Dylan Zucker & Dempsey Wade
Data Mining
March 1, 2019

Exercise 1: The Apriori Algorithm

Part a:

TID	**items_bought**
T100	{M, O, N, K, E, Y}
T200	{D, O, N, K, E, Y}
T300	{M, A, K, E}
T400	{M, U, C, K, Y}
T500	{C, O, K, I, E}

1-itemsets	**sup**
{M}	.6
{O}	.6
{N}	Does not meet min_sup
{K}	1
{E}	.8
{Y}	.6
{D}	Does not meet min_sup
{A}	Does not meet min_sup
{U}	Does not meet min_sup
{C}	Does not meet min_sup
{I}	Does not meet min_sup

2-itemsets	**sup**
{M, O}	Does not meet min_sup
{M, K}	.6
{M, E}	Does not meet min_sup
{M, Y}	Does not meet min_sup
{O, K}	.6
{O, E}	.6
{O, Y}	Does not meet min_sup
{K, E}	.8
{K, Y}	.6
{E, Y}	Does not meet min_sup

3-itemsets	**sup**
{O, K, E}	.6

// TODO: CHECK THIS IS THE ONLY ITEMSET LEFT

Part b:

An itemset is frequent if its support is greater than or equal to the minimum support threshold. An itemset is closed if none of the itemset's immediate supersets have the same support as the itemset.

Closed frequent itemsets from part a:
{M, K}
{K, Y}
{K, E} // **TODO: Does this count? Does one count if its superset has a greater support what does immediate mean in definition**
{O, K, E}

Part c:

An itemset is a maximal frequent itemset if it is frequent and if there is no superset of the itemset that is also frequent.
Max-frequent-itemsets from part a:
{M, K}
{K, Y}
{O, K, E}

Part d

Exercise 2: The FP-Growth Algorithm

a) Perform the first step and create create the initial F-list, which creates the list of 1-itemsets that are frequent, and sort the list in descending order of support. (You can use your answer above, since you already identified frequent 1-itemsets.)

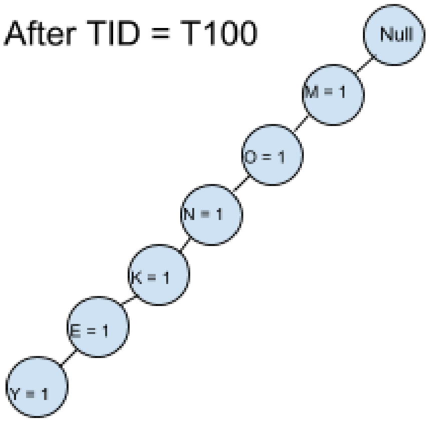
1-itemsets	**sup**
{M}	.6
{O}	.6
{K}	1
{E}	.8
{Y}	.6

b) Create the initial FP-tree *and* corresponding header of items / support count / node links. Label this tree structure as **tree{}**.

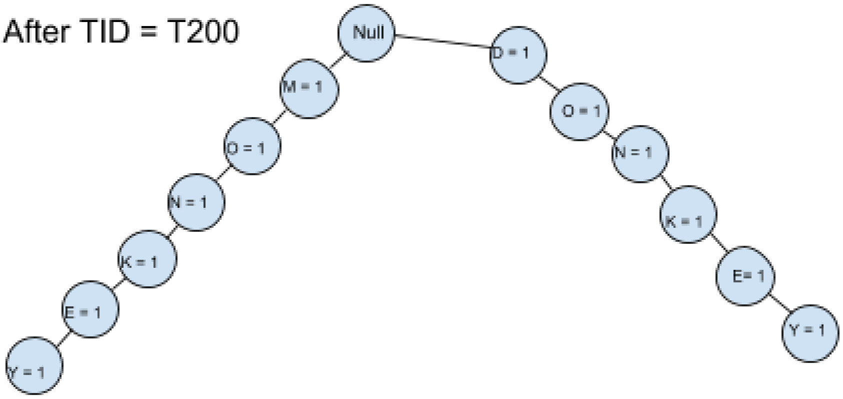
Steps illustrated in next subproblem.

