**CS 634**

**DATA MINING**

**MID-TERM PROJECT**

**IMPLEMENTATION OF APRIORI ALGORITHM**

**AND BRUTE-FORCE APPROACH**

**UCID – kg496**

**Name: Karthik Ganeshula**

**Email: kg496@njit.edu**

**Part – 1: Apriori**

This part of the project contains the implementation of the Apriori algorithm on five transactional data-sets and shows the performance of the algorithm in terms of time along with the Association Rules.

**Source code:**

**import** **time**

**from** **itertools** **import** combinations

min\_s = int(input("Enter the minimum support: "))

min\_c = int(input("Enter the minimum confidence: "))

start = time.time()

file\_object = open("DataSet-1.txt", "r")

l\_of\_data = file\_object.readlines()

allItemList = []

n\_list = 0

sup\_all\_items\_s={}

d = {}

items\_list = []

**print**("-------------------")

**print**("The following are the input transactions:")

**print**("-------------------")

**print**()

**for** q **in** l\_of\_data:

q = q.replace("**\n**", "")

**print**(q)

allItemList.append("".join(q.split(" ")[1].split(",")))

s = set()

**for** i **in** "".join(q.split(" ")[1:]).split(","):

**if** (i,) **in** d:

d[(i,)] += 1

**else**:

d[(i,)] = 1

s.add(i)

items\_list.append(s)

n\_list+=1

frequentSet = {}

eliminatedSet = []

**print**()

**print**("-------------------------")

**print**("Item Sets of size", 1)

**print**("-------------------------")

**print**()

**for** i **in** d:

**print**(i,round(d[i]\*100/n\_list))

**if** (d[i]/20)\*100 >= min\_s:

frequentSet[i] = d[i]

**else**:

eliminatedSet.append(set(i))

sup\_all\_items\_s.update(d)

l = []

**for** i **in** frequentSet.keys():

l.append(i[0])

**def** find\_frequent\_Sets(l, elims, items\_list, n):

combos = combinations(l, n)

support\_c\_dict = {}

**for** i **in** combos:

set\_of\_i = set(i)

i = tuple(sorted(i))

**for** j **in** items\_list:

**if** set\_of\_i.issubset(j):

**if** len(elims) > 0:

c = 0

**for** k **in** elims:

**if** k.issubset(set\_of\_i):

c = 1

**break**

**if** **not** c:

**if** i **in** support\_c\_dict:

support\_c\_dict[i] += 1

**else**:

support\_c\_dict[i] = 1

**else**:

**if** i **in** support\_c\_dict:

support\_c\_dict[i] += 1

**else**:

support\_c\_dict[i] = 1

frequentSet\_final = {}

elim\_set\_final = []

**if** len(support\_c\_dict) > 0:

**print**("------------------------------")

**print**("Itemsets for size: ", n)

**print**("------------------------------")

**print**()

**for** i **in** support\_c\_dict:

**print**(i,round(support\_c\_dict[i]\*100/n\_list,2))

**if** (support\_c\_dict[i]/n\_list)\*100 >= min\_s:

frequentSet\_final[i] = support\_c\_dict[i]

**else**:

elim\_set\_final.append(set(list(i)))

**print**()

**if** len(frequentSet\_final) > 0:

sup\_all\_items\_s.update(support\_c\_dict)

Assoc\_rules(frequentSet\_final)

**return** frequentSet\_final, elim\_set\_final

**return** None,None

**def** Print\_items(frequentSet,n):

**print**("----------------------------------------")

**print**("Itemsets after iteration of size: ", n)

**print**("Itemset", "Support value")

**print**("----------------------------------------")

**print**()

**for** i **in** frequentSet:

**print**(i,round(frequentSet[i]\*100/n\_list,2),)

**print**()

**def** Assoc\_rules(frequentSet):

**for** item **in** frequentSet.keys():

**print**("Association Rule for the itemset - ",item)

**print**("Support","Confidence")

SizeOfSet=len(item)

itemset=set(item)

**while** SizeOfSet-1>0:

combos = combinations(item, SizeOfSet-1)

**for** i **in** combos:

left\_items=i

right\_items=tuple(itemset-set(i))

item\_confidence=round(sup\_all\_items\_s[item]\*100/sup\_all\_items\_s[left\_items],2)

