Hands-on Exercise for FPM Module

1. Exploring properties of the dataset accidents_10k.dat. Read more about it here: http://fimi.uantwerpen.be/data/accidents.pdf (http://fimi.uantwerpen.be/data/accidents.pdf)

In [4]:
```
import numpy as np
!head accidents_10k.dat
```

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
30 31
2 5 7 8 9 10 12 13 14 15 16 17 18 20 22 23 24 25 27 28 29 32 33 34 35 36 37 3
8 39
7 10 12 13 14 15 16 17 18 20 25 28 29 30 33 40 41 42 43 44 45 46 47 48 49 50
51 52
1 5 8 10 12 14 15 16 17 18 19 20 21 22 24 25 26 27 28 29 30 31 41 43 46 48 49
51 52 53 54 55 56 57 58 59 60 61
5 8 10 12 14 15 16 17 18 21 22 24 25 26 27 28 29 31 33 36 38 39 41 43 46 56 6
2 63 64 65 66 67 68
7 8 10 12 17 18 21 23 24 26 27 28 29 30 33 34 35 36 38 41 43 47 59 63 66 69 7
0 71 72 73 74 75 76 77 78 79
1 12 14 15 16 17 18 21 22 23 24 25 27 28 29 30 31 35 38 41 43 44 53 56 57 58
59 60 63 66 80 81 82 83 84
10 12 14 15 16 17 18 21 22 24 25 26 27 28 29 30 31 33 39 41 43 44 46 49 59 60
62 63 66 82
1 8 10 12 14 15 16 17 18 21 22 23 24 25 27 29 30 31 38 41 43 53 56 59 61 63 6
6 68 85 86 87 88 89
1 8 12 13 14 15 16 17 18 22 24 25 28 30 38 41 42 43 46 49 60 63 64 66 80 82 8
4 90 91 92 93 94 95
```

Question 1a: . How many items are there in the data?

In [5]:
```
!awk -- '{for (i = 1; i <= NF; i++) wc[$i] += 1}; END {print length(wc)}' acci
dents_10k.dat
```

```
310
```

Answer: There are 310 items.

Question 1b: How many transactions are present in the data?

In [6]:
```
!wc -l accidents_10k.dat
```

```
10000 accidents_10k.dat
```

Answer: There are 10000 transactions.

Question 1c: . What is the length of the smallest transaction?

```
In [7]: ShortestCount = np.inf
        with open('accidents_10k.dat') as data:
            for l in data:
                length = len(l.strip().split())
                if length < ShortestCount:
                    ShortestCount = length
        ShortestCount
```

```
Out[7]: 23
```

Answer: The length of the smallest transaction is 23.

Question 1d: What is the length of the longest transaction?

```
In [8]: LongestCount = 0
        with open('accidents_10k.dat') as data:
            for l in data:
                length = len(l.strip().split())
                if length > LongestCount:
                    LongestCount = length
        LongestCount
```

```
Out[8]: 45
```

Answer: The length of the longest transaction is 45.

Question 1e: What is the size of the search space of frequent itemsets in this data?

```
In [9]: pow(2,310)
```

```
Out[9]: 2085924839766513752338888384931203236916703635113918720651407820138886450957656787131798913024
```

Answer: This is the search space for frequent itemsets because there are 310 items in our dataset. Each item will have a binary value so it is 2^310.

Question 1f: Assume that you work for the deparment of transportation that collected this data. What benefit do you see in using itemset mining approaches on this data?

Answer: The benefit I see for using itemset mining approaches in this data is that these approaches would be effective in finding which variables are frequent when associated with certain outcomes such as car wrecks. The data is also well suited for this approach, because it has the variables split into items where it will be easy to use these approaches.

Question 1g: What type of itemsets (frequent, maximial or closed) would you be interested in discovering this dataset? State your reason.

Answer: Frequent and Maximal itemsets would be the most useful, because they preserve the frequency of the subsets. The closed itemset would give a good summarization of the frequent itemsets from which we could derive the rest of the frequent itemsets. The frequent itemset could be useful, because we can then look at all of the frequent subsets to gather information.

Question 1h: What minsup threshold would you use and why?

