Report

November 17, 2023

0.1 Homework 2: Discovery of Frequent Itemsets and Association Rules

Group 56: Abdou Mustafa Saleh, wFarmaki Athanasia

Date: 17/11/2023 The scope of this assignment is to discover association rules between itemsets in a sales transaction database. The task involves two phases: 1) Find frequent itemsets with support at least s 2) Generating association rules with some confidence c from the itemsets found in the first step

[99]:
```
from collections import defaultdict
from itertools import chain, combinations
from tqdm import tqdm
```

[118]:
```
def getUnion(itemSet, length):
    return set([i.union(j) for i in itemSet for j in itemSet if len(i.union(j))␣
 ↪== length])

def getAprioriAlg(itemSetList, itemSetSet, minSup):
    C1List = getItemSetFromList(itemSetList) ## frequest 1 itemlist
    globalFreqItemSet = dict() # The ` ` is used to create an empty set in␣
 ↪Python. In the code, it is

    globalItemSetWithSup = defaultdict(int) ## total counts of each item "count␣
 ↪of existance"
    L1ItemSet = getMinSup(C1List,itemSetSet,minSup,globalItemSetWithSup) ##␣
 ↪return first level of freq item set
    currentLSet = L1ItemSet
    k = 2
    minSup = 0.01
    while currentLSet:
        print("k",k)
        globalFreqItemSet[k-1] = currentLSet ## for k = 1 "size of set is 1"
        candidateSet = getUnion(currentLSet, k) ## create  itemsets for size k ␣
 ↪from the current list
        # candidateSet = pruning(candidateSet, currentLSet, k-1)
        currentLSet =␣
 ↪getMinSup(candidateSet,itemSetSet,minSup,globalItemSetWithSup) ## return␣
 ↪only sets above min support
```

```
        k = k +1

    return globalFreqItemSet, globalItemSetWithSup

def pruning(candidateSet, prevFreqSet, length):
    print("=--------------------" )
    print("start pruning : ", len(candidateSet) )
    tempCandidateSet = candidateSet.copy()
    for item in candidateSet:
        subsets = combinations(item, length)
        for subset in subsets:
            # if the subset is not in previous K-frequent get, then remove the␣
 ↪set
            if(frozenset(subset) not in prevFreqSet):
                tempCandidateSet.remove(item)
                break

    print("after pruning : ",len(candidateSet) )
    return tempCandidateSet

def getPossibleSets(s):
    result = []
    for r in range(1, len(s)):
        result.extend(combinations(s, r))
    return result

def associationRule(freqItemSet, itemSetWithSup, minConf):
    rules = []
    for k, itemSet in freqItemSet.items():
        for item in itemSet:
            # print("---------")
            # print(item)
            subsets = getPossibleSets(item)
            for s in subsets:
                confidence = float(
                    itemSetWithSup[item] / itemSetWithSup[frozenset(s)])
                if(confidence > minConf):
                    rules.append([set(s), set(item.difference(s)), confidence])
    return rules

def getItemSetFromList(itemSetList):
    tempItemSet = set()
    for itemSet in itemSetList:
        for item in itemSet:
            tempItemSet.add(frozenset([item]))

    return tempItemSet
```

```
def getMinSup(itemSet, itemSetSet, minSup, globalItemSetWithSup):
    freqItemSet = set()
    localItemSetWithSup = defaultdict(int)
    print("count items occurence")
    num_candidates = len(itemSet)
    ## count the occurance of a set in all itemsets // we put it in global␣
 ↪store to be able to use it  for confidence calc at the end.
    for item in tqdm(itemSet):
        for itemSetNested in itemSetSet:
            if item.issubset(itemSetNested):
            # if item - itemSetNested:
                globalItemSetWithSup[item] += 1
                localItemSetWithSup[item] += 1

    print("calculating support")
    ## calc the support for each item and add it to fequest item list.
    for item, supCount in localItemSetWithSup.items():
        support = float(supCount / len(itemSetList))
        if(support >= minSup):
            freqItemSet.add(item)

    return freqItemSet
```

0.2 Read Dataset

[114]:
```
all_baskets = []
all_baskets_set = []
with open("T10I4D100K.dat","r") as f:
    while True:
        line = f.readline().strip("\n")
        all_baskets.append(line)
        all_baskets_set.append(line)
        if not line:
            break
for i, basket in enumerate(all_baskets):
    all_baskets[i] = [int(b) for b in basket.split(" ") if b != ""]
    all_baskets_set[i] = set(all_baskets[i] )
```

0.3 Run A-priori algorithm

[119]:
```
itemSetList = all_baskets
minSup = 0.01
globalFreqItemSet, globalItemSetWithSup = getAprioriAlg(all_baskets,␣
 ↪all_baskets_set, minSup)
```

```
count items occurence
```

```
100%|
                              | 870/870 [00:05<00:00,
164.70it/s]

calculating support
k 2
count items occurence

100%|
                            | 70125/70125 [07:00<00:00,
166.95it/s]

calculating support
k 3
count items occurence

100%|
                                | 4/4 [00:00<00:00,
158.84it/s]

calculating support
k 4
count items occurence

0it [00:00, ?it/s]

calculating support
```