**if** item\_confidence>=min\_c:

**print**(left\_items,"-->",right\_items,item\_confidence,"Qualified")

**else**:

**print**(left\_items,"-->",right\_items,item\_confidence,"Unqualified")

SizeOfSet -=1

**print**()

**print**()

item\_set\_size = 1

**while** len(l) > item\_set\_size:

frequentSet1, eliminatedSet1 = find\_frequent\_Sets(l, eliminatedSet, items\_list, item\_set\_size + 1)

**if** **not** frequentSet1:

**break**

listOfItems = []

**for** items **in** list(frequentSet1.keys()):

**for** i **in** items:

listOfItems.append(i)

l = list(set(listOfItems))

eliminatedSet = eliminatedSet1

frequentSet=frequentSet1

item\_set\_size += 1

**print**('The Apriori algorithm now executed takes: **%s** ' % (time.time() - start), "seconds")

The following are the Datasets used of the project:

DataSet -1 :

1 book, glass, mouse, hat, mat

2 pant, shirt, coat, ink, ball

3 bottle, wallet, coat, hat, paper

4 mouse, pillow, rock, shirt, lock

5 wallet, coat, book, pen, note

6 candy, usb, wire, cap, belt

7 wallet, glass, hat, coat, ink

8 lock, pant, bat, ball, phone

9 wallet, pillow, usb, pant, shirt

10 mouse, usb, glass, cap, helmet

11 gum, helmet, ball, bat, belt

12 book, pen, pencil, bat, ball

13 cap, rock, shirt, wire, bottle

14 book, mat, candy, belt, usb

15 pillow, cap, pant, mat, candy

16 glass, pen, helmet, rock, pant

17 wicket, note, gum, glass, rock

18 pant, coat, bat, ball, belt

19 usb, ink, paper, helmet, note

20 gum, cap, note, usb, wire

DataSet – 2:

1 pen, pencil, note, ball, wicket

2 coat, hat, coat, lock, rock

3 bottle, glass, pillow, hat, wallet

4 glass, shirt, pillow, wore, usb

5 cap, book, gum, hat, mouse,

6 wallet, wicket, coat, phone

7 wire, usb, hat, book, pen

8 coat, pencil, note, gum, pant

9 hat, lock, clock, usb, pillow

10 wicket, bat, ball, phone, mat

11 phone, hat, shirt, rock, clock

12 cap, pencil, ink, candy, lock

13 pant, bat, ball, note, gum

14 belt, rock, wicket, wallet, usb

15 shirt, gum, clock, belt, rock

16 cap, bat, ball, gum, note

17 rock, book, pillow, mat, clock

18 glass, pant, cap, coat, helmet

19 pen, pencil, book, ink, glass

20 pillow, helmet, cap, bottle, gum

DataSet – 3:

1 mat, mouse, helmet, gum, candy

2 shirt, coat, mat, phone, lock

3 usb, pillow, helmet, bottle, hat

4 note, gum, candy, hat, mouse

5 wallet, shirt, belt, rock, clock

6 gum, helmet, bottle, pen, pencil

7 belt, gum, pencil, ink, book

8 cap, mat, mouse, phone, pant

9 pen, ball, note, gum, pant

10 bottle, bat, clock, rock, ink

11 usb, wire, pillow, pant, coat

12 pen, ball, wicket, helmet, hat

13 pant, paper, phone, lock, gum

14 glass, bat, ball, belt, clock

15 pant, shirt, pillow, candy, helmet

16 cap, pant, book, usb, wire

17 pillow, bottle, rock, wallet, glass

18 paper, helmet, gum, pant, rock

19 shirt, coat, rock, pillow, helmet

20 gum, note, glass, bat, ball

DataSet – 4:

1 mat, ball, belt, rock, clock

2 hat, mouse, helmet, gum, wicket

3 belt, wallet, pillow, book, pen

4 phone, lock, rock, cap, shirt

5 book, note, gum, pillow, clock

6 hat, mat, gum, rock, bottle

7 wallet, rock, cap, wire, usb

8 belt, pant, shirt, coat, hat

9 mat, pillow, helmet, rock, clock

10 paper, rock, ink, wicket, pencil

11 wallet, glass, clock, usb, wire

12 cap, ink, paper, helmet, candy

13 phone, gum, book, pen, pencil

14 coat, mat, helmet, ink, wicket

15 bat, ball, cap, belt, pillow

16 lock, shirt, coat, book, gum

17 pen, pencil, ink, note, pant

18 hat, mat, pant, clock, pillow

19 candy, belt, pant, gum, pillow

20 helmet, mat, hat, wallet, glass

DataSet – 5:

1 wallet, gum, ink, note, helmet

2 phone, mat, hat, rock, wallet

3 candy, pencil, book, pen, clock

4 pant, bottle, wicket, ball, rock

5 usb, wire, pillow, bottle, belt

6 helmet, note, candy, clock, bottle

7 wicket, pant, phone, wire, pillow

8 note, gum, belt, shirt, rock

9 mouse, note, gum, cap, wire

10 pen, ink, phone, clock, book

11 hat, mat, helmet, usb, wire

12 pant, pillow, ink, ball, hat

13 coat, hat, rock, gum, note

14 cap, pant, coat, pencil, coat

15 usb, wire, pillow, pen, gum

16 hat, rock, clock, ink, wicket

17 bottle, wallet, ink, usb, wire

18 mat, belt, rock, book, shirt

19 candy, belt, rock, clock, pant

20 note, book, pen, pencil, glass

The following are the screenshots of the source code after the execution:

DataSet – 1:

Graphical user interface, text, application

Description automatically generatedText

Description automatically generatedText

Description automatically generatedGraphical user interface, text

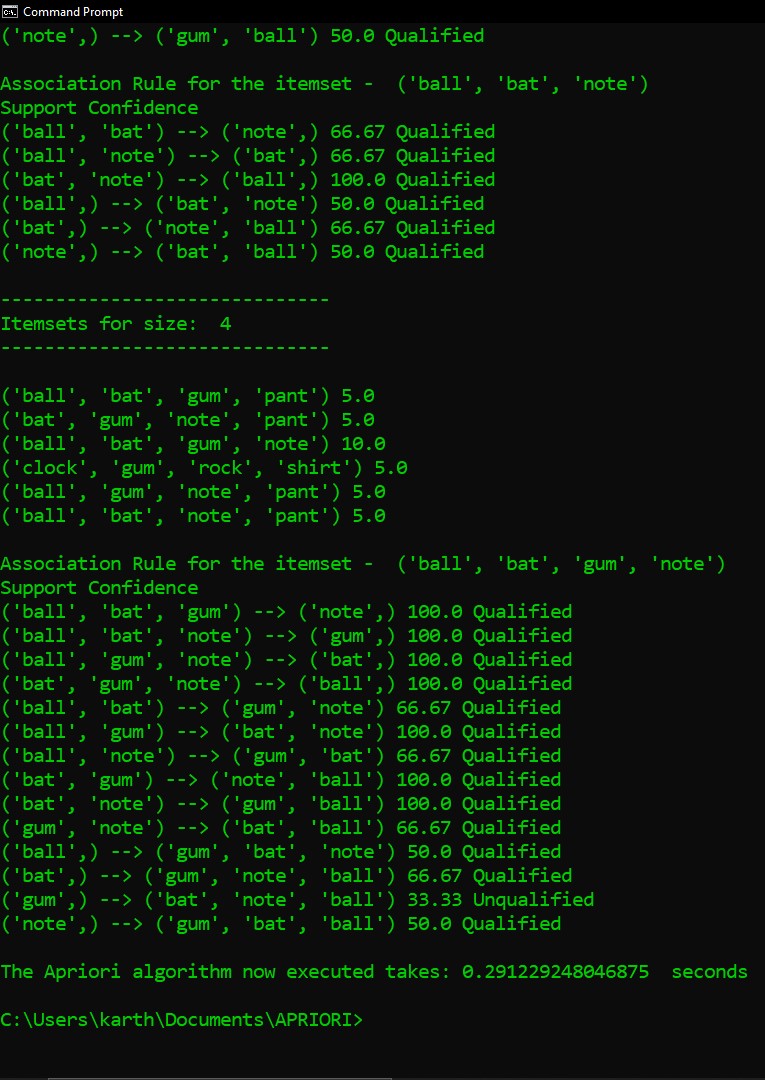
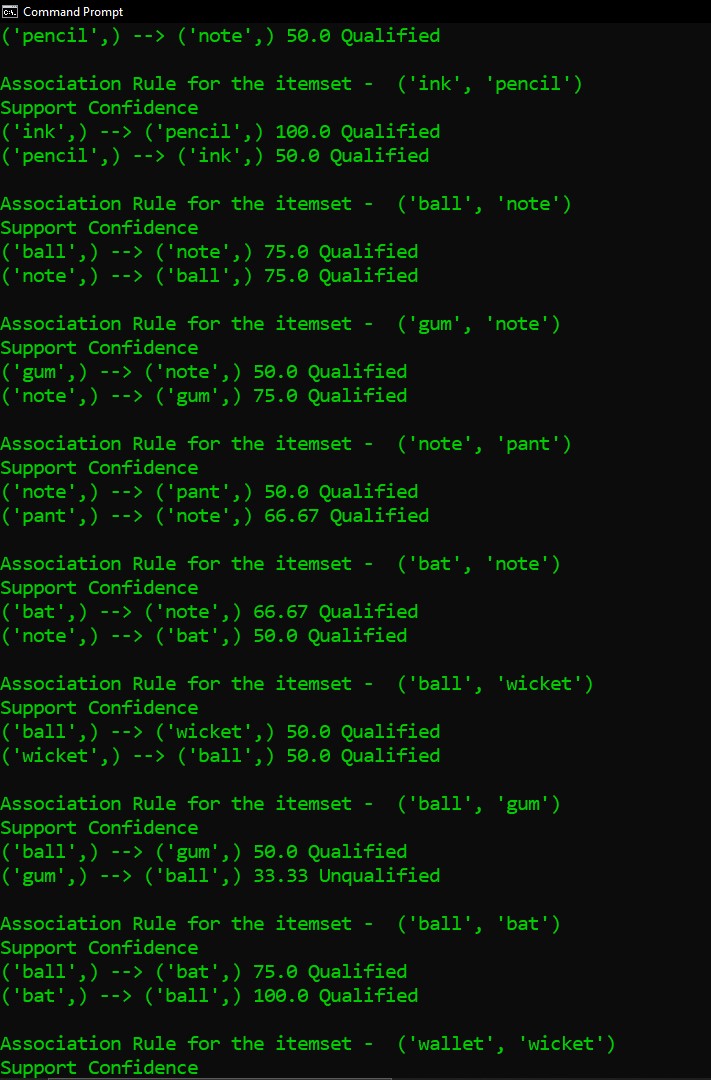
Description automatically generated

