Data Mining section 5

By
Rhma Yasser

Resources:

1. Articles:

- https://www.geeksforgeeks.org/frequent-item-set-in-data-set-association-rule-mining/
- https://www.geeksforgeeks.org/apriori-algorithm/
- https://www.geeksforgeeks.org/frequent-pattern-growth-algorithm/
- https://medium.com/@byanalytixlabs/apriori-algorithm-in-data-mining-implementation-examples-and-more-ab17662ecb0e
- https://medium.com/@palakvb02/apriori-algorithm-in-machine-learning-b70a0374a998

2. Books:

- Data Mining Concepts and Techniques.
https://datamineaz.org/textbooks/hanDataMiningConceptual.pdf

Table of content

01

Data Mining Importance

Introduction

- **Data mining** refers to extracting or mining knowledge from large amounts of data.
- In other words, Data mining is the science, art, and technology of discovering large and complex bodies of data in order to discover useful patterns.
- Theoreticians and practitioners are continually seeking improved techniques to make the process more efficient, cost-effective, and accurate.

Data mining is the process of discovering interesting patterns, models, and other kinds of knowledge in large datasets. The term data mining, as a vivid view of searching for gold nuggets from data, appeared in 1990s.

Is Data Mining Part of Machine Learning?
Yes, data mining often overlaps with machine learning. Many data mining algorithms are based on machine learning principles because both fields aim to make data-driven decisions and predictions.

Introduction

Differences Between Data Mining and Machine Learning

Data mining and machine learning are unique processes that are often considered synonymous. However, while they are both useful for detecting patterns in large data sets, they operate very differently.

Data mining is the process of finding patterns in data. The beauty of data mining is that it helps to answer questions we didn't know to ask by proactively identifying non-intuitive data patterns through algorithms (e.g., consumers who buy peanut butter are more likely to buy paper towels). However, the interpretation of these insights and their application to business decisions still require human involvement.

Machine learning, meanwhile, is the process of teaching a computer to learn as humans do. With machine learning, computers learn how to determine probabilities and make predictions based on their data analysis. And, while machine learning sometimes uses data mining as part of its process, it ultimately doesn't require frequent human involvement on an ongoing basis (e.g., a self-driving car relies on data mining to determine where to stop, accelerate, and turn).

Unsupervised Learning Algorithms

Clustering

- K-Means
- Polynomial
- Hierarchical
- Fuzzy C-Means

Dimensionality Reduction

- Principal Component Analysis
- Kernel Principal Analysis

Association (Data Mining)

- Apriori Algorithm
- Eclat Algorithm
- FP-Growth Algorithm

02

Basic Concepts

Introduction

Frequent patterns are patterns (e.g., **itemsets**, **subsequences**, or **substructures**) that appear frequently in a data set. For example, a set of items, such as milk and bread, that appear frequently together in a transaction data set is a frequent itemset.

A **subsequence**, such as buying first a smartphone, then a smart TV, and then a smart home device, if it occurs frequently in a shopping history database, is a (frequent) sequential pattern.

A **substructure** can refer to different structural forms, such as subgraphs, subtrees, or sublattices. If a substructure occurs frequently, it is called a (frequent) structured pattern.

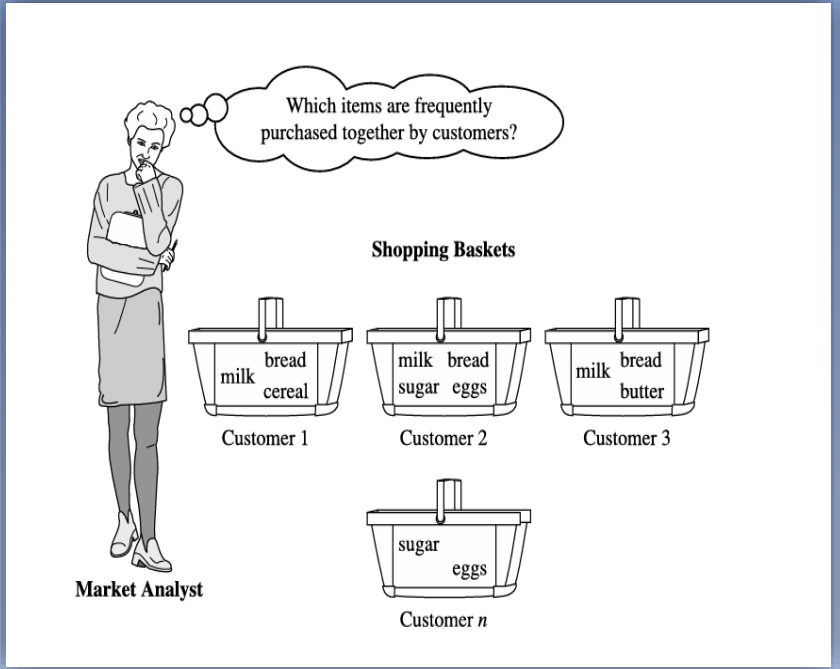
Finding frequent patterns plays an essential role in mining **associations**, **correlations**, and many other interesting **relationships** among data. Moreover, it helps in data **classification**, **clustering**, and other data mining tasks. Thus, frequent pattern mining has become an important data mining task and a focused theme in data mining research.

A set of items is referred to as an **itemset**.