[135]:
```python
for k,v in globalFreqItemSet.items():
    print("**********************")
    v = [set(value) for value in v]
    itemsets = ','.join(map(str, v))
    print(f"Frequent itemsets of size {k}: {itemsets}")
```

```
**********************
Frequent itemsets of size 1: {675},{33},{701},{274},{617},{561},{97},{185},{862}
,{104},{210},{823},{334},{618},{285},{25},{688},{111},{814},{874},{73},{496},{85
},{541},{707},{336},{765},{381},{276},{51},{75},{913},{653},{275},{296},{335},{6
31},{918},{515},{403},{170},{663},{116},{694},{841},{820},{177},{523},{919},{115
},{69},{41},{500},{280},{17},{832},{829},{157},{401},{638},{574},{38},{982},{326
},{129},{470},{984},{357},{308},{57},{789},{952},{279},{522},{132},{684},{234},{
810},{55},{941},{540},{946},{578},{843},{720},{201},{32},{120},{94},{325},{548},
{749},{890},{236},{105},{649},{947},{944},{163},{196},{554},{110},{378},{950},{9
14},{168},{368},{428},{173},{800},{788},{804},{572},{538},{227},{960},{486},{620
},{413},{630},{661},{935},{319},{504},{682},{581},{346},{259},{614},{343},{422},
{52},{348},{967},{242},{366},{846},{290},{147},{362},{744},{706},{72},{710},{469
},{867},{834},{597},{591},{769},{70},{579},{853},{205},{658},{310},{217},{424},{
240},{6},{740},{460},{35},{490},{716},{414},{793},{332},{322},{550},{888},{204},
{468},{546},{819},{932},{795},{354},{394},{912},{923},{526},{893},{634},{140},{1
43},{126},{964},{527},{458},{641},{21},{510},{989},{758},{145},{798},{450},{10},
```

```
{883},{774},{949},{58},{54},{192},{797},{639},{112},{906},{489},{37},{886},{611}
,{884},{970},{392},{171},{214},{966},{628},{487},{521},{792},{419},{995},{598},{
718},{738},{567},{594},{472},{377},{752},{461},{5},{988},{623},{583},{1},{39},{2
83},{571},{563},{736},{411},{673},{31},{48},{151},{576},{978},{529},{704},{266},
{100},{921},{790},{899},{385},{405},{28},{509},{805},{854},{516},{895},{975},{13
0},{350},{815},{778},{440},{122},{27},{910},{8},{991},{897},{45},{784},{775},{17
5},{154},{981},{780},{351},{605},{573},{606},{125},{4},{735},{294},{181},{530},{
258},{93},{825},{676},{183},{68},{593},{513},{692},{632},{207},{844},{265},{239}
,{162},{746},{387},{956},{803},{197},{708},{161},{438},{448},{763},{928},{494},{
208},{826},{937},{651},{812},{678},{12},{860},{871},{449},{963},{429},{766},{198
},{43},{674},{733},{480},{71},{809},{229},{998},{885},{517},{423},{577},{361},{6
86},{900},{511},{948},{309},{665},{782},{534},{857},{722},{427},{78},{477},{580}
,{390},{859},{90},{569},{471},{687},{887},{992},{373},{349},{878},{600}
**********
Frequent itemsets of size 2: {704, 825},{217, 346},{825, 39},{227, 390},{789,
829},{722, 390},{368, 829},{704, 39},{368, 682}
**********
Frequent itemsets of size 3: {704, 825, 39}
```

0.4 Optional Task: Find association rules

[121]:
```
print("Extracting association rules")
confidence_threshold = 0.6
rules = associationRule(globalFreqItemSet, globalItemSetWithSup,␣
 ↪confidence_threshold)
rules.sort(key=lambda x: x[2], reverse=True) ## sort by confidence
rules
```

```
Extracting association rules
```

[121]:
```
[[{704, 825}, {39}, 0.9392014519056261],
 [{39, 704}, {825}, 0.9349593495934959],
 [{39, 825}, {704}, 0.8719460825610783],
 [{704}, {39}, 0.617056856187291],
 [{704}, {825}, 0.6142697881828316]]
```

0.5 Rules found

[122]:
```
rules
```

[122]:
```
[[{704, 825}, {39}, 0.9392014519056261],
 [{39, 704}, {825}, 0.9349593495934959],
 [{39, 825}, {704}, 0.8719460825610783],
 [{704}, {39}, 0.617056856187291],
 [{704}, {825}, 0.6142697881828316]]
```

[ ]:
```

```