```
In [ ]:
```

Answer: The minimum support threshold I would use for this dataset would be around 2000. This could change though, dependy on how subtle of relationships I want to try to identify through frequency analysis. I started with 2000, because this is a high enough frequency to filter out many co-occuring items that may not have a relationship strong enough to be useful. If I did want to go much lower, though, I could also run into issues with complexity. I could increase this support threshold as well if it is more meaningful to capture items that occur very often together.

2. Generating frequent, maximal and closed itemsets using Apriori, ECLAT, and FPGrowth algorihtms from the dataset accidents_10k.dat

Question 2a: Generate frequent itemsets using Apriori, for minsup = 2000, 3000, and 4000. Which of these minsup thresholds results in a maximum number of frequent itemsets? Which of these minsup thresholds results in a least number of frequent itemsets? Provide a rationale for these observations.

In [10]:

```
!./apriori -ts -s-2000 accidents_10k.dat
!./apriori -ts -s-3000 accidents_10k.dat
!./apriori -ts -s-4000 accidents_10k.dat
```

```
./apriori - find frequent item sets with the apriori algorithm
version 6.27 (2017.08.01)        (c) 1996-2017   Christian Borgelt
reading accidents_10k.dat ... [310 item(s), 10000 transaction(s)] done [0.02
s].
filtering, sorting and recoding items ... [49 item(s)] done [0.00s].
sorting and reducing transactions ... [9951/10000 transaction(s)] done [0.00
s].
building transaction tree ... [20250 node(s)] done [0.01s].
checking subsets of size 1 2 3 4 5 6 7 8 9 10 11 12 13 done [18.73s].
writing <null> ... [851034 set(s)] done [0.01s].
./apriori - find frequent item sets with the apriori algorithm
version 6.27 (2017.08.01)        (c) 1996-2017   Christian Borgelt
reading accidents_10k.dat ... [310 item(s), 10000 transaction(s)] done [0.02
s].
filtering, sorting and recoding items ... [38 item(s)] done [0.00s].
sorting and reducing transactions ... [9674/10000 transaction(s)] done [0.00
s].
building transaction tree ... [24741 node(s)] done [0.01s].
checking subsets of size 1 2 3 4 5 6 7 8 9 10 11 12 done [4.39s].
writing <null> ... [133799 set(s)] done [0.01s].
./apriori - find frequent item sets with the apriori algorithm
version 6.27 (2017.08.01)        (c) 1996-2017   Christian Borgelt
reading accidents_10k.dat ... [310 item(s), 10000 transaction(s)] done [0.01
s].
filtering, sorting and recoding items ... [33 item(s)] done [0.01s].
sorting and reducing transactions ... [9381/10000 transaction(s)] done [0.00
s].
building transaction tree ... [22267 node(s)] done [0.00s].
checking subsets of size 1 2 3 4 5 6 7 8 9 10 11 done [1.13s].
writing <null> ... [29501 set(s)] done [0.00s].
```

Answer: The minimum support of 4,000 generated the least amount of subsets. The reason this is the case is because there is a greater restriction upon what qualifies as a frequent subset. A stronger constraint will give you back less subsets for this reason. Less subsets satisfy the criteria.

Question 2b: Using Apriori, compare the execution time for finding frequent itemsets for minsup = 2000, 3000, and 4000. Which of these minsup thresholds takes the least amount of time? Provide a rationale for this observation.

In [11]:

```
import datetime
start = datetime.datetime.now()
!./apriori -ts -s-2000 accidents_10k.dat
end = datetime.datetime.now()
elapsed = end - start
print(elapsed.seconds,"secs ",elapsed.microseconds,"microsecs");

start = datetime.datetime.now()
!./apriori -ts -s-3000 accidents_10k.dat
end = datetime.datetime.now()
elapsed = end - start
print(elapsed.seconds,"secs ",elapsed.microseconds,"microsecs");

start = datetime.datetime.now()
!./apriori -ts -s-4000 accidents_10k.dat
end = datetime.datetime.now()
elapsed = end - start
print(elapsed.seconds,"secs ",elapsed.microseconds,"microsecs");
```

