6.6 A database has five transactions. Let $min\_sup = 60\%$ and $min\_conf = 80\%$.

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, O, K, I, E}

(a) Find all frequent itemsets using Apriori and FP-growth, respectively. Compare the efficiency of the two mining processes.

(b) List all the *strong* association rules (with support s and confidence c) matching the following metarule, where X is a variable representing customers, and $item_i$ denotes variables representing items (e.g., "A," "B,"):

$$\forall x \in transaction,\ buys(X, item_1) \wedge buys(X, item_2) \Rightarrow buys(X, item_3) \quad [s, c]$$

- A

Apriori:

C1 =

m	3
o	3
n	2
k	5
e	4
y	3
d	1
a	1
u	1
c	2
i	1

L1 =

m	3
o	3
k	5
e	4
y	3

C2 =

mo	1
mk	3
me	2
my	2
ok	3
oe	3
oy	2
ke	4
ky	3
ey	2

L2 =

mk	3
ok	3
oe	3
ke	4
ky	3

C3=

oke	3
key	2

L3=

oke	3

  -

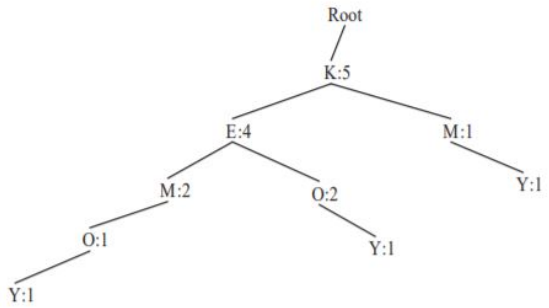


Figure 5.2: FP-tree for Exercise 5.3.

item	conditional pattern base	conditional tree	frequent pattern
y	{ {k,e,m,o:1}, {k,e,o:1}, {k,m:1} }	k:3	{k,y:3}
o	{ {k,e,m:1}, {k,e:2}	k:3,e:3	{k,o:3}, {e,o:3}, {k,e,o:3}
m	{ {k,e:2}, {k:1} }	k:3	{k,m: 3}
e	{ {k:4} }	k:4	{ k,e:4 }

Efficiency comparison: Apriori has to do multiple scans of the database while FP-growth builds the FP-Tree with a single scan. Candidate generation in Apriori is expensive (owing to the self-join), while FP-growth does not generate any candidates.

- B
  - k,o → e [0.6, 1]
  - e,o → k [0.6, 1]

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, namely $T_j : \{i_1, \ldots, i_m\}$, where T_j is a transaction identifier, and i_k $(1 \leq k \leq m)$ 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 overhead.

An algorithm to mine global association rules is as follows:

- Find the local frequent item sets in each store. Let CF be the union of all the local frequent items in the four stores.
- In each store, find the local (absolute) support for each itemset in CF
- Now to determine the global support for each itemset in CF we can sum up the local support for each itemset across the four stores, thus giving us the global support for an item and lets us determine if it passes the support threshold.
- Derive strong association rules from the global frequent item sets.

6.10 Suppose that frequent itemsets are saved for a large transactional database, *DB*. Discuss how to efficiently mine the (global) association rules under the same minimum support threshold, if a set of new transactions, denoted as ΔDB, is *(incrementally) added in*?

Treat ΔDB and DB as two partitions

- For item sets that are frequent in DB, scan ΔDB once and add their counts to see if they are still frequent in the updated database.
- For item sets that are frequent in ΔDB but not in DB, scan DB once to add their counts to see if they are frequent in the updated DB.

8.7 The following table consists of training data from an employee database. The data have been generalized. For example, "31 ... 35" for *age* represents the age range of 31 to 35. For a given row entry, *count* represents the number of data tuples having the values for *department, status, age*, and *salary* given in that row.

department	*status*	*age*	*salary*	*count*
sales	senior	31...35	46K...50K	30
sales	junior	26...30	26K...30K	40
sales	junior	31...35	31K...35K	40
systems	junior	21...25	46K...50K	20
systems	senior	31...35	66K...70K	5
systems	junior	26...30	46K...50K	3
systems	senior	41...45	66K...70K	3
marketing	senior	36...40	46K...50K	10
marketing	junior	31...35	41K...45K	4
secretary	senior	46...50	36K...40K	4
secretary	junior	26...30	26K...30K	6

Let *status* be the class label attribute.

(a) How would you modify the basic decision tree algorithm to take into consideration the *count* of each generalized data tuple (i.e., of each row entry)?