Market basket analysis: a motivating example

1. Frequent itemset mining leads to the discovery of associations and correlations among items in large transactional or relational data sets.
2. With massive amounts of data continuously being collected and stored, many industries are interested in mining such patterns from their databases.
3. The discovery of interesting correlation relationships among huge amounts of business transaction records can help in many business decision-making processes such as **catalog design**, **cross-marketing**, and **customer shopping behavior analysis**.

A typical example of frequent itemset mining is **market basket analysis**. This process **analyzes** customer buying habits by finding **associations** between the **different items** that customers place in their "**shopping baskets**"



Market basket analysis: a motivating example

1. The discovery of these associations can **help retailers** develop **marketing strategies** by gaining insight into which items are frequently purchased together by customers.
2. For instance, if customers are buying milk, how likely are they to also buy bread (and what kind of bread) on the same trip to the supermarket?
3. This information can lead to increased sales, revenue, and customer acquisition by helping retailers do selective marketing and planned shelf space.

Each basket can then be represented by a **Boolean vector** of values assigned to these variables. The Boolean vectors can be analyzed to **extract** buying **patterns** that reflect items that are frequently associated or **purchased** together. These patterns can be represented in the form of association rules.

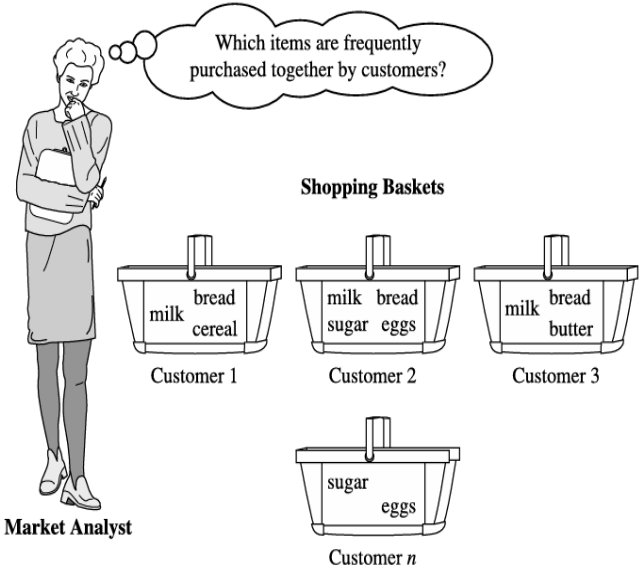
For example, the information that customers who purchase computers also tend to buy antivirus software at the same time is represented in the following association rule:

computer ⇒antivirus_software [support = 2%,confidence = 60%]

Market basket analysis: a motivating example

A **support** of **2%** for Last Rule means that **2% of all the transactions under analysis show that computer and antivirus software are purchased together.**

A **confidence** of **60**% means that **60% of the customers who purchased a computer also bought the software.**



03

Support & Confidence

Support

Support

In data mining, support refers to the relative frequency of an item set in a dataset. For example, if an itemset occurs in 5% of the transactions in a dataset, it has a support of 5%. Support is often used as a threshold for identifying frequent item sets in a dataset, which can be used to generate association rules.

```
Support({milk, bread}) = Number of transactions containing
                          {milk, bread} / Total number of transactions
                        = 100 / 1000
                        = 10%
```

Support(X) = (Number of transactions containing X) / (Total number of transactions)

Note:

This is called Relative support (as it is a proportion) and the support count is called the Absolute support.

Confidence

Confidence

In data mining, confidence is a measure of the reliability or support for a given association rule. It is defined as the proportion of cases in which the association rule holds true, or in other words, the percentage of times that the items in the antecedent (the “if” part of the rule) appear in the same transaction as the items in the consequent (the “then” part of the rule).

```
Confidence("If a customer buys milk, they will also buy bread")
  = Number of transactions containing
   {milk, bread} / Number of transactions containing {milk}
  = 100 / 200
  = 50%
```

Confidence(X => Y) = (Number of transactions containing X and Y) / (Number of transactions containing X)

Support	Confidence
Support is a measure of the number of times an item set appears in a dataset.	Confidence is a measure of the likelihood that an itemset will appear if another itemset appears.
Support is calculated by dividing the number of transactions containing an item set by the total number of transactions.	Confidence is calculated by dividing the number of transactions containing both itemsets by the number of transactions containing the first itemset.
Support is used to identify itemsets that occur frequently in the dataset.	Confidence is used to evaluate the strength of a rule.
Support is often used with a threshold to identify itemsets that occur frequently enough to be of interest.	Confidence is often used with a threshold to identify rules that are strong enough to be of interest.
Support is interpreted as the percentage of transactions in which an item set appears.	Confidence is interpreted as the percentage of transactions in which the second itemset appears given that the first itemset appears.

04

Important Definitions

•**Support :** It is one of the measures of interestingness. This tells about the usefulness and certainty of rules.
5% Support means total 5% of transactions in the database follow the rule.

$$Support(A \rightarrow B) = Support\_count(A \cup B)$$

•**Confidence:** A confidence of 60% means that 60% of the customers who purchased a milk and bread also bought butter.

$$Confidence(A \rightarrow B) = Support\_count(A \cup B) / Support\_count(A)$$

If a rule satisfies both minimum support and minimum confidence, it is a **strong rule**.