```
./apriori - find frequent item sets with the apriori algorithm
version 6.27 (2017.08.01)        (c) 1996-2017   Christian Borgelt
reading accidents_10k.dat ... [310 item(s), 10000 transaction(s)] done [0.01
s].
filtering, sorting and recoding items ... [49 item(s)] done [0.00s].
sorting and reducing transactions ... [9951/10000 transaction(s)] done [0.01
s].
building transaction tree ... [20250 node(s)] done [0.00s].
checking subsets of size 1 2 3 4 5 6 7 8 9 10 11 12 13 done [18.72s].
writing <null> ... [851034 set(s)] done [0.01s].
19 secs  328666 microsecs
./apriori - find frequent item sets with the apriori algorithm
version 6.27 (2017.08.01)        (c) 1996-2017   Christian Borgelt
reading accidents_10k.dat ... [310 item(s), 10000 transaction(s)] done [0.01
s].
filtering, sorting and recoding items ... [38 item(s)] done [0.00s].
sorting and reducing transactions ... [9674/10000 transaction(s)] done [0.01
s].
building transaction tree ... [24741 node(s)] done [0.00s].
checking subsets of size 1 2 3 4 5 6 7 8 9 10 11 12 done [4.34s].
writing <null> ... [133799 set(s)] done [0.00s].
4 secs  805812 microsecs
./apriori - find frequent item sets with the apriori algorithm
version 6.27 (2017.08.01)        (c) 1996-2017   Christian Borgelt
reading accidents_10k.dat ... [310 item(s), 10000 transaction(s)] done [0.03
s].
filtering, sorting and recoding items ... [33 item(s)] done [0.00s].
sorting and reducing transactions ... [9381/10000 transaction(s)] done [0.00
s].
building transaction tree ... [22267 node(s)] done [0.00s].
checking subsets of size 1 2 3 4 5 6 7 8 9 10 11 done [1.26s].
writing <null> ... [29501 set(s)] done [0.00s].
1 secs  599137 microsecs
```

Answer: The one that took the least time was with minsup 4000. The reason this is the case is because of the pruning procedure. There are less subsets that need to be explored and the alorithm can save time by not exploring these branches.

Question 2c: Using Apriori, find the frequent itemsets for minsup = 2000, 3000, and 4000. Determine the number of itemsets for each size (1 to max length of an itemset). What trends do you see that are common for all three minsup thresholds? What trends do you see that are different? Provide a rationale for these observations.

In [12]:

```
!./apriori -ts -s-2000 accidents_10k.dat
!./apriori -ts -s-3000 accidents_10k.dat
!./apriori -ts -s-4000 accidents_10k.dat
```

