Apriori Algorithm

An algorithm to find the maximum count of association rules between k-frequent items that can be formed from a dataset.

Table of Contents

Introduction

Apriori algorithm is given by R. Agrawal and R. Srikant in 1994 for finding frequent itemsets in a dataset for boolean association rule. Naming of the algorithm has been done as such because it uses prior knowledge of the frequent itemset properties. We use either level-wise search or iterative approach on k-frequent itemsets to form k+1 itemsets.

Efficiency of the level-wise generation of itemsets can be improved by the help of an important property called ***Apriori Property.***

> ***Apriori Property -***
>
> All non-empty subsets of frequent itemset must be frequent. While all the supersets of infrequent itemset must be infrequent. This is the key concept of Apriori Algorithm.

How to Use

After extraction of the `.zip` file, follow these steps:

1. Make sure that the `categories.txt` and `main.py` are in the same directory. For no errors, make sure that the `readpath` variable within the code is set to the *Relative Path* of the `categories.txt` file.
2. Place your transaction data in the categories.txt file. Each line should represent a transaction, and items within a transaction should be separated by semicolons (";").
3. Similarly, the `writepath` variable can also be modified to set the directory into which the `.txt` files will be written.
4. Open the `main.py` file in any Python IDE or VSCode.
5. Run the `main.py` and open the directory corresponding to `writepath`.
6. The output of `main.py` will be displayed in the terminal and two files; `patterns_1.txt` and `patterns_all.txt` will be written to `writepath` directory.

How it Works

The overall Time Complexity of the algorithm is `O(n^2*logn)`.

- Code begins with initializing values of variable `readpath` and `writepath`. These two variables are the corresponding read and write directory.
- The `minSup` variable contains the value of the minimum support value for comparisions and determining the frequent itemsets. This value can be modified.
- Itemset data from `categories.txt` is read and extracted to a variable `dataBase` using the `readFile()` function.

```
def readFile(readPath) -> list:
    with open(readPath, 'r') as rf:
        file_contents = rf.read()
        itemsets = file_contents.splitlines()
        items = list()

        for line in itemsets:
            items.append(set(line.split(';')))
```

- The function uses `split()` and `splitlines()` method to parse and extract data from the given dataset. Each element inserted into the `items` list is type-casted to `set` class.
- All the elements from `dataBase` are added into the set `total_categories` to make k = 1 i.e. length 1 candidate set using `addSet()` function.

> ***Candidate set -***
>
> A candidate set is collection of all length k itemsets which are a candidate for k+1 frequent itemsets.

- The `apriori()` function follows a recursive algorithm where initially the entry condition is checked.

```
  def apriori(input, data, minSup, k, freqSet, countSet):
      if len(input) <= 1:
          return freqSet, countSet
```

- The entry condition checks whether the `Ck` itemset, which is the length k candidate itemset, has only one or less elements. When the condition has been met, the `freqSet` and `countSet` lists are returned. These lists store the total frequent itemsets and their support counts.
- Within the `Ck` itemset, the support count is checked against `minSup` value and all the elements higher than the `minSup` value are made into another itemset called `Lk`. This `Lk` itemset serves as the `candidate set` for the k+1 length itemset.
- All the elements of `dataBase` are of `set` class that means taking union of these k length sets will give k or higher length itemsets.

```
```

```
def makeUnion(input: list, k: int) -> list:
    thisList = list()
    for item in input:

        for obj in input:
            x = item.union(obj)

            if len(x) == k:
                checkList(x, thisList)

  return thisList
```

- The `makeUnion()` functions takes two elements from the same k-1 length itemset and finds the union set containing all k length itemset.
- Since, all the itemsets are stored in `sets`, all the itemsets will be unique and their support count can be calculated in `dataBase` by using `issubset()` method of `set` class and `countSup()` function.

```
if k == 1:
    path = 'patterns_1.txt'
    len_1_set = addlist(countSet)
    writeFile(path, len_1_set, total(countSet))
```

- The above block of code takes all length 1 frequent itemsets and writes them into a `patterns_1.txt` file using the `writeFile()` function.
- Apriori algorithm is run recursively until the `Lk` has 1 or lesser elements and finally all the frequent itemsets and their support counts are written in `patterns_all.txt` file.

Conclusion

This Apriori algorithm is costly in terms of time efficiency and is a heavyweight algorithm with *Time Complexity* = `O(n^2*logn)` and *Space Complexity* ~ `O(n^2)`

Optimizations

The code can be further optimized using an iterative approach and reusing the data structures already present i.e. modifying the contents of data structures to reduce the Space Complexity upto `O(1)` while the Time Complexity can be reduced till `O(n*logn)`. It can also be optimized by using external libraries like `itertools` which offers a `combinations` function that can make candidate itemsets efficiently in `O(n)` time. There are also data structures from external libraries that can help reduce Space Complexity by considerable amount, one such is `defaultdict` from `collections` library. The Space Complexity of the algorithm can be reduced to `O(n)` by using this data structure.

References

- GeeksForGeeks

- JavatPoint
- Basic Writing and Formatting syntax
- Markdown CheatSheet