



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

Name:	Shreeya Sunil Hudekar
Roll No:	13
Class/Sem:	TE/V
Experiment No.:	9
Title:	Implementation of association mining algorithms like FP Growth using languages like JAVA/ python.
Date of Performance:	
Date of Submission:	
Marks:	
Sign of Faculty:	



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

Aim :- To implement the FP-Growth algorithm using Python.

Objective: Understand the working principles of the FP-Growth algorithm and implement it in Python.

Theory

FP-Growth (Frequent Pattern Growth) is an algorithm for frequent item set mining and association rule learning over transactional databases. It efficiently discovers frequent patterns by constructing a compact data structure called the FP-Tree and mining it to extract frequent item sets.

Key Concepts:

1. **FP-Tree:** A data structure that represents the transaction database compressed by linking frequent items in a tree structure, along with their support counts.
2. **Header Table:** A compact structure that stores pointers to the first occurrences of items in the FP-Tree and their support counts.
3. **Frequent Item Set Mining:**
 - **Conditional Pattern Base:** For each frequent item, construct a conditional pattern base consisting of the prefix paths in the FP-Tree.
 - **Conditional FP-Tree:** Construct a conditional FP-Tree from the conditional pattern base and recursively mine frequent item sets.

Steps in FP-Growth Algorithm:

1. **Build FP-Tree:** Construct the FP-Tree by inserting transactions and counting support for each item.
2. **Create Header Table:** Build a header table with links to the first occurrences of items in the FP-Tree.
3. **Mine FP-Tree:**
 - Identify frequent single items by their support.
 - Construct conditional pattern bases and conditional FP-Trees recursively.
 - Combine frequent item sets from conditional FP-Trees to find all frequent item sets.

Example

Given a transactional database:

- Implement the FP-Growth algorithm to find all frequent itemsets with a specified minimum support threshold.

Code

```
from collections import defaultdict, namedtuple
```

```
import itertools
```

```
# FP-tree node structure
```

```
class TreeNode:
```



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

```
def __init__(self, item, count, parent):
```

```
    self.item = item
```

```
    self.count = count
```

```
    self.parent = parent
```

```
    self.children = { }
```

```
    self.link = None
```

```
def increment(self, count):
```

```
    self.count += count
```

```
class FPTree:
```

```
    def __init__(self):
```

```
        self.root = TreeNode(None, 1, None) # Create the root node
```

```
        self.header_table = defaultdict(list) # Header table
```

```
    def add_transaction(self, transaction, count):
```

```
        current_node = self.root
```

```
        for item in transaction:
```

```
            if item in current_node.children:
```

```
                current_node.children[item].increment(count)
```

```
            else:
```

```
                new_node = TreeNode(item, count, current_node)
```

```
                current_node.children[item] = new_node
```

```
                self.header_table[item].append(new_node)
```

```
                current_node = current_node.children[item]
```

```
    def conditional_tree(self, base_pattern):
```



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

```
conditional_tree = FPTree()
```

```
item_counts = defaultdict(int)
```

```
for node in self.header_table[base_pattern]:
```

```
    count = node.count
```

```
    path = []
```

```
    parent = node.parent
```

```
    while parent and parent.item:
```

```
        path.append(parent.item)
```

```
        parent = parent.parent
```

```
    for i in range(count):
```

```
        conditional_tree.add_transaction(reversed(path), 1)
```

```
        item_counts.update({item: 1 for item in path})
```

```
    return conditional_tree, item_counts
```

```
def build_fp_tree(transactions, min_support):
```

```
    item_count = defaultdict(int)
```

```
    for transaction in transactions:
```

```
        for item in transaction:
```

```
            item_count[item] += 1
```

```
    # Remove items that don't meet the min_support threshold
```

```
    item_count = {k: v for k, v in item_count.items() if v >= min_support}
```

```
    sorted_items = sorted(item_count.items(), key=lambda x: (-x[1], x[0]))
```



```
def sort_transaction(transaction):  
    transaction = [item for item in transaction if item in item_count]  
    transaction.sort(key=lambda x: (-item_count[x], x))  
    return transaction
```

```
tree = FPTree()  
  
for transaction in transactions:  
    sorted_transaction = sort_transaction(transaction)  
    tree.add_transaction(sorted_transaction, 1)  
  
return tree, sorted_items
```

```
def mine_fp_tree(tree, min_support, prefix):  
    patterns = { }  
  
    for item, nodes in tree.header_table.items():  
        support = sum(node.count for node in nodes)  
        if support >= min_support:  
            new_pattern = prefix + [item]  
            patterns[tuple(new_pattern)] = support  
  
            conditional_tree, _ = tree.conditional_tree(item)  
            if conditional_tree.root.children:  
                conditional_patterns = mine_fp_tree(conditional_tree, min_support,  
new_pattern)  
                patterns.update(conditional_patterns)
```



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

return patterns

```
def fpgrowth(transactions, min_support):  
    fp_tree, sorted_items = build_fp_tree(transactions, min_support)  
    return mine_fp_tree(fp_tree, min_support, [])  
  
# Sample usage  
if __name__ == "__main__":  
    transactions = [  
        ['milk', 'bread', 'butter'],  
        ['beer', 'bread'],  
        ['milk', 'bread', 'butter', 'beer'],  
        ['bread', 'butter'],  
        ['milk', 'bread', 'butter', 'beer'],  
    ]  
    min_support = 2  
    patterns = fpgrowth(transactions, min_support)  
  
    for pattern, support in patterns.items():  
        print(f"Pattern: {pattern}, Support: {support}")
```

Output:

```
Pattern: ('bread',), Support: 5  
Pattern: ('butter',), Support: 4  
Pattern: ('butter', 'bread'), Support: 4  
Pattern: ('milk',), Support: 3  
Pattern: ('milk', 'bread'), Support: 3  
Pattern: ('milk', 'butter'), Support: 3  
Pattern: ('milk', 'butter', 'bread'), Support: 3  
Pattern: ('milk', 'beer'), Support: 2  
Pattern: ('milk', 'beer', 'bread'), Support: 2
```



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

Pattern: ('milk', 'beer', 'butter'), Support: 2

Pattern: ('milk', 'beer', 'butter', 'bread'), Support: 2

Pattern: ('beer',), Support: 3

Pattern: ('beer', 'bread'), Support: 3

Pattern: ('beer', 'butter'), Support: 2

Pattern: ('beer', 'butter', 'bread'), Support: 2



Conclusion

Explain how FP-Growth manages and mines item sets of varying lengths in transactional databases.

FP-Growth manages and mines itemsets of varying lengths by first building an FP-tree, a compact representation of the transaction database, where frequent items are stored in sorted order and common prefixes are shared. It handles varying-length itemsets by representing each transaction as a path of nodes, with longer itemsets extending deeper in the tree. During mining, the algorithm recursively extracts frequent patterns by constructing conditional FP-trees for each item, progressively building itemsets without generating all possible candidates, ensuring efficiency. This divide-and-conquer approach allows it to mine patterns of different lengths seamlessly.