```
./apriori - find frequent item sets with the apriori algorithm
version 6.27 (2017.08.01)        (c) 1996-2017   Christian Borgelt
reading accidents_10k.dat ... [310 item(s), 10000 transaction(s)] done [0.03
s].
filtering, sorting and recoding items ... [49 item(s)] done [0.00s].
sorting and reducing transactions ... [9951/10000 transaction(s)] done [0.00
s].
building transaction tree ... [20250 node(s)] done [0.00s].
checking subsets of size 1 2 3 4 5 6 7 8 9 10 11 12 13 done [18.74s].
writing <null> ... [851034 set(s)] done [0.02s].
./apriori - find frequent item sets with the apriori algorithm
version 6.27 (2017.08.01)        (c) 1996-2017   Christian Borgelt
reading accidents_10k.dat ... [310 item(s), 10000 transaction(s)] done [0.01
s].
filtering, sorting and recoding items ... [38 item(s)] done [0.01s].
sorting and reducing transactions ... [9674/10000 transaction(s)] done [0.01
s].
building transaction tree ... [24741 node(s)] done [0.00s].
checking subsets of size 1 2 3 4 5 6 7 8 9 10 11 12 done [4.41s].
writing <null> ... [133799 set(s)] done [0.01s].
./apriori - find frequent item sets with the apriori algorithm
version 6.27 (2017.08.01)        (c) 1996-2017   Christian Borgelt
reading accidents_10k.dat ... [310 item(s), 10000 transaction(s)] done [0.01
s].
filtering, sorting and recoding items ... [33 item(s)] done [0.00s].
sorting and reducing transactions ... [9381/10000 transaction(s)] done [0.01
s].
building transaction tree ... [22267 node(s)] done [0.00s].
checking subsets of size 1 2 3 4 5 6 7 8 9 10 11 done [1.26s].
writing <null> ... [29501 set(s)] done [0.00s].
```

In [13]:

```
./apriori - find frequent item sets with the apriori algorithm
version 6.27 (2017.08.01)        (c) 1996-2017   Christian Borgelt
reading accidents_10k.dat ... [310 item(s), 10000 transaction(s)] done [0.02
s].
filtering, sorting and recoding items ... [49 item(s)] done [0.00s].
sorting and reducing transactions ... [9951/10000 transaction(s)] done [0.00
s].
building transaction tree ... [20250 node(s)] done [0.01s].
checking subsets of size 1 2 3 4 5 6 7 8 9 10 11 12 13 done [18.77s].
writing <null> ... [851034 set(s)] done [0.01s].
all: 851034
  0: 0
  1: 49
  2: 705
  3: 5285
  4: 23745
  5: 69647
  6: 139628
  7: 195730
  8: 193299
  9: 133819
 10: 63937
 11: 20497
 12: 4189
 13: 483
 14: 21
./apriori - find frequent item sets with the apriori algorithm
version 6.27 (2017.08.01)        (c) 1996-2017   Christian Borgelt
reading accidents_10k.dat ... [310 item(s), 10000 transaction(s)] done [0.02
s].
filtering, sorting and recoding items ... [38 item(s)] done [0.00s].
sorting and reducing transactions ... [9674/10000 transaction(s)] done [0.00
s].
building transaction tree ... [24741 node(s)] done [0.01s].
checking subsets of size 1 2 3 4 5 6 7 8 9 10 11 12 done [4.35s].
writing <null> ... [133799 set(s)] done [0.00s].
all: 133799
  0: 0
  1: 38
  2: 468
  3: 2830
  4: 9887
  5: 21779
  6: 31964
  7: 32020
  8: 21862
  9: 9839
 10: 2705
 11: 387
 12: 20
./apriori - find frequent item sets with the apriori algorithm
version 6.27 (2017.08.01)        (c) 1996-2017   Christian Borgelt
reading accidents_10k.dat ... [310 item(s), 10000 transaction(s)] done [0.02
s].
filtering, sorting and recoding items ... [33 item(s)] done [0.00s].
sorting and reducing transactions ... [9381/10000 transaction(s)] done [0.00
s].
```

```
building transaction tree ... [22267 node(s)] done [0.01s].
checking subsets of size 1 2 3 4 5 6 7 8 9 10 11 done [1.33s].
writing <null> ... [29501 set(s)] done [0.00s].
all: 29501
  0: 0
  1: 33
  2: 319
  3: 1492
  4: 4043
  5: 6926
  6: 7751
  7: 5626
  8: 2546
  9: 668
 10: 91
 11: 6
```

```
In [ ]:
```

Answer:

Question 2d: Using Apriori with minsup=2000, compare the number of frequent, maximal, and closed itemsets. Which is the largest set and which is the smallest set? Provide a rationale for these observations.

In [14]:

```
!./apriori -ts -s-2000 accidents_10k.dat
!./apriori -tc -s-2000 accidents_10k.dat
!./apriori -tm -s-2000 accidents_10k.dat
```