c) Execute the **FP_growth** algorithm. Start with **FP_growth(tree{}, null)**, and clearly indicate each step of the algorithm starting with the least frequent item. Indicate the recursive call with parameters to show what suffix and tree is currently being applied. Indicate when a frequent pattern is generated, and circle it. As you work through the algorithm, clearly show each conditional pattern base (CPB) and corresponding conditional FP-tree constructed with the CPB. Label each conditional FP-tree as tree *suffix*, replacing *suffix* with the actual suffix being applied to generate the tree for the recursive call. Your resulting frequent patterns should be identical to the previous exercise.

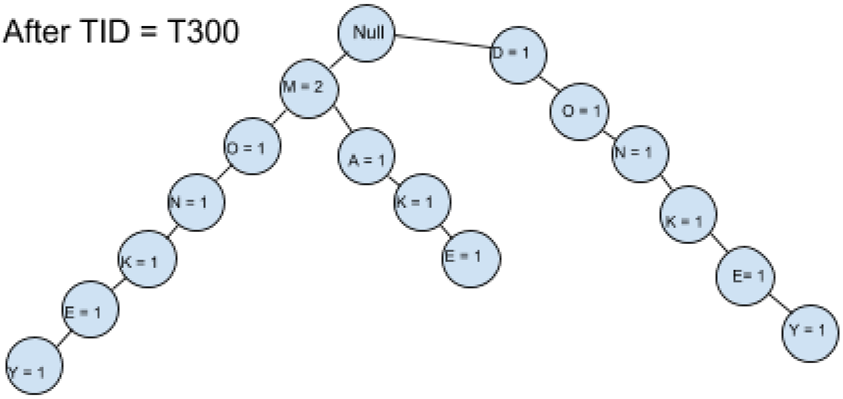
After TID = T100

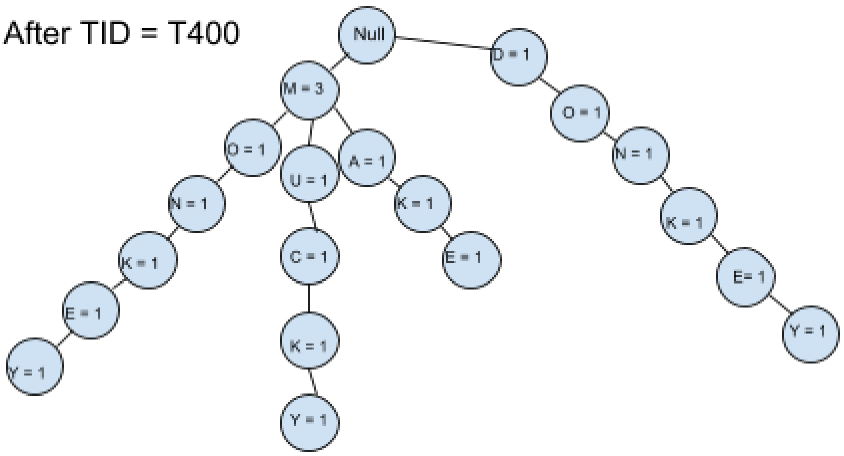


After TID = T200

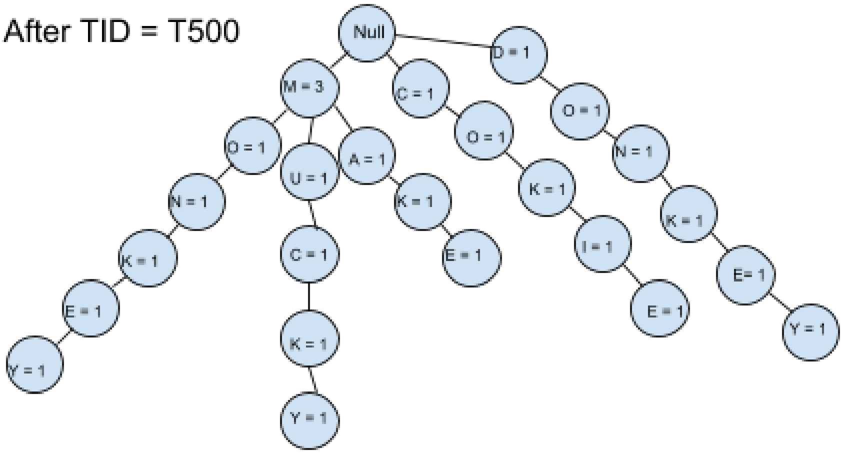


After TID = T300





After TID = T400



After TID = T500

d) Compare and contrast the computational requirements (time and space) for both Apriori and FP-growth working on this exercise.

Exercise 3: The Eclat algorithm

a) Convert the dataset in Exercise 1 to a vertical data format

M	{T100, T300, T400}
O	{T100, T200, T500}

N	{T100, T200}
K	{T100, T200, T300, T400, T500}
E	{T100, T200, T300, T500}
Y	{T100, T200, T400}
~~D~~	~~{T200}~~
~~A~~	~~{T300}~~
~~U~~	~~{T400}~~
~~C~~	~~{T500}~~
~~I~~	~~{T500}~~

b) Find the frequent itemset using the Eclat algorithm

~~M, O~~	~~{T100}~~
~~M, N~~	~~{T100}~~
M, K	{T100, T300, T400}
~~M, E~~	~~{T100, T300}~~
~~M, Y~~	~~{T100, T400}~~
~~O, N~~	~~{T100, T200}~~
O, K	{T100, T200, T500}
O, E	{T100, T200, T500}
~~O, Y~~	~~{T100, T200}~~
~~N, K~~	~~{T100, T200}~~