•**Support_count(X)**: Number of transactions in which X appears. If X is A **union** B then it is the number of transactions in which A and B both are present.

•**Maximal Itemset:** An itemset is maximal frequent if none of its supersets are frequent.

•**Closed Itemset:** An itemset is closed if none of its immediate supersets have same support count same as Itemset.

•**K- Itemset:** Itemset which contains K items is a K-itemset. So it can be said that an itemset is frequent if the corresponding support count is greater than the minimum support count.

05

Apriori Algorithm

Introduction

Apriori algorithm is given by R. Agrawal and R. Srikant in 1994 for finding frequent itemsets in a dataset for boolean association rule. Name of the algorithm is Apriori because it uses **prior** knowledge of frequent itemset properties. We apply an iterative approach or **level-wise search** where k-frequent itemsets are used to find k+1 itemsets.

To **improve the efficiency** of level-wise generation of frequent itemsets, an important property is used called ***Apriori property*** which helps by **reducing** the **search space**.

Apriori Property
All non-empty subset of frequent itemset must be frequent.

The key concept of Apriori algorithm is its **anti-monotonicity** of support measure. Apriori assumes that

> *All subsets of a frequent itemset must be frequent(Apriori property).*
>
> *If an itemset is infrequent, all its supersets will be infrequent.*

Introduction

This property belongs to a special category of properties called **antimonotonicity** in the sense that *if a set cannot pass a test, all of its supersets will fail the same test as well*. It is called *antimonotonicity* because the property is monotonic in the context of failing a test.

"How is the Apriori property used in the algorithm?" To understand this, let us look at how L_{k-1} is used to find L_k for $k \geq 2$. A two-step process is followed, consisting of **join** and **prune** actions.

1. **The join step**. To find L_k, a set of **candidate** k-itemsets is generated by joining L_{k-1} with itself. This set of candidates is denoted C_k. Let l_1 and l_2 be itemsets in L_{k-1}. The notation $l_i[j]$ refers to the jth item in l_i (e.g., $l_1[k-2]$ refers to the second to the last item in l_1). For efficient implementation, Apriori assumes that items within a transaction or itemset are sorted in lexicographic order. For the $(k-1)$-itemset, l_i, this means that the items are sorted such that $l_i[1] < l_i[2] < \cdots < l_i[k-1]$. The join, $L_{k-1} \bowtie L_{k-1}$, is performed, where members of L_{k-1} are joinable if their first $(k-2)$ items are in common. That is, members l_1 and l_2 of L_{k-1} are joined if $(l_1[1] = l_2[1]) \wedge (l_1[2] = l_2[2]) \wedge \cdots \wedge (l_1[k-2] = l_2[k-2]) \wedge (l_1[k-1] < l_2[k-1])$. The condition $l_1[k-1] < l_2[k-1]$ simply ensures that no duplicates are generated. The resulting itemset formed by joining l_1 and l_2 is $\{l_1[1], l_1[2], \ldots, l_1[k-2], l_1[k-1], l_2[k-1]\}$.
2. **The prune step**. C_k is a superset of L_k, that is, its members may or may not be frequent, but all of the frequent k-itemsets are included in C_k. A database scan to determine the count of each candidate in C_k would result in the determination of L_k (i.e., all candidates having a count no less than the minimum support count are frequent by definition and therefore belong to L_k). C_k, however, can be huge, and so this could involve heavy computation. To reduce the size of C_k, the Apriori property is used as follows. Any $(k-1)$-itemset that is not frequent cannot be a subset of a frequent k-itemset. Hence, if any $(k-1)$-subset of a candidate k-itemset is not in L_{k-1}, then the candidate cannot be frequent either and so can be removed from C_k. This **subset testing** can be done quickly by maintaining a hash tree of all frequent itemsets.

Steps – Step 1

Step-1: K=1

(I) Create a table containing support count of each item present in dataset – Called **C1(candidate set)**

Itemset	sup_count
I1	6
I2	7
I3	6
I4	2
I5	2

TID	items
T1	I1, I2 , I5
T2	I2,I4
T3	I2,I3
T4	I1,I2,I4
T5	I1,I3
T6	I2,I3
T7	I1,I3
T8	I1,I2,I3,I5
T9	I1,I2,I3

minimum support count is 2

minimum confidence is 60%

Steps – Step 1

(II) compare candidate set item's support count with minimum support count(here min_support=2 if support_count of candidate set items is less than min_support then remove those items). This gives us itemset L1.

Itemset	sup_count
I1	6
I2	7
I3	6
I4	2
I5	2

Steps – Step 2

Step-2: K=2

•Generate candidate set C2 using L1 (this is called join step). Condition of joining L_{k-1} and L_{k-1} is that it should have (K-2) elements in common.

•Check all subsets of an itemset are frequent or not and if not frequent remove that itemset.(Example subset of{I1, I2} are {I1}, {I2} they are frequent. Check for each itemset)

•Now find support count of these itemsets by searching in dataset.

Itemset	sup_count
I1,I2	4
I1,I3	4
I1,I4	1
I1,I5	2
I2,I3	4
I2,I4	2
I2,I5	2
I3,I4	0
I3,I5	1
I4,I5	0

Steps - Step 2

(II) compare candidate (C2) support count with minimum support count(here min_support=2 if support_count of candidate set item is less than min_support then remove those items) this gives us itemset L2.