```
./apriori - find frequent item sets with the apriori algorithm
version 6.27 (2017.08.01)        (c) 1996-2017   Christian Borgelt
reading accidents_10k.dat ... [310 item(s), 10000 transaction(s)] done [0.02
s].
filtering, sorting and recoding items ... [49 item(s)] done [0.00s].
sorting and reducing transactions ... [9951/10000 transaction(s)] done [0.00
s].
building transaction tree ... [20250 node(s)] done [0.00s].
checking subsets of size 1 2 3 4 5 6 7 8 9 10 11 12 13 done [18.78s].
writing <null> ... [851034 set(s)] done [0.01s].
./apriori - find frequent item sets with the apriori algorithm
version 6.27 (2017.08.01)        (c) 1996-2017   Christian Borgelt
reading accidents_10k.dat ... [310 item(s), 10000 transaction(s)] done [0.01
s].
filtering, sorting and recoding items ... [49 item(s)] done [0.00s].
sorting and reducing transactions ... [9951/10000 transaction(s)] done [0.01
s].
building transaction tree ... [20250 node(s)] done [0.00s].
checking subsets of size 1 2 3 4 5 6 7 8 9 10 11 12 13 14 done [28.50s].
filtering for closed item sets ... done [0.46s].
writing <null> ... [519902 set(s)] done [0.01s].
./apriori - find frequent item sets with the apriori algorithm
version 6.27 (2017.08.01)        (c) 1996-2017   Christian Borgelt
reading accidents_10k.dat ... [310 item(s), 10000 transaction(s)] done [0.02
s].
filtering, sorting and recoding items ... [49 item(s)] done [0.00s].
sorting and reducing transactions ... [9951/10000 transaction(s)] done [0.01
s].
building transaction tree ... [20250 node(s)] done [0.00s].
checking subsets of size 1 2 3 4 5 6 7 8 9 10 11 12 13 14 done [28.38s].
filtering for maximal item sets ... done [0.03s].
writing <null> ... [12330 set(s)] done [0.01s].
```

Answer: The largest itemset is the frequent itemset and the smallest itemset is the maximal itemset. The reason the maximal itemset is much smaller is because it is a summarization of the frequent itemset, though, you lose information about frequency. From it, you can derive frequent itemsets but not information about their frequency.

Question 2e: For a minsup = 2000, compare the execution time for Apriori, ECLAT and FPGrowth. Which of these algorithms took the least amount of time. Provide a rationale for this observation.

In [15]:

```
start = datetime.datetime.now()
!./apriori -ts -s-2000 accidents_10k.dat
end = datetime.datetime.now()
elapsed = end - start
print(elapsed.seconds,"secs ",elapsed.microseconds,"microsecs");

start = datetime.datetime.now()
!./eclat -ts -s-2000 accidents_10k.dat
end = datetime.datetime.now()
elapsed = end - start
print(elapsed.seconds,"secs ",elapsed.microseconds,"microsecs");

start = datetime.datetime.now()
!./fpgrowth -ts -s-2000 accidents_10k.dat
end = datetime.datetime.now()
elapsed = end - start
print(elapsed.seconds,"secs ",elapsed.microseconds,"microsecs");
```