~~N, E~~	~~{T100, T200}~~
~~N, Y~~	~~{T100, T200}~~
K, E	{T100, T200, T300, T500}

K, Y	{T100, T200, T400}
~~E, Y~~	~~{T100, T200}~~

~~M, K, O~~	~~{T100}~~
~~M, K, E~~	~~{T100, T300}~~
~~M, K, Y~~	~~{T100, T400}~~
O, K, E	{T100, T200, T500}
~~O, K, Y~~	~~{T100, T200}~~

Exercise 4: Correlation

	A	NOT A
B	65	40
NOT B	35	10

```
import numpy as np
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules

d = {'A': [65, 35], 'NOTA': [40, 10]}
df = pd.DataFrame(data=d, index = ['B', 'NOTB'])
print(df)
te = TransactionEncoder()
te_ary = te.fit(df).transform(df)
df = pd.DataFrame(te_ary, columns=te.columns_)
print(df)
x = apriori(df, min_support=0.005)
x
```

a) Let *min_sup* = 0.4, and *min_conf* = 0.6. Compute support and confidence for the rule **AàB**. Is this a strong
Rule?

	A	NOT A	Sum
B	65	40	105
NOT B	35	10	45
Sum	100	50	150

Support:
P(A) = 100/150 = 0.67
P(B) = 105/150 = 0.70
~~P(NOT A) = 50/150 = 0.33~~
~~P(NOT B) = 45/150 = 0.30~~
P(A AND B) = 65/150 = 0.43
~~P(A NOT B) = 35/150 = 0.23~~
~~P(B NOT A) = 40/150 = 0.27~~

Confidence:
P(A | B) = 0.67/0.70 = 0.96
P~~(NOT A | B) = 0.33/0.70 = 0.47~~
~~P(NOT B | A) = 0.30/0.67 = 0.45~~
P(NOT B | NOT A) = 0.30/0.33 = 0.91

b) What does the *lift* measure tell us? Compute **lift(A,B)**. What does this suggest about the occurrence of A and B? What does it suggest about the rule?

Lift(A,B) = Support (A AND B) / (Support(A) * Support(B))

= 0.43 / (0.67*0.70) = 0.43 / 0.469 = 0.917

Since Lift(A,B) ~= 1, we cannot say that there is a strong rule about A or B. We conclude that if A, then Not B is most likely.

c) Compute the expected values for each observed value above, showing your results in a table.