Itemset	sup_count
I1,I2	4
I1,I3	4
I1,I5	2
I2,I3	4
I2,I4	2
I2,I5	2

Steps – Step 3

Step-3:

- Generate candidate set C3 using L2 (join step). Condition of joining L_{k-1} and L_{k-1} is that it should have (K-2) elements in common. So here, for L2, first element should match. So itemset generated by joining L2 is {I1, I2, I3}{I1, I2, I5}{I1, I3, i5}{I2, I3, I4}{I2, I4, I5}{I2, I3, I5}
- Check if all subsets of these itemsets are frequent or not and if not, then remove that itemset.(Here subset of {I1, I2, I3} are {I1, I2},{I2, I3},{I1, I3} which are frequent. For {I2, I3, I4}, subset {I3, I4} is not frequent so remove it. Similarly check for every itemset)
- find support count of these remaining itemset by searching in dataset.

Itemset	sup_count
I1,I2,I3	2
I1,I2,I5	2

Steps – Step 3

(II) Compare candidate (C3) support count with minimum support count(here min_support=2 if support_count of candidate set item is less than min_support then remove those items) this gives us itemset L3.

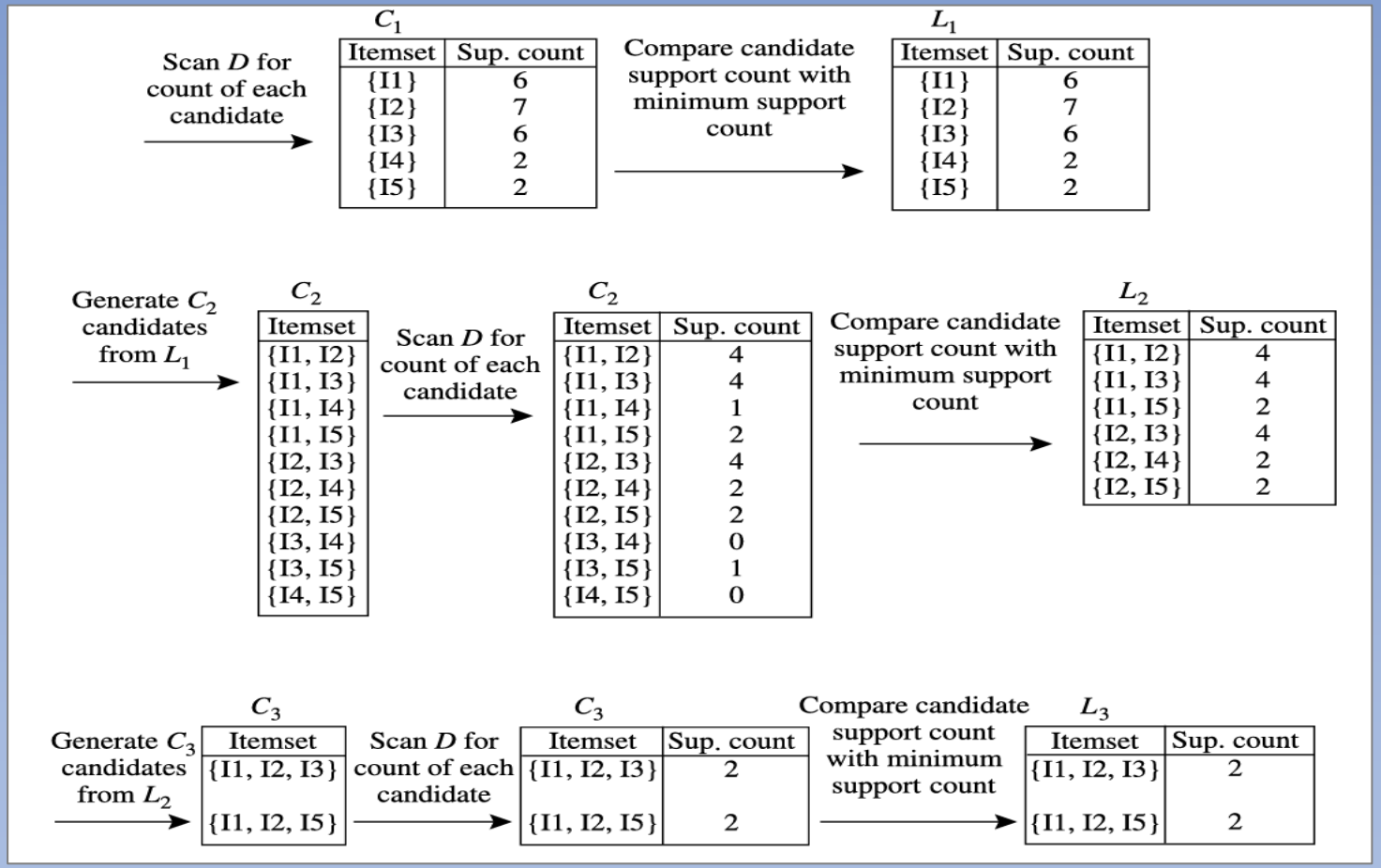
Itemset	sup_count
I1,I2,I3	2
I1,I2,I5	2

Steps- Step 4

Step-4:

- Generate candidate set C4 using L3 (join step). Condition of joining L_{k-1} and L_{k-1} (K=4) is that, they should have (K-2) elements in common. So here, for L3, first 2 elements (items) should match.
- Check all subsets of these itemsets are frequent or not (Here itemset formed by joining L3 is {I1, I2, I3, I5} so its subset contains {I1, I3, I5}, which is not frequent). So no itemset in C4
- We stop here because no frequent itemsets are found further

The Whole Steps



Steps- Confidence Calculation

A confidence of 60% means that 60% of the customers, who purchased milk and bread also bought butter.