```
./apriori - find frequent item sets with the apriori algorithm
version 6.27 (2017.08.01)        (c) 1996-2017   Christian Borgelt
reading accidents_10k.dat ... [310 item(s), 10000 transaction(s)] done [0.02
s].
filtering, sorting and recoding items ... [49 item(s)] done [0.00s].
sorting and reducing transactions ... [9951/10000 transaction(s)] done [0.00
s].
building transaction tree ... [20250 node(s)] done [0.00s].
checking subsets of size 1 2 3 4 5 6 7 8 9 10 11 12 13 done [18.75s].
writing <null> ... [851034 set(s)] done [0.01s].
19 secs  366237 microsecs
./eclat - find frequent item sets with the eclat algorithm
version 5.20 (2017.05.30)        (c) 2002-2017   Christian Borgelt
reading accidents_10k.dat ... [310 item(s), 10000 transaction(s)] done [0.01
s].
filtering, sorting and recoding items ... [49 item(s)] done [0.00s].
sorting and reducing transactions ... [9951/10000 transaction(s)] done [0.01
s].
writing <null> ... [851034 set(s)] done [0.27s].
0 secs  585580 microsecs
./fpgrowth - find frequent item sets with the fpgrowth algorithm
version 6.17 (2017.05.30)        (c) 2004-2017   Christian Borgelt
reading accidents_10k.dat ... [310 item(s), 10000 transaction(s)] done [0.01
s].
filtering, sorting and recoding items ... [49 item(s)] done [0.00s].
sorting and reducing transactions ... [9951/10000 transaction(s)] done [0.01
s].
writing <null> ... [851034 set(s)] done [0.08s].
0 secs  406896 microsecs
```

Answer: The algorithm which took the least amount of time was the FPGrowth algorithm. This is because the computational complexity is lower for this algorithm than for the other two. The reason the second two algorithms were much more efficient if because they carry information about frequency with them as they pick out frequent subsets. The Apriori algorithm requires that you search the database for each itemset while the other two do not.

Question 2f: For a minsup = 4000, compare the execution time for Apriori, ECLAT and FPGrowth. Which of these algorithms took the least amount of time. Provide a rationale for this observation.

In [16]:

```
start = datetime.datetime.now()
!./apriori -ts -s-4000 accidents_10k.dat
end = datetime.datetime.now()
elapsed = end - start
print(elapsed.seconds,"secs ",elapsed.microseconds,"microsecs");

start = datetime.datetime.now()
!./eclat -ts -s-4000 accidents_10k.dat
end = datetime.datetime.now()
elapsed = end - start
print(elapsed.seconds,"secs ",elapsed.microseconds,"microsecs");

start = datetime.datetime.now()
!./fpgrowth -ts -s-4000 accidents_10k.dat
end = datetime.datetime.now()
elapsed = end - start
print(elapsed.seconds,"secs ",elapsed.microseconds,"microsecs");
```

```
./apriori - find frequent item sets with the apriori algorithm
version 6.27 (2017.08.01)        (c) 1996-2017   Christian Borgelt
reading accidents_10k.dat ... [310 item(s), 10000 transaction(s)] done [0.01
s].
filtering, sorting and recoding items ... [33 item(s)] done [0.00s].
sorting and reducing transactions ... [9381/10000 transaction(s)] done [0.01
s].
building transaction tree ... [22267 node(s)] done [0.00s].
checking subsets of size 1 2 3 4 5 6 7 8 9 10 11 done [1.27s].
writing <null> ... [29501 set(s)] done [0.00s].
1 secs  596511 microsecs
./eclat - find frequent item sets with the eclat algorithm
version 5.20 (2017.05.30)        (c) 2002-2017   Christian Borgelt
reading accidents_10k.dat ... [310 item(s), 10000 transaction(s)] done [0.01
s].
filtering, sorting and recoding items ... [33 item(s)] done [0.00s].
sorting and reducing transactions ... [9381/10000 transaction(s)] done [0.01
s].
writing <null> ... [29501 set(s)] done [0.03s].
0 secs  315336 microsecs
./fpgrowth - find frequent item sets with the fpgrowth algorithm
version 6.17 (2017.05.30)        (c) 2004-2017   Christian Borgelt
reading accidents_10k.dat ... [310 item(s), 10000 transaction(s)] done [0.01
s].
filtering, sorting and recoding items ... [33 item(s)] done [0.00s].
sorting and reducing transactions ... [9381/10000 transaction(s)] done [0.01
s].
writing <null> ... [29501 set(s)] done [0.02s].
0 secs  316969 microsecs
```

Answer: In this case, the fastest algorithms were FPGrowth and ECLAT. They were nearly tied (FPGrowth is slightly better). The reason that they are tied this time is that the computational complexity of FPGrowth is slightly better, but the complexity of the search was not high enough for it to make much a difference between the two. These two algorithms performed much better than appriori for the same reason as before, but noteably, the time reduced drastically. Apriori suffers much more from having a lower support threshold.

Question 2g: For a minsup = 6000, compare the execution time for Apriori, ECLAT and FPGrowth. Which of these algorithms took the least amount of time. Provide a rationale for this observation.

In [17]:
```
start = datetime.datetime.now()
!./apriori -ts -s-6000 accidents_10k.dat
end = datetime.datetime.now()
elapsed = end - start
print(elapsed.seconds,"secs ",elapsed.microseconds,"microsecs");

