CSE-3019 DATA MINING

LAB: L47 +L48

Name: Nimit Agrawal

REG. NO: 16BCE0170

Lab-4

Apply Apriori Algorithm to a dataset and perform market basket analysis with varying support and confidence values.

Output should have the association rule with higher confidence.

Dataset:

https://www.kaggle.com/shazadudwadia/supermarket

MILK, BREAD, BISCUIT

Numeric

items list

BREAD,TEA,BOURNVITA 11%

COFFEE,COCK,BISCUIT,CORNFL... 11%

BREAD,COFFEE,SUGER 11%

BREAD,MILK,BISCUIT,CORNFLA... 5%

Other (12) 63%

 Valid ■
 19
 100%

 Mismatched ■
 0
 0%

 Missing ■
 0
 0%

 Unique
 16
 16

 Most Common
 BREAD, TEA, I
 11%

Dataset

1	MILK,BREAD,BISCUIT
2	BREAD,MILK,BISCUIT,CORNFLAKES
3	BREAD,TEA,BOURNVITA
4	JAM,MAGGI,BREAD,MILK
5	MAGGI,TEA,BISCUIT
6	BREAD,TEA,BOURNVITA
7	MAGGI,TEA,CORNFLAKES
8	MAGGI,BREAD,TEA,BISCUIT
9	JAM,MAGGI,BREAD,TEA
10	BREAD,MILK
11	COFFEE,COCK,BISCUIT,CORNFLAKES
12	COFFEE,COCK,BISCUIT,CORNFLAKES
13	COFFEE,SUGER,BOURNVITA
14	BREAD,COFFEE,COCK
15	BREAD,SUGER,BISCUIT
16	COFFEE,SUGER,CORNFLAKES
17	BREAD,SUGER,BOURNVITA
18	BREAD,COFFEE,SUGER
19	BREAD,COFFEE,SUGER
20	TEA,MILK,COFFEE,CORNFLAKES

Apriori Code:

```
import sys
import operator
from itertools import chain, combinations
from collections import defaultdict
from optparse import OptionParser
```

```
def subsets(arr):  return\ chain(*[combinations(arr,\ i+1)\ for\ i,\ a\ in\ enumerate(arr)])
```

```
def returnItemsWithMinSupport(itemSet, transactionList, minSupport, freqSet):
     _itemSet = set()
    localSet = defaultdict(int)
     for item in itemSet:
          for transaction in transactionList:
               if item.issubset(transaction):
                    freqSet[item] += 1
                    localSet[item] += 1
     for item, count in localSet.items():
          support = float(count)/len(transactionList)
          if support >= minSupport:
               _itemSet.add(item)
    return _itemSet
def joinSet(itemSet, length):
  return set([i.union(j) for i in itemSet for j in itemSet if len(i.union(j)) == length])
def getItemSetTransactionList(data_iterator):
  transactionList = list()
  itemSet = set()
  for record in data_iterator:
     transaction = frozenset(record)
     transactionList.append(transaction)
     for item in transaction:
```

```
itemSet.add(frozenset([item]))
                                             # Generate 1-itemSets
  return itemSet, transactionList
def runApriori(data_iter, minSupport, minConfidence):
  itemSet, transactionList = getItemSetTransactionList(data_iter)
  freqSet = defaultdict(int)
  largeSet = dict()
  # Global dictionary which stores (key=n-itemSets,value=support)
  # which satisfy minSupport
  assocRules = dict()
  # Dictionary which stores Association Rules
  oneCSet = returnItemsWithMinSupport(itemSet,
                        transactionList,
                        minSupport,
                        freqSet)
  currentLSet = oneCSet
  k = 2
  while(currentLSet != set([])):
    largeSet[k-1] = currentLSet
    currentLSet = joinSet(currentLSet, k)
    currentCSet = returnItemsWithMinSupport(currentLSet,
                             transactionList,
                             minSupport,
                             freqSet)
    currentLSet = currentCSet
```

```
def getSupport(item):
       """local function which Returns the support of an item"""
       return float(freqSet[item])/len(transactionList)
  toRetItems = []
  for key, value in largeSet.items():
     toRetItems.extend([(tuple(item), getSupport(item))
                 for item in value])
  toRetRules = []
  for key, value in list(largeSet.items())[1:]:
     for item in value:
       _{\text{subsets}} = \text{map}(\text{frozenset}, [x \text{ for } x \text{ in subsets(item)}])
       for element in _subsets:
          remain = item.difference(element)
          if len(remain) > 0:
            confidence = getSupport(item)/getSupport(element)
            if confidence >= minConfidence:
               toRetRules.append(((tuple(element), tuple(remain)),
                           confidence))
  return toRetItems, toRetRules
def printResults(items, rules):
  for item, support in sorted(items, key=operator.itemgetter(1)):
     print("item: %s , %.3f" % (str(item), support))
  print("\n----- RULES:")
```