Confidence(A->B)=Support_count(A∪B)/Support_count(A)

So here, by taking an example of any frequent itemset, we will show the rule generation.

Itemset {I1, I2, I3} //from L3

SO rules can be

[I1^I2]=>[I3] //confidence = sup(I1^I2^I3)/sup(I1^I2) = 2/4*100=50%

[I1^I3]=>[I2] //confidence = sup(I1^I2^I3)/sup(I1^I3) = 2/4*100=50%

[I2^I3]=>[I1] //confidence = sup(I1^I2^I3)/sup(I2^I3) = 2/4*100=50%

[I1]=>[I2^I3] //confidence = sup(I1^I2^I3)/sup(I1) = 2/6*100=33%

[I2]=>[I1^I3] //confidence = sup(I1^I2^I3)/sup(I2) = 2/7*100=28%

[I3]=>[I1^I2] //confidence = sup(I1^I2^I3)/sup(I3) = 2/6*100=33%

So if minimum confidence is 50%, then first 3 rules can be considered as **strong association** rules.

Steps- Confidence Calculation

The confidence of a rule $\{I1, I2\} \Rightarrow \{I3\}$ is calculated as:

$$\text{Confidence}(\{I1, I2\} \Rightarrow \{I3\}) = \frac{\text{Support}(\{I1, I2, I3\})}{\text{Support}(\{I1, I2\})}$$

This specific calculation is used because confidence measures the likelihood that $\{I3\}$ appears in a transaction, given that $\{I1, I2\}$ already appear in it.

Why Not $\frac{\text{Support}(\{I1,I2\})}{\text{Support}(\{I3\})}$**?**

If we used $\frac{\text{Support}(\{I1,I2\})}{\text{Support}(\{I3\})}$, it would mean calculating how often $\{I1, I2\}$ appears relative to $\{I3\}$, which does not tell us about the likelihood of $\{I3\}$ appearing when $\{I1, I2\}$ is present.

Confidence aims to assess conditional probability, or the chance of finding $\{I3\}$ in transactions that already contain $\{I1, I2\}$, aligning with the conditional probability definition:

$$P(\{I3\}|\{I1, I2\}) = \frac{P(\{I1, I2, I3\})}{P(\{I1, I2\})}$$

Try it yourself – Apriori Only

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,"):

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

Apriori Limitations

Apriori can suffer from two nontrivial costs.

• It may still need to generate a **huge number of candidate sets**. For example, if there are 10^4 frequent 1-itemsets, the Apriori algorithm will need to generate more than 10^7 candidate 2-itemsets.

• It may need to **repeatedly scan** the whole database and check a large set of candidates by pattern matching. It is costly to go over each transaction in the database to determine the support of the candidate itemsets.

05

FP-growth Algorithm

Introduction

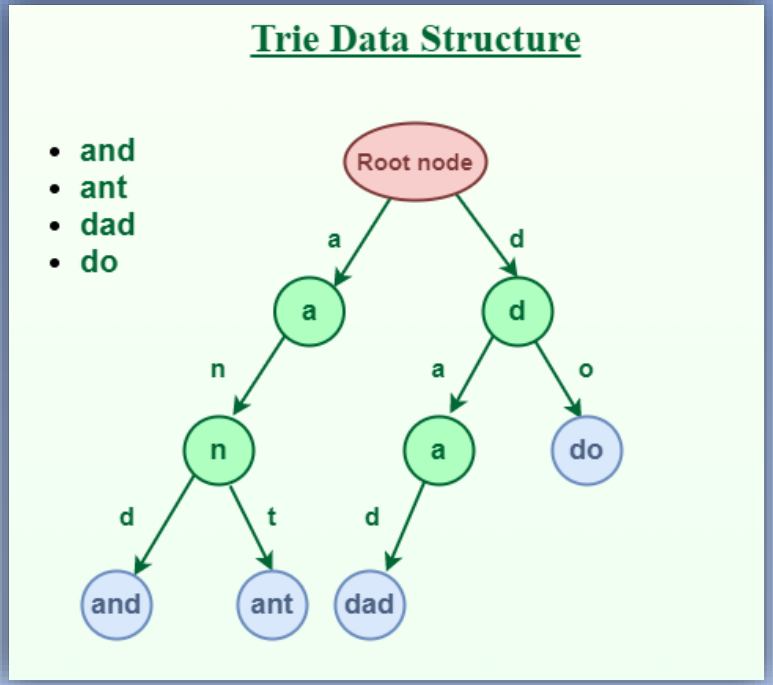
FP-growth stand for frequent pattern growth and it is used for finding frequent itemsets **without candidate generation.**

The two primary drawbacks of the Apriori Algorithm are:

1.At each step, candidate sets have to be built.

2.To build the candidate sets, the algorithm has to repeatedly scan the database.