start = datetime.datetime.now()
!./eclat -ts -s-6000 accidents_10k.dat
end = datetime.datetime.now()
elapsed = end - start
print(elapsed.seconds,"secs ",elapsed.microseconds,"microsecs");

start = datetime.datetime.now()
!./fpgrowth -ts -s-6000 accidents_10k.dat
end = datetime.datetime.now()
elapsed = end - start
print(elapsed.seconds,"secs ",elapsed.microseconds,"microsecs");
```

```
./apriori - find frequent item sets with the apriori algorithm
version 6.27 (2017.08.01)        (c) 1996-2017   Christian Borgelt
reading accidents_10k.dat ... [310 item(s), 10000 transaction(s)] done [0.01
s].
filtering, sorting and recoding items ... [20 item(s)] done [0.00s].
sorting and reducing transactions ... [3216/10000 transaction(s)] done [0.01
s].
building transaction tree ... [6478 node(s)] done [0.00s].
checking subsets of size 1 2 3 4 5 6 7 8 done [0.03s].
writing <null> ... [2254 set(s)] done [0.00s].
0 secs  323938 microsecs
./eclat - find frequent item sets with the eclat algorithm
version 5.20 (2017.05.30)        (c) 2002-2017   Christian Borgelt
reading accidents_10k.dat ... [310 item(s), 10000 transaction(s)] done [0.01
s].
filtering, sorting and recoding items ... [20 item(s)] done [0.00s].
sorting and reducing transactions ... [3216/10000 transaction(s)] done [0.01
s].
writing <null> ... [2254 set(s)] done [0.00s].
0 secs  301202 microsecs
./fpgrowth - find frequent item sets with the fpgrowth algorithm
version 6.17 (2017.05.30)        (c) 2004-2017   Christian Borgelt
reading accidents_10k.dat ... [310 item(s), 10000 transaction(s)] done [0.03
s].
filtering, sorting and recoding items ... [20 item(s)] done [0.00s].
sorting and reducing transactions ... [3216/10000 transaction(s)] done [0.00
s].
writing <null> ... [2254 set(s)] done [0.00s].
0 secs  298298 microsecs
```

Answer: As before, FPGrowth and ECLAT were nearly tied in their times (times have basically converged). They are both better than apriori still, but apriori has now nearly converged to the quick times of FPGrowth and ECLAT. Apriori suffers much more and more quickly than the other two algorithms, because of its need to search the database over and over again to calculate the support. But now, with less calculation for this purpose necessary, it is less hindered in comparison to the other two algorithms.

Question 2h: Fill the following table based on execution times computed in **2e**, **2f**, and **2g**. State your observations on the relative computational efficiency at different support thresholds. Based on your knowledge of these algorithms, provide the reasons behind your observations.

Algorithm	minsup=2000	minsup=4000	minsup=6000
Apriori	23 secs 38407 microsecs	1 secs 636405 microsecs	0 secs 368071 microsecs
Eclat	0 secs 599343 microsecs	0 secs 345872 microsecs	0 secs 302940 microsecs
FPGrowth	0 secs 443076 microsecs	0 secs 314695 microsecs	0 secs 300142 microsecs

Answer: Apriori is the algorithm with the worst computational complexity. It suffers very significantly with a low support threshold because of the database search it must do for each itemset. ECLAT has a slightly greater computational complexity than FPGrowth and this is reflected most clearly at the low support threshold where it is slightly slower. As the minimum support increases, these algorithms converge in their time take to be completed. This is because the effects of the complexity explosion in Apriori, and to a lesser exten ECLAT, have not come to fruition due to the time spent searching the database being less significant than the other parts of the algorithm as a result of having relatively few itemsets to compute frequency.

3. Discovering frequent subsequences and substrings

Assume that roads in a Cincinnati are assigned numbers. Participants are enrolled in a transportation study and for every trip they make using their car, the sequence of roads taken are recorded. Trips that involves freeways are excluded. This data is in the file road_seq_data.dat.

Question 3a: What 'type' of sequence mining will you perform to determine frequently taken 'paths'? Paths are sequences of roads traveresed consecutively in the same order.

Answer: I would use substring mining, because substring mining preserves the consecutive order of the items contained in the substring

Question 3b: How many sequences are there in this sequence database?

