CS412: Introduction to Data Mining

Hanfei Lin

Nov. 5th, 2017

**Question 1**

**Answers:**

a. 4, 4, 4

b. 1, 0, 1

**Question 2**

**Answers:**

6

4

1

2

5

4, 5

1, 4

5, 6

1, 2

2, 4

1, 4, 5

1, 2, 4

4, 5, 6

1, 2, 4, 5

**Code:**

To solve this problem, I extend the *apriori* algorithm developed in MP2.

Here is the code:

# Question 2

# -\*- coding: utf-8 -\*-

import pandas as pd

import numpy as np

import math

#import data

TDB = []

with open('data') as data\_file:

index = 0

for record in data\_file.readlines():

record = record.replace(" ","").strip()

if index == 0:

support = record.split(',')[0]

outlier\_resilience = record.split(',')[1]

else:

TDB.append(record.split(','))

index += 1

def outlier\_apriori(minsup, maxoutl):

DBsize = len(TDB)

# k is the length for FP(number of element in a FP)

k = 1

# a Python dictionary that saves: FP -> support,

# each freqItemset contains only FPs with same length

freqItemset = {}

# a Python list that saves freqItemset of all length

freqItemsetPool = []

# Find the 1-length freqItemset of FPs.

index = 1

for T in TDB:

for item in T:

if freqItemset.has\_key((item,)):

if index not in freqItemset[(item,)]:

freqItemset[(item,)].append(index)

else:

freqItemset[(item,)] = []

freqItemset[(item,)].append(index)

index += 1

# Remove 1-length patterns with supports that less than minsup

temp = {}

for itemset in freqItemset:

if float(len(freqItemset[itemset]))/ float(DBsize) >= minsup:

temp[itemset] = freqItemset[itemset]

freqItemset = temp

# Save 1-length freqItemset into freqItemsetPool

freqItemsetPool.append(freqItemset)

# Perform Apriori algorithm until freqItemset is empty

# I.e. no larger FPs can be found.

while freqItemset:

# Accumulate the length of FP

k += 1

# Find all k-length FP candidates from (k-1)-length FP

candidates = set()

freqSets = list(freqItemset.keys())

for i in range(len(freqSets) - 1):

for j in range(i + 1, len(freqSets)):

temp = set(freqSets[i] + freqSets[j])

if len(temp) == k:

candidates.add(tuple(sorted(tuple(temp))))

candidates = sorted(list(candidates))

# Find k-length freqItemset of FPs.

# Similar to the above 1-length situation.

freqItemset = {}

index = 1

for T in TDB:

for candidate in candidates:

if exist(candidate, T):

if freqItemset.has\_key(candidate):

if index not in freqItemset[candidate]:

freqItemset[candidate].append(index)

else:

freqItemset[candidate] = []

freqItemset[candidate].append(index)

index += 1

# Remove k-length patterns with supports that less than minsup

temp = {}

for itemset in freqItemset:

if float(len(freqItemset[itemset])) / float(DBsize) >= minsup:

temp[itemset] = freqItemset[itemset]

freqItemset = temp

# Remove itemsets that are not outlier resilient

temp = {}

for itemset in freqItemset:

if outlier(itemset, freqItemset[itemset], TDB, maxoutl, minsup):

temp[itemset] = freqItemset[itemset]

freqItemset = temp

# Save k-length freqItemset into freqItemsetPool

if freqItemset:

freqItemsetPool.append(freqItemset)

return freqItemsetPool

# Save 1-length freqItemset into freqItemsetPool

freqItemsetPool.append(freqItemset)

# Whether a pattern exist in a sequence

def exist(candidate, sequence):

e = True

for i in candidate:

if i not in sequence:

e = False

return e

# Whether a pattern is outlier resilient

def outlier(itemset, freq, TDB, maxoutl, minsup):

#To satisfy outlier resilience, at least one of max\_length subsequence should contains itemset

max\_length = int(math.floor(maxoutl \* len(itemset) + len(itemset)))

count = []

for index in freq:

seq = TDB[index - 1]

front = 0

tail = max\_length

if tail > len(seq):

if exist(itemset, seq):

if index not in count:

count.append(index)

while tail <= len(seq):

subseq = seq[front : tail]

if exist(itemset, subseq):

if index not in count:

count.append(index)

front += 1

tail += 1

if float(len(count)) / len(TDB) >= minsup:

return True

return False

ans = outlier\_apriori(float(support), float(outlier\_resilience))

#Output the result

f = open("hanfeil2-HW3.txt", "w")

for item in ans:

for i in item.keys():

i = list(i)

i = [int(x) for x in i]

i.sort()

f.write(', '.join(str(s) for s in i) + "\n")

f.close()