These two properties inevitably make the algorithm **slower**. To overcome these redundant steps, a new association-rule mining algorithm was developed named Frequent Pattern Growth Algorithm. It overcomes the disadvantages of the Apriori algorithm by storing all the transactions in a Trie Data Structure.



Steps

Transaction ID	Items
T1	{E, K, M, N, O, Y}
T2	{D, E, K, N, **O**, Y}
T3	{A, E, K, M}
T4	{C, K, M, U, Y}
T5	{C, E, I, K, O, O}

Item	Frequency
A	1
C	2
D	1
E	4
I	1
K	5
M	3
N	2
O	4
U	1
Y	3

The above-given data is a hypothetical dataset of transactions with each letter representing an item. The frequency of each individual item is computed:-

Steps

Let the minimum support be 3. A **Frequent Pattern set** is built which will contain all the elements whose frequency is greater than or equal to the minimum support. These elements are stored in **descending** order of their respective frequencies. After insertion of the relevant items, the set L looks like this:-

3

L = {K : 5, E : 4, M : 3, O : 4, Y : 3}

Now, for each transaction, the respective **Ordered-Item set** is built. It is done by iterating the Frequent Pattern set and checking if the current item is contained in the transaction in question. If the current item is contained, the item is inserted in the Ordered-Item set for the current transaction. The following table is built for all the transactions:

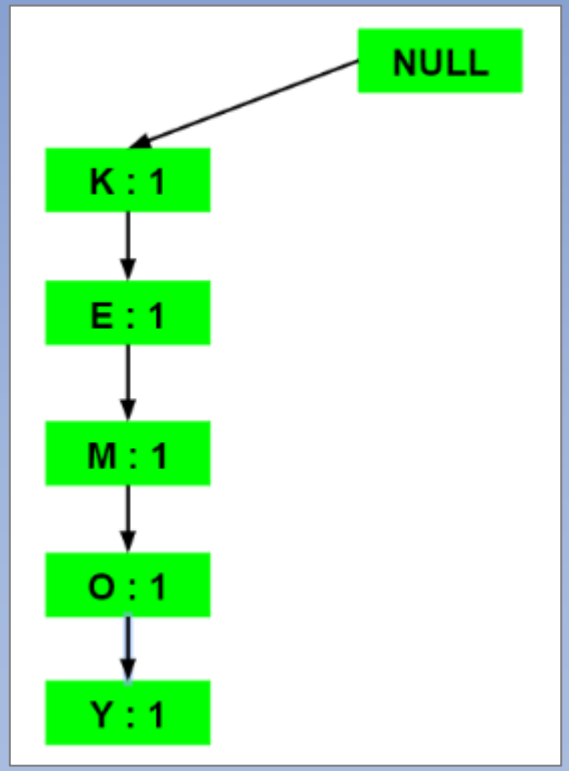
Transaction ID	Items	Ordered-Item Set
T1	{E, K, M, N, O, Y}	{K, E, M, O, Y}
T2	{D, E, K, N, O, Y}	{K, E, O, Y}
T3	{A, E, K, M}	{K, E, M}
T4	{C, K, M, U, Y}	{K, M, Y}
T5	{C, E, I, K, O, O}	{K, E, O}

Steps

Now, all the Ordered-Item sets are inserted into a **Trie Data Structure**.

a) **Inserting the set {K, E, M, O, Y}:**

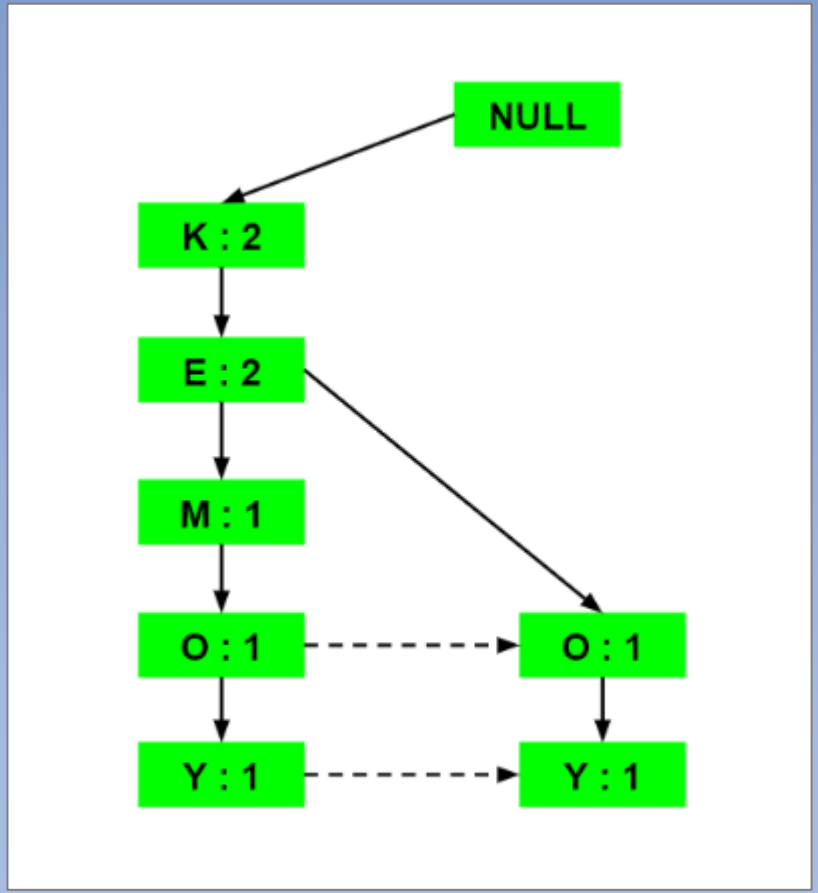
Here, all the items are simply linked one after the other in the order of occurrence in the set and initialize the support count for each item as 1.



Steps

b) **Inserting the set {K, E, O, Y}:**

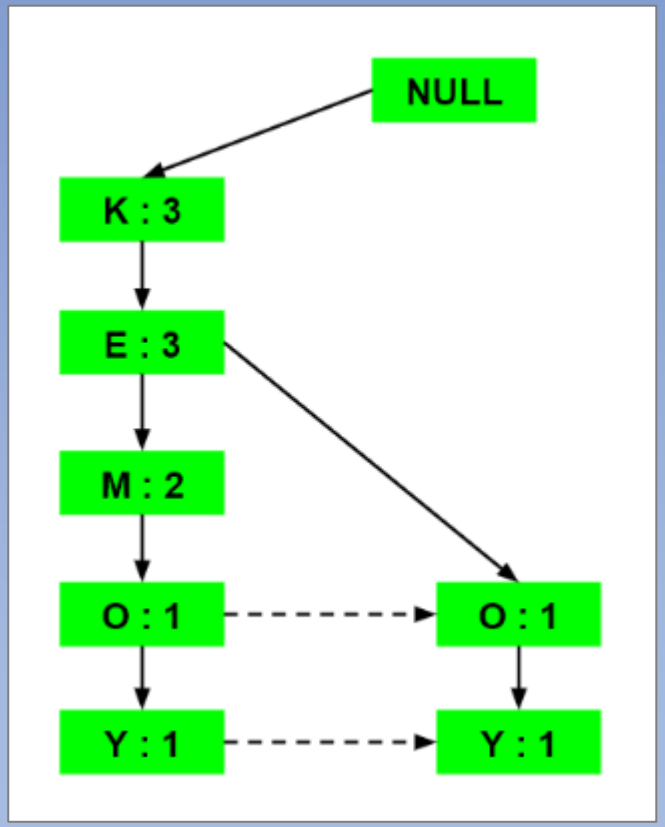
Till the insertion of the elements K and E, simply the support count is increased by 1. On inserting O we can see that there is no direct link between E and O, therefore a new node for the item O is initialized with the support count as 1 and item E is linked to this new node. On inserting Y, we first initialize a new node for the item Y with support count as 1 and link the new node of O with the new node of Y.



Steps

c) **Inserting the set {K, E, M}:**

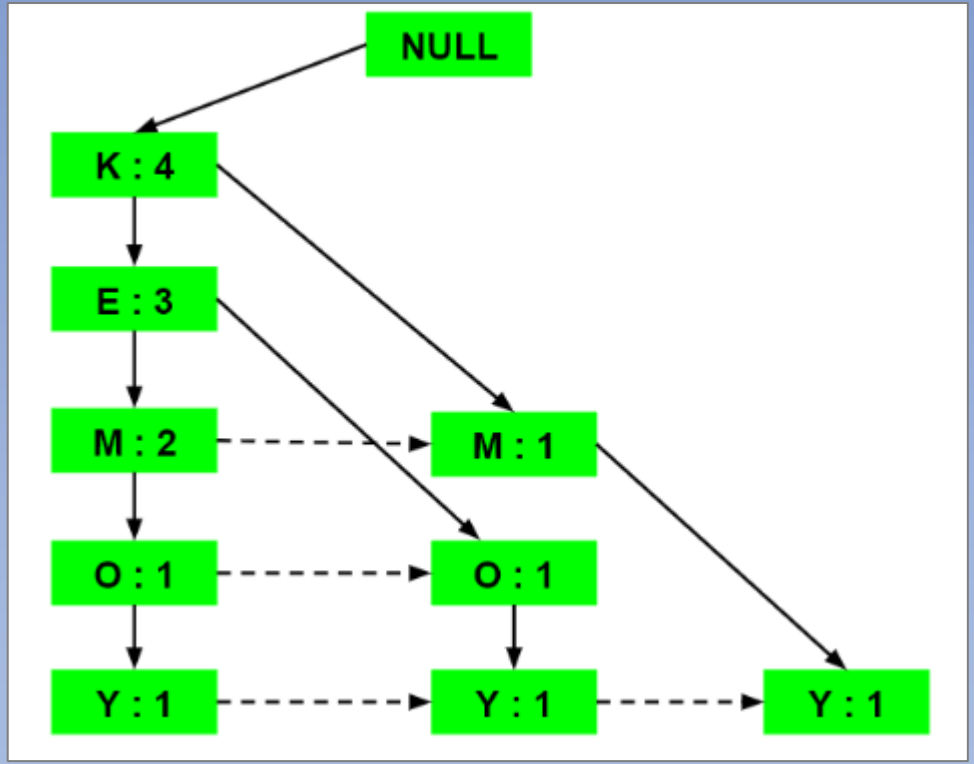
Here simply the support count of each element is increased by 1.



