CS470 Homework 2

Jaekyum Kim

2020-02-20

Collaboration Statement THIS CODE IS MY OWN WORK, IT WAS WRITTEN WITHOUT CONSULTING A TUTOR OR CODE WRITTEN BY OTHER STUDENTS - Jaekyum Kim

1 Introduction

The program is executable with 3 parameters: the name of the input dataset file, the threshold of minimum support count, and the name of the output file. The program creates an output a file that contains all the frequent itemsets together with their support counts. The test dataset is a synthetic dataset that contains 100,000 transactions with an average size of 10 items from a set of 1000 distinct items.

2 How algorithm runs

```
// apriori.py
oneTransaction =  infile.read().split()
unsortedTransac = list(map(int, oneTransaction)) # converts to a
                                  one-liner string of transaction
candidates_list = sorted(list(dict.fromkeys(unsortedTransac)))
```

The algorithm first reads the opened file in a single string and uses the split function to divide into a list of strings by using list(map())) functions. The line 3 removes duplicates and sorts the list in numerical order.

```
// apriori.py

for eachcand in candidates_list:
    count = unsortedTransac.count(eachcand)
    if count >= minsup:
        whole[eachcand] = count
        L1.append(eachcand)
```

Using the iterator in python built-in function, each candidate of the whole itemsets are counted by using list.count() function to create a dictionary set of number of frequencies per each of the individual itemsetes. One was put into the final result set, and the other was put into the list to create the next set of candidates.

```
// apriori.py

for eachLine in linesReadString:
    convertedList = set([int(i) for i in eachLine.split()])

    if len(convertedList.intersection(set(L1))) == 0:
        linesReadString.remove(eachLine)
```

This function was created to rule out the transaction that was not needed. To select the ones that are needed, I retrieved the one line string list of all candidates that are above the minimum support count and compared it per string to see if there is any intersection. If there is none, then that particular line of string will not be used later, so it is safe to delete it.

```
 // apriori.py

 comb_length=2
d=whole
while True:
    proceednxt = False
    L2=[]
    d2={} #temp dictionary var i made

    for eachset in d: # breakdown into a one-liner
        if type(eachset) != int: #exception for one digits
            for eachitem in eachset:#rest of the receiving sets
                L2.append(eachitem)
        else:
            L2.append(eachset)

    L2 = list(dict.fromkeys(L2))#remove duplicates
    #print("L2: ", L2)

    for eachLine in linesReadString:
        convertedList = set([int(i) for i in eachLine.split()])
        #each line of transaction is converted into a list of
                                          integers per iterated per
                                           line

        modifed_list = sorted(list(convertedList.intersection(set(
                                          L2))))
        #list of candidates were formed by intersecting a line of
                                          transaction and
        #a set of possible candidates from previous iteration and
                                          sorted into order
        #("if not going to use all the item in a line of
                                          transaction then dont use
                                           it")

        C3 = list(itertools.combinations(modifed_list, comb_length)
                                          )
        #this line unions and creats combinations of candidates and
                                           puts them into the list

        for eachcand in C3:
            index = frozenset(eachcand)
            d2[index] = d2.get(index, 0) + 1
            if d2[index] >= minsup:
                proceednxt = True

    for key in [key for key in d2 if int(d2[key]) < minsup]:
        del d2[key]#deletes those that didnt meet the minimum
                                          support count and proceed
                                           to next level


    whole.update(d2)
    if proceednxt == False:
        break;
    d = d2
    comb_length += 1
```

This loop executes the loop phase to continuously find the frequent set of items until there is none that can be find. The first for loop breaks down a newly made dictionary's keys and append those into a one strip of string. The next for loop checks each line of modified transactional records and compares with the list of sets that were created by combination function that was imported. For each line, I compared each of the newly created sets and checked the number of counts. The variable modifiedlist already ruled out unnecessary items so It was safe to just make a combinations out of the changed list. I did not need to count each element. I counted as i read along the transactions.

3 Algorithm's Performance

For 300 min supp counts the algorithm ran around 117.4040 seconds and for 400, it was 43.93 sec. and 500 mark, it was around 22.6493 seconds and at 1000 mark the fastest it got was 14.6723. As the min sup rises, the algorithm is faster.