Observed:

	A	NOT A	Sum
B	65	40	105
NOT B	35	10	45

Sum	100	50	150

Expected:

	A	NOT A	Sum
B	70	35	105
NOT B	30	15	45
Sum	100	50	150

d) Compute the **X_2 correlation coefficient** using the table above and your expected values you computed in the previous question. Does the value imply dependency among A and B?

cor(A,B) = (sup(A,B) - Sup(A) * SuP(B)) / sqrt(Sup(A) * (1-Sup(A))*Sup(B)*(1-Sup(B)))
= (0.43 - (0.67*0.70)) / sqrt((0.67)*(1-0.67)*(0.70)*(1-0.70)
= -0.039 / sqrt(0.464) = -0.039 / 0.21547 = -0.18

X^2 = SUM((Observed - Expected)^2 / Expected)
AUB= 25/70
B U NOTA= 25/35
NOTB U A= 25/30
NOTA U NOTB= 25/15
SUM(X) = 3.57
Chi Squared = 3.57

When the expected value is > the observed value, the items are not independent.
From this we can conclude that A and B are dependent, as well as NOT A and NOT B.

e) Consider the rule **A**à**NOT B**. What is the *support*, *confidence* and *lift* for this rule?

P(A) = 100/150 = 0.67
P(NOT B) = 45/150 = 0.30

Support: P(A NOT B) = 35/150 = 0.23
Confidence: P(A | NOT B) = 0.67/0.30 = 2.23
Lift: 0.23 / (0.67*0.30) = 1.15

f) What is the confidence and lift of the rule **NOT B**à**A** ? You should notice there is an imbalance between your answer here and the previous question. Which rule is stronger? Why?

P(A) = 100/150 = 0.67
P(NOT B) = 45/150 = 0.30

Confidence: P(NOT B | A) = 0.30/0.67 = 0.45

Lift: 0.45 / (0.67 * 0.30) = 2.25

The rule **NOT BàA** shows a positive correclation, same as **AàNOT B**, but is much stronger. In this example, the higher lift value tells us that the rule is stronger.

g) Compute the *Kulczynski measure* for the items **A** and **NOT B**.

P(NOT B | A) = 0.30/0.67 = 0.45

P(A | NOT B) = 0.67/0.30 = 2.23

Kulc(A, NOT B) = ½ * (P(A|NOT B) + P(NOT B|A))

= ½ * (2.23 + 0.45) = 1.34

h) Compute the *imbalance ratio* (IR) on **A** and **NOT B**. What do these results say? Does the result confirm your observations on questions e) and f) above?

IR(A, NOT B) = |Sup(A) - Sup(NOT B)| / (Sup(A) + Sup(NOT B) - Sup(A AND NOTB))

= |0.67 - 0.3| / (0.67 + 0.3 - 0.23)

= 0.37 / 0.74 = 0.5

Since our result is directly in between 0 and 1, we cannot say whether there is an imbalance is not.

Exercise 5: Distributed mining

Complete exercise 6.9 in the book.

(HINT: Think about our in-class discussion on methods that reduce database scans. One of those methods focused on partitioning a transaction dataset...)

6.9- Suppose that a large store has a transactional database that is *distributed* among four locations. Transactions in each component database have the same format, names Tj:{i1,...im}, where Tj is a transaction identifier, and ik, (1 <= k <= m) is the identifier of an item purchased in the transaction. Propose an efficient algorithm to mine global association rules. You may present your algorithm in the form of an outline. Your algorithm should not require shipping all the data to one site and should not cause excessive network communication.

Use FP-Tree

Step1: Construct a tree for each database. This requires no initial network communication and builds each tree independently.

Step2: Create a new overarching tree on the 4 existing trees.

- Add the first tree to the super tree.
- Sort through the second tree one node at a time and add it to the first tree.
    - Same FP-Tree algorithm:

- If the super tree contains the node trying to be inserted:
    - Increase the count of that node and check the children nodes
    - If a child node does not exist that is the next node to be added, create a new branch

Step3: Repeat Step2 for the remaining 2 trees.