Steps

d) **Inserting the set {K, M, Y}:**

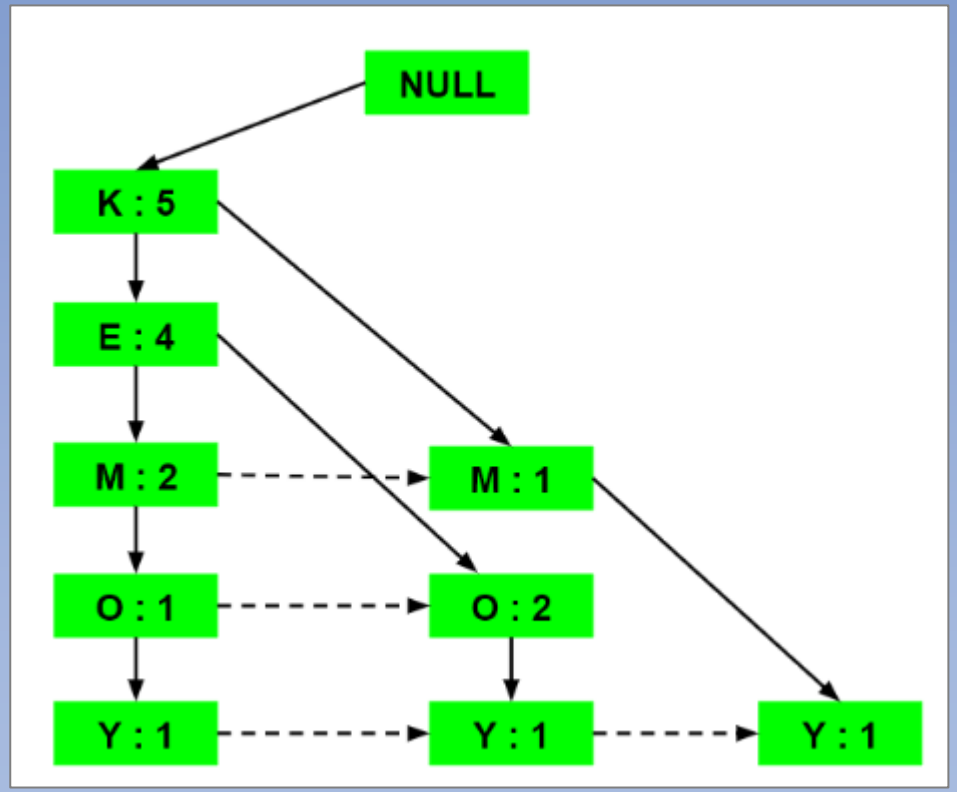
Similar to step b), first the support count of K is increased, then new nodes for M and Y are initialized and linked accordingly.



Steps

e) **Inserting the set {K, E, O}:**

Here simply the support counts of the respective elements are increased. Note that the support count of the new node of item O is increased.



Steps

Now, for each item, the **Conditional Pattern Base** is computed which is path labels of all the paths which lead to any node of the given item in the frequent-pattern tree. Note that the items in the below table are arranged in the ascending order of their frequencies.

Items	Conditional Pattern Base
Y	{{K,E,M,O : 1}, {K,E,O : 1}, {K,M : 1}}
O	{{K,E,M : 1}, {K,E : 2}}
M	{{K,E : 2}, {K : 1}}
E	{K : 4}
K	

Steps

Now for each item, the **Conditional Frequent Pattern Tree is built.** It is done by taking the set of elements that is common in all the paths in the Conditional Pattern Base of that item and calculating its support count by summing the support counts of all the paths in the Conditional Pattern Base.

Items	Conditional Pattern Base	Conditional Frequent Pattern Tree
Y	{{K,E,M,O : 1}, {K,E,O : 1}, {K,M : 1}}	{K : 3}
O	{{K,E,M : 1}, {K,E : 2}}	{K,E : 3}
M	{{K,E : 2}, {K : 1}}	{K : 3}
E	{K : 4}	{K : 4}
K		

Steps

From the Conditional Frequent Pattern tree, the **Frequent Pattern rules** are generated by pairing the items of the Conditional Frequent Pattern Tree set to the corresponding to the item as given in the below table.

or each row, two types of association rules can be inferred for example for the first row which contains the element, the rules K -> Y and Y -> K can be inferred. To determine the valid rule, the confidence of both the rules is calculated and the one with confidence greater than or equal to the minimum confidence value is retained.

Items	Frequent Pattern Generated
Y	{<K,Y : 3>}
O	{<K,O : 3>, <E,O : 3>, <E,K,O : 3>}
M	{<K,M : 3>}
E	{<E,K : 4>}
K	

Try it yourself – FP-growth only

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,"):

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

Limitations

The FP-growth method transforms the problem of finding long frequent patterns into searching for shorter ones in much smaller conditional databases recursively and then concatenating the suffix. It uses the least frequent items as a suffix, offering good selectivity. The method substantially reduces the search costs.

When the database is **large**, it is sometimes unrealistic to construct a main memory-based FP-tree. An interesting alternative is to first **partition** the database into a set of projected databases and then construct an FP-tree and mine it in each projected database. This process can be recursively applied to any projected database if its **FP-tree still cannot fit in main memory**.

Thank You