```
In [18]: !wc -l road_seq_data.dat
```

```
1000 road_seq_data.dat
```

Answer: 1000

Question 3c: What is the size of the alphabet in this sequence database?

In [19]:

```
!awk -- '{for (i = 1; i <= NF; i++) wc[$i] += 1}; END {print length(wc)}' road_seq_data.dat
```

```
1283
```

Answer: 1283

Question 3d: What are the total number of possible subsequences of length 2 in this dataset?

In [20]:

```
import math
math.factorial(1283)/math.factorial(1283-2)
```

Out[20]:

```
1644806.0
```

Answer: There are this many possible subsequences, because we look at these subsequences as combinations. We check how many ordered combinations are possible within this data set for choosing two values.

Question 3e: What are the total number of possible substrings of length 2 in this dataset?

Answer: The total number of possible substrings will be 1282. This is because substrings must be consecutive. The total number is one less, because you will not be able to make a substring starting with the last item in the sequences. The formula for the number of substrings will be n - (Substring Length - 1) for substrings of any size.

Question 3f: Discover frequent **subsequences** with minsup = 10 and report the number of subsequences discovered.

In [ ]:

```

```

In [22]:

```
!./prefixspan -min_sup 10 road_seq_data.dat | sed -n 'p;n' > subseq_road_seq_data_minsup_10
```

```
PrefixSpan version 1.00 - Sequential Pattern Miner
Written by Yasuo Tabei
```

In [23]:

```
!wc -l subseq_road_seq_data_minsup_10
```

```
4589 subseq_road_seq_data_minsup_10
```

Answer: There are 4589 frequent subsequences with minsup 10.

Question 3g: Discover frequent **substrings** with minsup = 10 and report the number of substrings discovered.

In [24]:

```
!./seqwog -ts -s-10 road_seq_data.dat | sed -n 'p;n' > substr_road_seq_data_minsup_10
```

```
./seqwog - find frequent sequences without gaps
version 3.16 (2016.10.15)        (c) 2010-2016   Christian Borgelt
reading road_seq_data.dat ... [1283 item(s), 1000 transaction(s)] done [0.00s].
recoding items ... [1283 item(s)] done [0.00s].
reducing and triming transactions ... [844/1000 transaction(s)] done [0.00s].
writing <null> ... [613 sequence(s)] done [0.00s].
```

In [25]:

```
!wc -l substr_road_seq_data_minsup_10
```

```
0 substr_road_seq_data_minsup_10
```

Answer: There are 0 Substrings of length 10.

Question 3h: Explain the difference in the number of frequent subsequences and substrings found in **3f** and **3g** above.

Answer: The difference in the number of frequent subsequences and substrings is the result of how they substrings and subsequences are classified. Subsequences allow you to pull items accross the span of a transaction and piece them together while substrings have an additional constraint. The items must all be in consecutive order as well for it to be considered a substring.

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: