note that $ba\mathbf{b}$ can be generated again.

(c) *etc.*

The description above corresponds to a *naïve* version of FreeSpan, which essentially attempts a straightforward translation of FP-Growth to sequential pattern mining. The description aims at illustrating the shortcomings of this translation.
The actual FreeSpan algorithm uses a number of data structures that help to avoid generating redundant candidates. See the original paper (link available on Toledo). This mitigates certain technical, but not conceptual shortcomings of the naïve version.

PrefixSpan

1. Compute frequencies of individual items (*f-list*): $c : 4,\ b : 4,\ f : 4,\ a : 3,\ d : 3,\ g : 3,\ h : 1,\ i : 1,\ j : 1$

2. Remove infrequent items from the database:

Id	Seq
1	(ab)c(ad)bgf
2	bcagf(ac)(cd)
3	(bgf)c(bg)
4	(bc)dcfa(bc)ad(cd)

3. Mining *c-projection*:

1’	(ad)bgf
2’	agf(ac)(cd)
3’	(bg)
4’	(_b)dcfa(bc)ad(cd)

Compute frequencies of individual items (*f-list*): $a : 3,\ b : 3,$ ~~$(\_b) : 1$~~, $c : 3,\ d : 2,\ f : 3,\ g : 3$

Recursively mine projections:

(a) ca-projection (i.e. a-projection of c-projection)

1”	(_d)bgf
2”	gf(ac)(cd)
4”	(bc)ad(cd)

(b) cb-projection

1’	gf
3’	(_g)
4’	(_c)ad(cd)

(c) cc-projection…

(d) cd-projection…

(e) cf-projection…

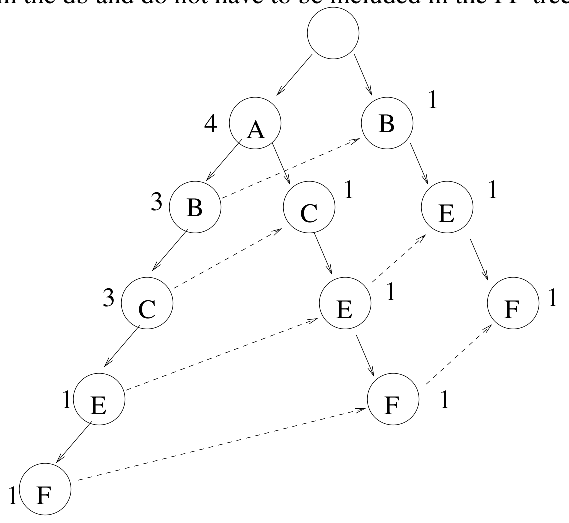
(f) cg-projection…

3 FP Growth

Using the transactional database below, construct its frequent pattern tree (FP tree) using a minimum support threshold, $s = 3$.

TID	Items
1	A, B, C, E, F
2	A, C, D, E, F
3	A, B, C, G, I
4	A, B, C, G
5	B, E, F, H, I, J

Frequent **items**: A(4), B(4), C(4), ~~D(1)~~, E(3), F(3), ~~G(2)~~, ~~H(1)~~, ~~I(2)~~, ~~J(1)~~ (already ordered :)) – infrequent ones can be deleted from the db and do not have to be included in the FP-tree



4 Thought Question

Pattern explosion is a well-known problem in frequent itemset mining. High support thresholds typically result only in few well-known patterns; but for low support thresholds, the number of frequent itemsets can easily be orders of magnitude larger than the number of transactions. Knowledge discovery in such humongous itemset collections is virtually impossible.
What are the causes of pattern explosion? Can you think of a way to solve or at least alleviate this issue?

Primary causes of pattern explosions include:

- *The nature of the problem.*
  If $\{A, B, C\}$ is frequent, then all its 7 supersets are necessarily returned.
- *Functional or statistical relations between attributes.*
  Dependencies between multiple variables might result in a large number of itemsets.
- *Locality of the support constraint.*
  A frequent itemset is always returned, independent of already returned itemsets.

Possible solutions are:

- *Top-k mining*, i.e. only returning k most frequent itemsets.
  Furthermore, frequency can be replaced with another interestingness measure.
- *Condensed representations*, lossless (closed itemsets) or lossy (maximal itemsets).
- *Pattern set mining*, i.e. introducing global constraints on the result sets to eliminate redundancy.
  For example, ensuring that covers of returned itemsets do not overlap. More advanced methods rely on heuristics rooted in probability theory, information theory, compression, etc.