k = k + 1

```
for rule, confidence in sorted(rules, key=operator.itemgetter(1)):
     pre, post = rule
     print("Rule: %s ==> %s , %.3f" % (str(pre), str(post), confidence))
def dataFromFile(fname):
     """Function which reads from the file and yields a generator"""
     file_iter = open(fname, 'rU')
     for line in file_iter:
          line = line.strip().rstrip(',')
                                                   # Remove trailing comma
          record = frozenset(line.split(','))
          yield record
if __name__ == "__main__":
  inFile=dataFromFile("GroceryStoreDataSet.csv")
  minSupport = 0.15
  minConfidence = 0.6
  items, rules = runApriori(inFile, minSupport, minConfidence)
  printResults(items, rules)
```

```
apriori.py - C:\Nimit\apriori.py (2.7.14)
                                                                                                                                                                                                                         File Edit Format Run Options Window Help
import operator
                                                                                                                           Python 2.7.14 Shell
from itertools import chain, combinations
                                                                                                                           File Edit Shell Debug Options Window Help
from collections import defaultdict
from optparse import OptionParser
                                                                                                                           Python 2.7.14 (v2.7.14:84471935ed, Sep 16 2017, 20:25:58) [MSC v.1500 64 bit (AM 📥
                                                                                                                           D64)1 on win32
                                                                                                                           Type "copyright", "credits" or "license()" for more information.
def subsets(arr):
    return chain(*[combinations(arr, i + 1) for i, a in enumerate(arr)])
                                                                                                                                         RESTART: C:\Nimit\apriori.py ====
                                                                                                                           item: ('BISCUIT',) , 0.150
                                                                                                                           item: ('"MAGGI',) , 0.150
item: ('BOURNVITA"', '"BREAD') , 0.150
def returnItemsWithMinSupport(itemSet, transactionList, minSupport, freqSet):
                                                                                                                           item: ('"COFFEE', 'CORNFLAKES"'), 0.150
item: ('BISCUIT', 'CORNFLAKES"'), 0.150
        localSet = defaultdict(int)
                                                                                                                          item: ('"MAGGI', 'TEA') , 0.150
item: ('COFFEE', "BREAD') , 0.150
item: ("COFFEE', 0.200
item: ('BISCUIT"',) , 0.200
         for item in itemSet:
                  for transaction in transactionList:
                           if item.issubset(transaction):
                                     freqSet[item] += 1
                                                                                                                           item: ('SUGER',) , 0.200
                                     localSet[item] += 1
                                                                                                                           item: ('BREAD',) , 0.200
                                                                                                                           item: ('COFFEE',) , 0.200
item: ('BOURNVITA"',) , 0.200
         for item, count in localSet.items():
                 support = float(count)/len(transactionList)
                                                                                                                           item: ('TEA',) , 0.250
item: ('CORNFLAKES"',) , 0.300
                  if support >= minSupport:
                                                                                                                           item: ('"BREAD',) , 0.450
                           _itemSet.add(item)
                                                                                                                                         ----- RULES:
         return _itemSet
                                                                                                                           Rule: ('TEA',) ==> ('"MAGGI',) , 0.600
                                                                                                                           Rule: ('BOURNVITA"',) ==> ('"BREAD',) , 0.750
                                                                                                                           Rule: ("MOSFEE",) ==> ("CORNFLARES"",) , 0.750
Rule: ("COFFEE",) ==> ("BREAD",) , 0.750
Rule: ("BISCUIT",) ==> ("CORNFLARES"",) , 1.000
def joinSet(itemSet, length):
    return set([i.union(j) for i in itemSet for j in itemSet if len(i.union(j)) == length])
                                                                                                                           Rule: ('"MAGGI',) ==> ('TEA',) , 1.000
def getItemSetTransactionList(data_iterator):
    transactionList = list()
    itemSet = set()
    for record in data_iterator:
         transaction = frozenset(record)
         transactionList.append(transaction)
             itemSet.add(frozenset([item]))
                                                               # Generate 1-itemSets
    return itemSet, transactionList
                                                                                                                                                                                                                   Ln: 29 Col: 4
ief runApriori(data iter, minSupport, minConfidence):
   Python 2.7.14 Shell
   File Edit Shell Debug Options Window Help
   Python 2.7.14 (v2.7.14:84471935ed, Sep 16 2017, 20:25:58) [MSC v.1500 64 bit (AMD64)] on win32
   Type "copyright", "credits" or "license()" for more information.
    >>>
                  ====== RESTART: C:\Nimit\apriori.pv =====
    item: ('BISCUIT',) , 0.150
   item: ('"MAGGI',) , 0.150
item: ('BOURNVITA"', '"BREAD') , 0.150
   item: ('BOCKNVIA', 'BREAD'), 0.150
item: ('BISCUIT', 'CORNFLAKES"), 0.150
item: ('MAGGI', 'TEA'), 0.150
item: ('COFFEE', '"BREAD'), 0.150
    item: ('"COFFEE',) , 0.200
    item: ('BISCUIT"',) , 0.200
    item: ('SUGER',) , 0.200
    item: ('BREAD',) , 0.200
   item: ('COFFEE',) , 0.200
item: ('BOURNVITA"',) , 0.200
    item: ('TEA',) , 0.250
    item: ('CORNFLAKES"',) , 0.300
    item: ('"BREAD',) , 0.450
    Rule: ('TEA',) ==> ('"MAGGI',) , 0.600
    Rule: ('BOURNVITA"',) ==> ('"BREAD',) , 0.750
    Rule: ('"COFFEE',) ==> ('CORNFLAKES"',) , 0.750
   Rule: ('COFFEE',) ==> ('BREAD',) , 0.750
Rule: ('BISCUIT',) ==> ('CORNFLAKES"',) , 1.000
   Rule: ('"MAGGI',) ==> ('TEA',) , 1.000
    >>>
```

apriori.py - C:\Nimit\apriori.py (2.7.14) O X File Edit Format Run Options Window Help import sys import operator from itertools import chain, combinations from collections import defaultdict from optparse import OptionParser def subsets(arr): return chain(*[combinations(arr, i + 1) for i, a in enumerate(arr)]) def returnItemsWithMinSupport(itemSet, transactionList, minSupport, freqSet): itemSet = set() localSet = defaultdict(int) for item in itemSet: for transaction in transactionList: if item.issubset(transaction): freqSet[item] += 1 localSet[item] += 1 for item, count in localSet.items(): support = float(count)/len(transactionList) if support >= minSupport: itemSet.add(item) return _itemSet def joinSet(itemSet, length): return set([i.union(j) for i in itemSet for j in itemSet if len(i.union(j)) == length]) def getItemSetTransactionList(data iterator): transactionList = list() itemSet = set() for record in data iterator: transaction = frozenset(record) transactionList.append(transaction) for item in transaction: # Generate 1-itemSets itemSet.add(frozenset([item])) return itemSet, transactionList def runingiori (data iter. minSunnort. minConfidence):

Support:0.2

Confidence:0.7

```
apriori.py - C:\Nimit\apriori.py (2.7.14)
                                                                                                                                                                              Ö
File Edit Format Run Options Window Help
            Frank Trons(Tridace(Torm))\trus(eranemorenmane)
    toRetItems = []
                                                                                                  Python 2.7.14 Shell
    for key, value in largeSet.items():
                                                                                                  File Edit Shell Debug Options Window Help
        toRetItems.extend([(tuple(item), getSupport(item))
                          for item in value])
                                                                                                  Python 2.7.14 (v2.7.14:84471935ed, Sep 16 2017, 20:25:58) [MSC v.1500 64 bit (AM 4
    toRetRules = []
                                                                                                  Type "copyright", "credits" or "license()" for more information.
    for key, value in list(largeSet.items())[1:]:
        for item in value:
                                                                                                       ----- RESTART: C:\Nimit\apriori.py
            subsets = map(frozenset, [x for x in subsets(item)])
                                                                                                  item: ('"COFFEE',) , 0.200
            for element in subsets:
                                                                                                  item: ('BISCUIT"',) , 0.200
               remain = item.difference(element)
                                                                                                  item: ('SUGER',) , 0.200
               if len(remain) > 0:
                                                                                                  item: ('BREAD',) , 0.200
                   confidence = getSupport(item)/getSupport(element)
                                                                                                  item: ('COFFEE',) , 0.200
                   if confidence >= minConfidence:
                                                                                                  item: ('BOURNVITA"',) , 0.200
                       toRetRules.append(((tuple(element), tuple(remain)),
                                                                                                  item: ('TEA',) , 0.250
                                          confidence))
                                                                                                  item: ('CORNFLAKES"',) , 0.300
    return toRetItems, toRetRules
                                                                                                  item: ('"BREAD',) , 0.450
                                                                                                  ----- RULES:
def printResults(items, rules):
    for item, support in sorted(items, key=operator.itemgetter(1)):
       print ("item: %s , %.3f" % (str(item), support))
    print("\n----- RULES:")
    for rule, confidence in sorted(rules, key=operator.itemgetter(1)):
        pre, post = rule
        print("Rule: %s ==> %s , %.3f" % (str(pre), str(post), confidence))
def dataFromFile(fname):
        """Function which reads from the file and yields a generator"""
        file_iter = open(fname, 'rU')
        for line in file iter:
               line = line.strip().rstrip(',')
                                                                     # Remove trailing comma
               record = frozenset(line.split(','))
               yield record
if __name__ == "__main__":
    inFile=dataFromFile("GroceryStoreDataSet.csv")
    minSupport = 0.2
    minConfidence = 0.7
    items, rules = runApriori(inFile, minSupport, minConfidence)
                                                                                                                                                                         Ln: 16 Col: 4
    printResults(items, rules)
```

Support: 0.12

Confidence: 0.5

```
aprior.py - C\Nimit\aprior.py (2.7.14)
File Edit Format Run Options Window Help
            remark tronsitredocalracmilisteriatempasarameras
                                                                                         Python 2.7.14 Shell
    toRetItems = []
    for key, value in largeSet.items():
                                                                                         File Edit Shell Debug Options Window Help
       toRetItems.extend([(tuple(item), getSupport(item))
                                                                                         Python 2.7.14 (v2.7.14:84471935ed, Sep 16 2017, 20:25:58) [MSC v.1500 64 bit (AM 4
                           for item in value))
                                                                                         Type "copyright", "credits" or "license()" for more information.
    toRetRules = []
    for key, value in list(largeSet.items())[1:]:
                                                                                                    RESTART: C:\Nimit\apriori.py
        for item in value:
                                                                                         item: ('BISCUIT',) , 0.150
            _subsets = map(frozenset, [x for x in subsets(item)])
                                                                                         item: ('"MAGGI',) , 0.150
item: ('BOURNVITA"', '"BREAD') , 0.150
            for element in subsets:
                remain = item.difference(element)
                                                                                         item: ('"COFFEE', 'CORNFLAKES"') , 0.150
                if len(remain) > 0:
                                                                                         item: ('BISCUIT', 'CORNFLAKES"') , 0.150
                   confidence = getSupport(item)/getSupport(element)
                                                                                         item: ('"MAGGI', 'TEA') , 0.150
                    If confidence >= minConfidence:
                                                                                         item: ('COFFEE', '"BREAD') , 0.150
                       toRetRules.append(((tuple(element), tuple(remain)),
                                                                                         item: ('"COFFEE',) , 0.200
                                          confidence))
                                                                                         item: ('BISCUIT"',) , 0.200
    return toRetItems, toRetRules
                                                                                         item: ('SUGER',) , 0.200
                                                                                         item: ('BREAD',) , 0.200
                                                                                         item: ('COFFEE',) , 0.200
                                                                                         item: ('BOURNVITA"',) , 0.200
def printResults(items, rules):
                                                                                         item: ('TEA',) , 0.250
    for item, support in sorted(items, key=operator.itemgetter(1)):
                                                                                         item: ('CORNFLAKES"',) , 0.300
       print("item: %s , %.3f" % (str(item), support))
                                                                                         item: ('"BREAD',) , 0.450
    print ("\n----- RULE5:")
    for rule, confidence in sorted(rules, keymoperator.itemgetter(1)):
                                                                                         ----- RULES:
      pre, post = rule
                                                                                         Rule: ('CORNFLARES"',) ==> ('"COFFEE',) , 0.500
        print("Rule: %s ==> %s , %.3f" % (str(pre), str(post), confidence))
                                                                                         Rule: ('CORNFLAKES"',) ==> ('BISCUIT',) , 0.500
                                                                                         Rule: ('TEA',) ==> ('"MAGGI',) , 0.600
                                                                                         Rule: ('BOURNVITA"',) ==> ('"BREAD',) , 0.750
def dataFromFile(fname);
                                                                                         Rule: ('"COFFEE',) ==> ('CORNFLAKES"',) , 0.750
        """Function which reads from the file and yields a generator"""
                                                                                         Rule: ('COFFEE',) ==> ('"BREAD',) , 0.750
        file iter = open(fname, 'IU')
                                                                                         Rule: ('BISCUIT',) ==> ('CORNFLAKES"',) , 1.000
        for line in file iter:
                                                                                         Rule: ('"MAGGI',) ==> ('TEA',) , 1,000
                line = line.strip().rstrip(',')
                                                                       # Remove trailing
                                                                                         555
                record = frozenset(line.split(','))
                vield record
if __name__ == "__main__":
    inFile=dataFromFile("GroceryStoreDataSet.day")
    minSupport = 0.12
    minConfidence = 0.5
    items, rules = runApriori(inFile, minSupport, minConfidence)
                                                                                                                                                                  Ln: 31 Col: 4
    printResults(items, rules)
```

