CSE 469: Assignment 2

Principle Association Analysis

Peter M. VanNostrand

10/12/2019

# Assignment 2: Association Analysis

## Frequent Itemsets – Gene Dataset

### L1 Frequent Itemsets

{gene\_1}: sup = 0.83

{gene\_12}: sup = 0.54

{gene\_14}: sup = 0.52

{gene\_17}: sup = 0.55

{gene\_21}: sup = 0.62

{gene\_22}: sup = 0.55

{gene\_23}: sup = 0.54

{gene\_25}: sup = 0.57

{gene\_26}: sup = 0.52

{gene\_27}: sup = 0.51

{gene\_3}: sup = 0.71

{gene\_31}: sup = 0.51

{gene\_36}: sup = 0.61

{gene\_37}: sup = 0.56

{gene\_39}: sup = 0.51

{gene\_4}: sup = 0.5

{gene\_43}: sup = 0.5

{gene\_45}: sup = 0.58

{gene\_47}: sup = 0.66

{gene\_48}: sup = 0.57

{gene\_5}: sup = 0.73

{gene\_50}: sup = 0.5

{gene\_53}: sup = 0.5

{gene\_54}: sup = 0.67

{gene\_55}: sup = 0.55

{gene\_56}: sup = 0.51

{gene\_59}: sup = 0.76

{gene\_6}: sup = 0.66

{gene\_60}: sup = 0.54

{gene\_63}: sup = 0.5

{gene\_64}: sup = 0.5

{gene\_66}: sup = 0.59

{gene\_67}: sup = 0.62

{gene\_71}: sup = 0.58

{gene\_72}: sup = 0.74

{gene\_75}: sup = 0.57

{gene\_77}: sup = 0.58

{gene\_78}: sup = 0.59

{gene\_8}: sup = 0.66

{gene\_81}: sup = 0.58

{gene\_83}: sup = 0.5

{gene\_84}: sup = 0.54

{gene\_87}: sup = 0.67

{gene\_89}: sup = 0.59

{gene\_9}: sup = 0.5

{gene\_90}: sup = 0.52

{gene\_91}: sup = 0.65

{gene\_93}: sup = 0.53

{gene\_94}: sup = 0.62

{gene\_98}: sup = 0.51

{gene\_99}: sup = 0.56

### L2 Frequent Itemsets

{gene\_5, gene\_72}: sup = 0.51

{gene\_59, gene\_5}: sup = 0.51

{gene\_1, gene\_59}: sup = 0.62

{gene\_47, gene\_3}: sup = 0.5

{gene\_3, gene\_72}: sup = 0.53

{gene\_47, gene\_5}: sup = 0.53

{gene\_1, gene\_67}: sup = 0.55

{gene\_1, gene\_3}: sup = 0.63

{gene\_1, gene\_8}: sup = 0.53

{gene\_1, gene\_84}: sup = 0.5

{gene\_81, gene\_1}: sup = 0.51

{gene\_87, gene\_59}: sup = 0.51

{gene\_1, gene\_6}: sup = 0.59

{gene\_1, gene\_89}: sup = 0.52

{gene\_1, gene\_72}: sup = 0.61

{gene\_87, gene\_1}: sup = 0.56

{gene\_1, gene\_21}: sup = 0.53

{gene\_87, gene\_5}: sup = 0.51

{gene\_91, gene\_5}: sup = 0.5

{gene\_1, gene\_94}: sup = 0.54

{gene\_59, gene\_3}: sup = 0.56

{gene\_1, gene\_54}: sup = 0.58

{gene\_91, gene\_1}: sup = 0.55

{gene\_1, gene\_47}: sup = 0.59

{gene\_5, gene\_3}: sup = 0.59

{gene\_59, gene\_6}: sup = 0.51

{gene\_1, gene\_5}: sup = 0.65

{gene\_6, gene\_5}: sup = 0.52

{gene\_59, gene\_72}: sup = 0.62

### L3 Frequent Itemsets

{gene\_1, gene\_72, gene\_59}: sup = 0.5

{gene\_1, gene\_5, gene\_3}: sup = 0.52

## Length 3 Candidate Itemsets – Gene Dataset

{'gene\_72', 'gene\_3', 'gene\_5'}

{'gene\_47', 'gene\_3', 'gene\_5'}

{'gene\_59', 'gene\_5', 'gene\_1'}

{'gene\_1', 'gene\_91', 'gene\_5'}

{'gene\_59', 'gene\_5', 'gene\_6'}

{'gene\_1', 'gene\_3', 'gene\_5'}

{'gene\_59', 'gene\_5', 'gene\_72'}

{'gene\_87', 'gene\_59', 'gene\_1'}

{'gene\_87', 'gene\_1', 'gene\_5'}

{'gene\_72', 'gene\_3', 'gene\_1'}

{'gene\_47', 'gene\_3', 'gene\_1'}

{'gene\_72', 'gene\_1', 'gene\_5'}

{'gene\_59', 'gene\_1', 'gene\_6'}

{'gene\_72', 'gene\_59', 'gene\_1'}

{'gene\_59', 'gene\_3', 'gene\_5'}

{'gene\_1', 'gene\_47', 'gene\_5'}

{'gene\_59', 'gene\_3', 'gene\_1'}

{'gene\_1', 'gene\_5', 'gene\_6'}

{'gene\_59', 'gene\_3', 'gene\_72'}

{'gene\_59', 'gene\_87', 'gene\_5'}

## Apriori Algorithm Code

Below are the portions of code which I wrote to fill in the template functions provided. These have been nicely formatted to fit in this document. A full copy paste of the code is available in the appendix, read at your own risk.

### Apriori Gen

### Get Freq



## Appendix

Full text of Apriori\_template.py

from \_\_future\_\_ import print\_function

import sys

def apriori(dataset, min\_support=0.5, verbose=False):

"""Implements the Apriori algorithm.

The Apriori algorithm will iteratively generate new candidate

k-itemsets using the frequent (k-1)-itemsets found in the previous

iteration.

Parameters

----------

dataset : list

The dataset (a list of transactions) from which to generate

candidate itemsets.

min\_support : float

The minimum support threshold. Defaults to 0.5.

Returns

-------

F : list

The list of frequent itemsets.

support\_data : dict

The support data for all candidate itemsets.

References

----------

.. [1] R. Agrawal, R. Srikant, "Fast Algorithms for Mining Association

Rules", 1994.

"""

C1 = create\_candidates(dataset)

D = list(map(set, dataset))

F1, support\_data = get\_freq(D, C1, min\_support, verbose=False) # get frequent 1-itemsets

F = [F1] # list of frequent itemsets; initialized to frequent 1-itemsets

k = 2 # the itemset cardinality

while (len(F[k - 2]) > 0):

Ck = apriori\_gen(F[k-2], k) # generate candidate itemsets

Fk, supK = get\_freq(D, Ck, min\_support) # get frequent itemsets

support\_data.update(supK)# update the support counts to reflect pruning

F.append(Fk) # add the frequent k-itemsets to the list of frequent itemsets

k += 1

if verbose:

# Print a list of all the frequent itemsets.

for kset in F:

for item in kset:

print("" + "{" + "".join(str(i) + ", " for i in iter(item)).rstrip(', ') + "}" + ": sup = " + str(round(support\_data[item], 3)))

return F, support\_data

def create\_candidates(dataset, verbose=False):

"""Creates a list of candidate 1-itemsets from a list of transactions.

Parameters

----------

dataset : list

The dataset (a list of transactions) from which to generate candidate

itemsets.

Returns

-------

The list of candidate itemsets (c1) passed as a frozenset (a set that is

immutable and hashable).

"""

c1 = [] # list of all items in the database of transactions

for transaction in dataset:

for item in transaction:

if not [item] in c1:

c1.append([item])

c1.sort()

if verbose:

# Print a list of all the candidate items.

print("" + "{" + "".join(str(i[0]) + ", " for i in iter(c1)).rstrip(', ') + "}")

# Map c1 to a frozenset because it will be the key of a dictionary.

return list(map(frozenset, c1))

def get\_freq(dataset, candidates, min\_support, verbose=False):

"""

This function separates the candidates itemsets into frequent itemset and infrequent itemsets based on the min\_support,

and returns all candidate itemsets that meet a minimum support threshold.

