Data mining ID 2222

Short report on homework 2

Discovery of Frequent Itemsets and Association Rules

Sofie Schnitzer and Adam Åström

November 18, 2024

Introduction

The focus of this week's assignment was to discover frequent itemsets in a datafile of transaction data. The programming language chosen for this assignment is Python because of its ease of use and built-in tools for handling data-driven tasks. In this assignment, we implemented the A-Priori algorithm, which is commonly used to find frequent itemsets in transaction datasets based on a given minimum support threshold.

The A-Priori algorithm was implemented step by step, starting with generating 1-itemsets, followed by iteratively creating and filtering larger candidate itemsets until no more frequent itemsets could be found. Python's data structures, such as sets and dictionaries, made it straightforward to manage and count itemsets efficiently.

Basic Implementation

In order to be able to use the dataset in testing our code, we read the dataset with a basic function that translates each line in the document to a transaction (a list of strings where each item in a transaction is one item in the list).

```
        transactions = [line.strip().split() for line in file]
```

Apriori

When we have read all transactions, we use the Apriori algorithm to find frequent sets of items within the transaction. What is classed as "frequent" has to do with the support threshold that we decide on, in this case 1000. This means that if a set of items does not occur in at least 1000 transactions, it does not have enough support and is not categorized as frequent.

For the apriori algorithm we have a few functions. We begin with a set of all items in the transaction list, held by variable items. We then make a tuple of each itemset containing 1 item (singletons), aka we make a tuple of each unique item in transactions.

```
items = {item for transaction in transactions for item in transaction}
current_itemsets = [tuple([item]) for item in sorted(items)]
```

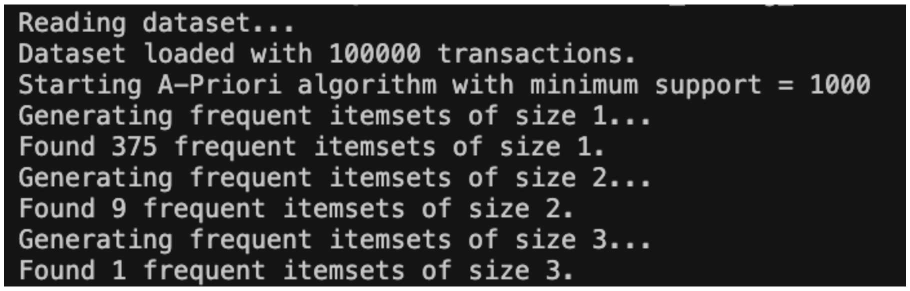
We then get the support for each item/itemset with a function 'get_itemset_support', which based on the current_itemsets and transactions will map each itemset to its support count, which is stored by the 'itemset_support' set. For each transaction, we check if the itemset is present, and if it is, we add 1 to the support counter for that itemset. This is first done for all itemsets with 1 item, 2 items, 3 items (there were no present itemsets containing 4 or more items).

```
for transaction in transactions:
    transaction_set = set(transaction)
    for itemset in itemsets:
        if transaction_set.issuperset(itemset):
            itemset_support[itemset] += 1
```

When we have done this for singleton itemsets, we will use these frequent singletons to identify the frequent doubletons. This is done with the `generate_candidates` function, which combines pairs of frequent k-1 -itemsets to create candidate k -itemsets. The function ensures that only itemsets sharing the first k-2 -items are combined. These candidates are then evaluated against the transaction data to count their support. The same process is repeated until no more frequent itemsets are found.

```
for i in range(len(frequent_items)):
    for j in range(i + 1, len(frequent_items)):
        itemset1, itemset2 = frequent_items[i], frequent_items[j]
        if itemset1[:k - 1] == itemset2[:k - 1]:
            candidate = tuple(sorted(set(itemset1).union(set(itemset2))))
            candidates.add(candidate)
```

The result of this part of the algorithm looks like below. We see that 375 singletons were found >1000 times, 9 doubletons were found and only 1 tripleton.



```
Reading dataset...
Dataset loaded with 100000 transactions.
Starting A-Priori algorithm with minimum support = 1000
Generating frequent itemsets of size 1...
Found 375 frequent itemsets of size 1.
Generating frequent itemsets of size 2...
Found 9 frequent itemsets of size 2.
Generating frequent itemsets of size 3...
Found 1 frequent itemsets of size 3.
```

Bonus implementation

Association rules

For the bonus points, we were to find association rules between items in the dataset. The association rule means that A→B, where A is an itemset and B is an item. The rule itself is that where the set of items A (the antecedent) are present, B will likely be present as well. However, likely is a quite bad term for statistical use, instead we use a value for confidence. The confidence support is how sure we are of the rule A→B.

$$\text{Confidence}(A \rightarrow B) = \frac{\text{Support}(A \cup B)}{\text{Support}(A)}$$

A={apple} appears in 4 transactions, and A ∪ B={apple, pear} appears in 3 transactions, the confidence is 3/4=0.753/4 = 0.753/4=0.75.

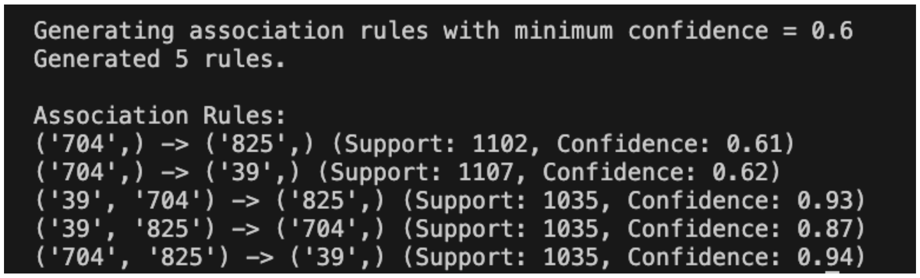
We use a confidence threshold of 0.6 to find association rules with quite high confidence scores.

```
        subsets = [tuple(sorted(comb)) for i in range(1, len(itemset)) for comb in
combinations(itemset, i)]
        for antecedent in subsets:
            consequent = tuple(sorted(set(itemset) - set(antecedent)))

            if not consequent:
                continue
            antecedent_support = frequent_itemsets.get(antecedent, 0)
            confidence = itemset_support / antecedent_support

            if confidence >= min_confidence:
                rules.append((antecedent, consequent, itemset_support, confidence))
```

The antecedent is determined by finding all possible subsets of a frequent itemset, excluding the empty set and the full itemset itself. These subsets are generated using the combinations function.

The consequent is calculated as the remaining items in the frequent itemset after removing the antecedent, using a set difference operation. For each antecedent-consequent pair, the support of the antecedent is retrieved, and the confidence of the rule A→ B is computed. Only rules where the confidence meets the minimum threshold are retained, ensuring that the strongest associations are identified.

When using a threshold of 60%, 5 rules were generated and can be seen in the output.



```
Generating association rules with minimum confidence = 0.6
Generated 5 rules.

Association Rules:
('704',) -> ('825',) (Support: 1102, Confidence: 0.61)
('704',) -> ('39',) (Support: 1107, Confidence: 0.62)
('39', '704') -> ('825',) (Support: 1035, Confidence: 0.93)
('39', '825') -> ('704',) (Support: 1035, Confidence: 0.87)
('704', '825') -> ('39',) (Support: 1035, Confidence: 0.94)
```