Support:0.12

Confidence:0.7

```
apriori.py - C:\Nimit\apriori.py (2.7.14)
File Edit Format Run Options Window Help
            (outmoscompasinos)([most]soupers)outmos
                                                                                              Python 2.7.14 Shell
                                                                                                                                                                   for key, value in largeSet.items():
       toRetItems.extend([(tuple(item), getSupport(item))
                                                                                             File Edit Shell Debug Options Window Help
                        for item in value])
                                                                                             Python 2.7.14 (v2.7.14:84471935ed, Sep 16 2017, 20:25:58) [MSC v.1500 64 bit (AM 4
                                                                                             D64)] on win32
   toRetRules = []
                                                                                             Type "copyright", "credits" or "license()" for more information.
   for key, value in list(largeSet.items())[1:]:
                                                                                             555
      for item in value:
                                                                                             subsets = map(frozenset, [x for x in subsets(item)])
                                                                                             item: ('BISCUIT',) , 0.150
           for element in subsets:
                                                                                             item: ('"MAGGI',) , 0.150
              remain = item.difference(element)
                                                                                             item: ('BOURNVITA"', '"BREAD') , 0.150
               if len(remain) > 0:
                                                                                             item: ('"COFFEE', 'CORNFLAKES"') , 0.150
                  confidence = getSupport(item)/getSupport(element)
                                                                                             item: ('BISCUIT', 'CORNFLAKES"') , 0.150
                  if confidence >= minConfidence:
                                                                                             item: ('"MAGGI', 'TEA') , 0.150
                      toRetRules.append(((tuple(element), tuple(remain)),
                                                                                             item: ('COFFEE', '"BREAD') , 0.150
                                        confidence))
                                                                                             item: ('"COFFEE',) , 0.200
   return toRetItems, toRetRules
                                                                                             item: ('BISCUIT"',) , 0.200
                                                                                             item: ('SUGER',) , 0.200
                                                                                             item: ('BREAD',) , 0.200
                                                                                             item: ('COFFEE',) , 0.200
def printResults(items, rules):
                                                                                             item: ('BOURNVITA"',) , 0.200
   for item, support in sorted(items, key=operator.itemgetter(1)):
                                                                                             item: ('TEA',) , 0.250
      print("item: %s , %.3f" % (str(item), support))
                                                                                             item: ('CORNFLAKES"',) , 0.300
   print ("\n----- RULES:")
                                                                                             item: ('"BREAD',) , 0.450
   for rule, confidence in sorted(rules, key=operator.itemgetter(1)):
       pre, post = rule
                                                                                              ----- RULES:
       print("Rule: %s ==> %s , %.3f" % (str(pre), str(post), confidence))
                                                                                             Rule: ('BOURNVITA"',) ==> ('"BREAD',) , 0.750
                                                                                             Rule: ('"COFFEE',) ==> ('CORNFLAKES"',) , 0.750
                                                                                             Rule: ('COFFEE',) ==> ('"BREAD',) , 0.750
def dataFromFile(fname):
                                                                                             Rule: ('BISCUIT',) ==> ('CORNFLAKES"',) , 1.000
       """Function which reads from the file and yields a generator"""
                                                                                             Rule: ('"MAGGI',) ==> ('TEA',) , 1.000
       file iter = open(fname, 'rU')
       for line in file iter:
              line = line.strip().rstrip(',')
                                                                    # Remove trailing comma
              record = frozenset(line.split(','))
              yield record
if name == " main ":
   inFile=dataFromFile("GroceryStoreDataSet.csv")
   minSupport = 0.12
   minConfidence = 0.7
   items, rules = runApriori(inFile, minSupport, minConfidence)
                                                                                                                                                                   Ln: 28 Col: 4
   nrintDagulte/itame rules)
```