Parameters

----------

dataset : list

The dataset (a list of transactions) from which to generate candidate

itemsets.

candidates : frozenset

The list of candidate itemsets.

min\_support : float

The minimum support threshold.

Returns

-------

freq\_list : list

The list of frequent itemsets.

support\_data : dict

The support data for all candidate itemsets.

"""

freq\_list = []

support\_data = dict()

for candidateSet in candidates:

supportCount = 0

for transaction in dataset:

if candidateSet.issubset(transaction):

supportCount += 1

support = supportCount / len(dataset)

if support >= min\_support:

freq\_list.append(candidateSet)

support\_data[candidateSet] = support

return freq\_list, support\_data

def apriori\_gen(freq\_sets, k):

"""Generates candidate itemsets (via the F\_k-1 x F\_k-1 method).

This part generates new candidate k-itemsets based on the frequent

(k-1)-itemsets found in the previous iteration.

The apriori\_gen function performs two operations:

(1) Generate length k candidate itemsets from length k-1 frequent itemsets

(2) Prune candidate itemsets containing subsets of length k-1 that are infrequent

Parameters

----------

freq\_sets : list

The list of frequent (k-1)-itemsets.

k : integer

The cardinality of the current itemsets being evaluated.

Returns

-------

candidate\_list : list

The list of candidate itemsets.

"""

n = len(freq\_sets)

if n<2: # Minimum 2 frequent itemsets needed to generate candidates

return []

# generate all possible candidate itemsets

candidate\_set = set()

for i in range(0, n-1): # iterate through each element

for j in range(i+1, n): # and try to combine it with every element after it

commonElems = freq\_sets[i].intersection(freq\_sets[j])

if len(commonElems) >= k-2: # if k-2 of the items in the sets match

newCandidate = freq\_sets[i].union(freq\_sets[j]) # combine the sets to make a length k itemset

candidate\_set.add(newCandidate) # add that itemset to the list of candidates

# find candidate itemsets which have k-1 length subsets that are infrequent

invalidCandidates = set()

for candidate in candidate\_set:

for elem in candidate:

k1subset = candidate.difference({elem}) # generate every possible k-1 subset

subsetIsFrequent = False

for freqItemset in freq\_sets: # check that every k-1 subset is frequent

if k1subset == freqItemset:

subsetIsFrequent = True

break

if not subsetIsFrequent: # if one or more of the k-1 subsets was infrequent

invalidCandidates.add(candidate) # prune the candidate itemset

# prune the invalid candidates

for candidate in invalidCandidates:

candidate\_set.remove(candidate)

return list(candidate\_set)

def loadDataSet(fileName, delim=','):

fr = open(fileName)

stringArr = [line.strip().split(delim) for line in fr.readlines()]

return stringArr

def run\_apriori(data\_path, min\_support, verbose=False):

dataset = loadDataSet(data\_path)

F, support = apriori(dataset, min\_support=min\_support, verbose=verbose)

return F, support

def bool\_transfer(input):

''' Transfer the input to boolean type'''

input = str(input)

if input.lower() in ['t', '1', 'true' ]:

return True

elif input.lower() in ['f', '0', 'false']:

return False

else:

raise ValueError('Input must be one of {T, t, 1, True, true, F, F, 0, False, false}')

if \_\_name\_\_ == '\_\_main\_\_':

if len(sys.argv)==3:

F, support = run\_apriori(sys.argv[1], float(sys.argv[2]))

elif len(sys.argv)==4:

F, support = run\_apriori(sys.argv[1], float(sys.argv[2]), bool\_transfer(sys.argv[3]))

else:

raise ValueError('Usage: python apriori\_templete.py <data\_path> <min\_support> <is\_verbose>')

print(F)

print(support)

'''

Example:

python apriori\_templete.py market\_data\_transaction.txt 0.5

python apriori\_templete.py market\_data\_transaction.txt 0.5 True

'''