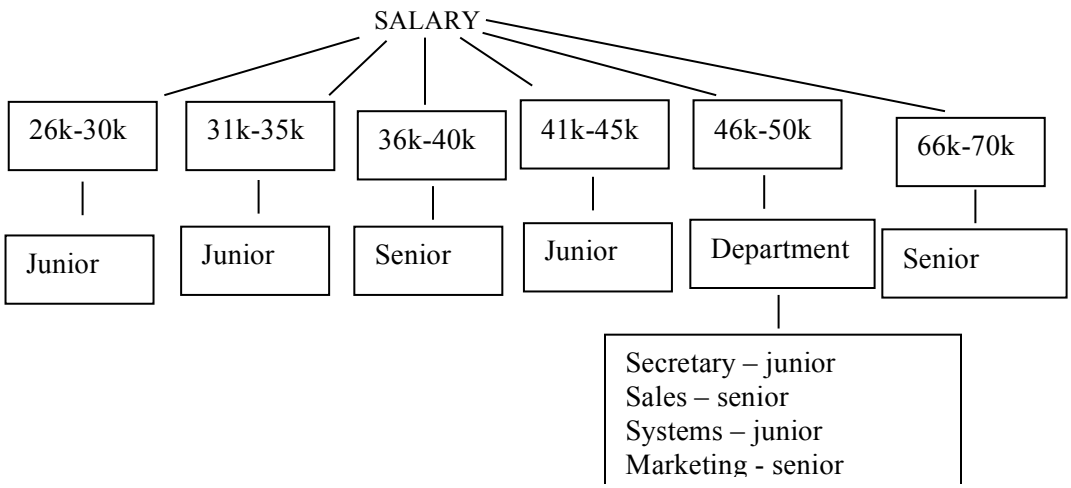
(b) Use your algorithm to construct a decision tree from the given data.

(c) Given a data tuple having the values "*systems*," "*26...30*," and "*46–50K*" for the attributes *department, age*, and *salary*, respectively, what would a naïve Bayesian classification of the *status* for the tuple be?

(d) Design a multilayer feed-forward neural network for the given data. Label the nodes in the input and output layers.

(e) Using the multilayer feed-forward neural network obtained above, show the weight values after one iteration of the backpropagation algorithm, given the training instance "*(sales, senior, 31...35, 46K...50K)*". Indicate your initial weight values and biases, and the learning rate used.

- A
  - The count of each tuple can be integrated into the calculation of the attribute selection measure (information gain)
  - Take the count into consideration to determine the most common class among the tuples.
- B



- C
  - Given system, 26-30, 46-50k calvulate naïve Bayesian classification
    - 3/165 = .018 Junior 0/165 = 0 Senior
      - Junior wins
- D
  -

9.4 Compare the advantages and disadvantages of *eager* classification (e.g., decision tree, Bayesian, neural network) versus *lazy* classification (e.g., *k*-nearest neighbor, case-based reasoning).

- Eager classification is faster at classifying while sacrificing accuracy.
- Lazy classification uses a richer hypothesis space, which can improve classification accuracy. It requires less time for training than eager classification. A disadvantage of lazy classification is that all training tuples need to be stored, which leads to expensive storage costs and requires efficient indexing techniques. Another disadvantage is that it is slower at classification because classifiers are not built until new tuples need to be classified. Furthermore, attributes are all equally weighted, which can decrease classification accuracy.

9.5 Write an algorithm for *k-nearest-neighbor classification* given k, the nearest number of neighbors, and n, the number of attributes describing each tuple.

```
#given k, the number of neighbors to compare, and n, the number of #their
attributes

test_tuple = test.pop()

# make priority queue limited on k
# so queue is len k
# is max heap, so pop highest value off
queue = priority_queue()

#for each neighbor
for neighbor in training_data:
        # get total distance for each neighbor
        dist = 0
        # I forget if this is how you iterate
        for x in list(range(n)):
                dist += eu_dist(neighbor[x], test_tuple[x])

        # update nearest neighbor
        queue.push(neighbor, dist)
        if len(queue) > k:
                queue.pop()

# get majority vote of neighbor class label
votes = {}

for near_neighbor in queue:
        # assuming neighbor is list with index 0 as the class label
        vote = near_neighbor[0]
        if vote in votes:
                votes[vote] += 1
        else:
                votes[vote] = 1

# return predicted class label
# get_max is method that returns key with max value from dict
return get_max(votes)
```

10.2 Suppose that the data mining task is to cluster points (with (x, y) representing location) into three clusters, where the points are

$$A_1(2,10), A_2(2,5), A_3(8,4), B_1(5,8), B_2(7,5), B_3(6,4), C_1(1,2), C_2(4,9).$$

The distance function is Euclidean distance. Suppose initially we assign A_1, B_1, and C_1 as the center of each cluster, respectively. Use the *k-means* algorithm to show *only*

(a) The three cluster centers after the first round of execution.

(b) The final three clusters.

- A
  - Euclidian function is (A1x1-A2x1)^2 + (A1x2-A2x2)^2
  - (1) {A1} (2) {B1, A3, B2, B3, C2} (3) {C1, A2}
  - (1) Center 2,10 (2) 6,6 (3) 1.5, 3.5
- B
  - (1) {A1, C2, B1} (2) {A3, B2, B3} (3) {C1, A2}