Time taken for execution: 0.31 sec

DataSet-2:

Graphical user interface, text

Description automatically generated



Time taken for execution: 0.29 sec

DataSet-3:

Text

Description automatically generated

Text

Description automatically generatedGraphical user interface, text

Description automatically generated

Time taken for execution: 0.37 sec

DataSet-4:

Graphical user interface, text

Description automatically generated

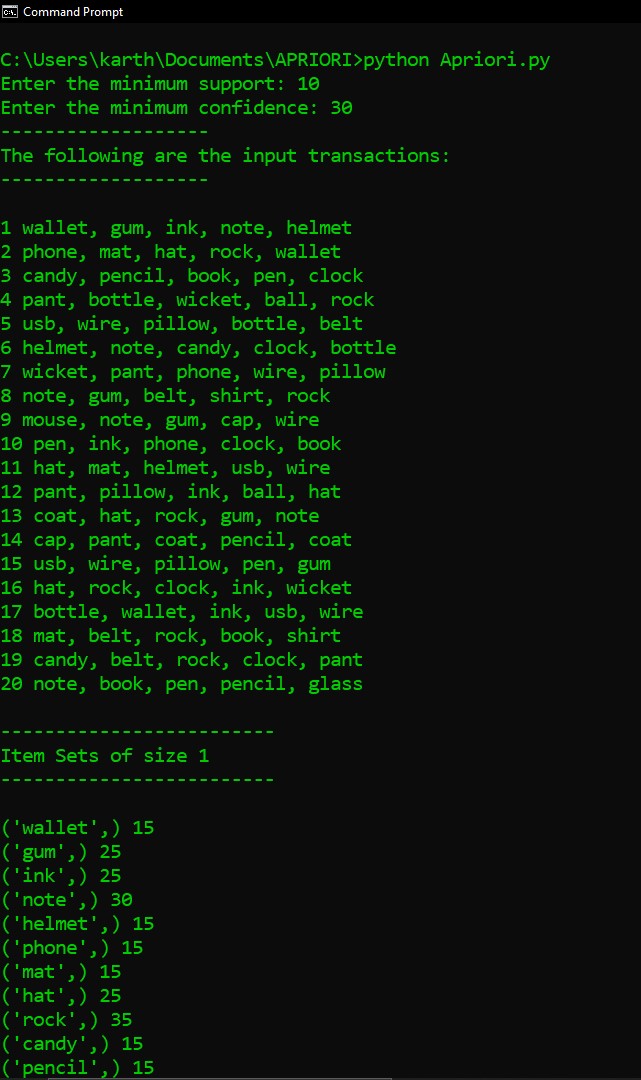
Graphical user interface, text

Description automatically generatedGraphical user interface, text

Description automatically generated

Time taken for execution: 0.28 sec

DataSet-5:



Graphical user interface, text

Description automatically generatedGraphical user interface, text

Description automatically generated

Time taken for execution: 0.32 sec

**Part-B: Brute-Force approach**

**Source code:**

**from** **itertools** **import** combinations

**import** **time**

min\_s = int(input("Enter the minimum support: "))

min\_c = int(input("Enter the minimum Confidence: "))

start = time.time()

file\_object = open("DataSet-5.txt", "r")

l\_of\_data = file\_object.readlines()

allItemList = []

items\_list = []

d = {}

n\_list = 0

support\_of\_all\_item\_set={}

**for** q **in** l\_of\_data:

q = q.replace("**\n**", "")

allItemList.append("".join(q.split(" ")[1].split(",")))

s = set()

**for** i **in** "".join(q.split(" ")[1:]).split(","):

**if** (i,) **in** d:

d[(i,)] += 1

**else**:

d[(i,)] = 1

s.add(i)

items\_list.append(s)

n\_list+=1

k = []

**for** i **in** items\_list:

i = list(i)

k.extend(i)

k = list(set(k))

**def** find\_support(l):

dict\_sup = {}

length = len(items\_list)

**for** i **in** l:

c = 0

**for** j **in** items\_list:

**if** type(i) != tuple:

**if** set([i]).issubset(set(j)):

c += 1

**else**:

**if** set(i).issubset(set(j)):

c += 1

dict\_sup[i] = c\*100/length

**return** dict\_sup

individual\_support = find\_support(k)

**def** Printer(frequentSet):

**print**("ItemSets for input: ")

**for** i **in** frequentSet.keys():

**print**(i,)

**def** Assoc\_rules(frequentSet):

**for** item **in** frequentSet.keys():

SizeOfSet=len(item)

itemset=set(item)

**while** SizeOfSet-1>0:

combos = combinations(item, SizeOfSet-1)

**for** i **in** combos:

lefts=i[0]

rights=tuple(itemset-set(i))

item\_confidence=round(current\_support[item]\*100/individual\_support[lefts],2)

**if** item\_confidence >= min\_c **and** current\_support[item] >= min\_s:

**print**("Association rules of the item: ", item, "**\n**Support: ", current\_support[item])

**print**(lefts," -->", rights, "Confidence: ", item\_confidence)

**print**()

SizeOfSet -=1

current\_size = 1

f = []

**while** current\_size <= len(items\_list[0]):

**print**("--------------------------------------------------------------------------------------------------------------")

**print**("ItemSets with ", current\_size, " items")

**print**("--------------------------------------------------------------------------------------------------------------")

**if** current\_size == 1:

**for** i **in** individual\_support.keys():

**print**(i, "support: ", individual\_support[i])

**print**()

**elif** current\_size > 1:

current\_support = find\_support(list(combinations(k,current\_size)))

**for** i **in** current\_support.keys():

lefts = i[0]

item\_confidence=round(current\_support[i]\*100/individual\_support[lefts],2)

**if** current\_support[i] >= min\_s:

f.append(i)

**print**(i, "with support: ", current\_support[i], "and Confidence: ", item\_confidence)

**print**()

Assoc\_rules(current\_support)

**if** current\_size > 1:

**if** len(f) > 1:

f = []

**else**:

**break**

current\_size += 1

**print**()

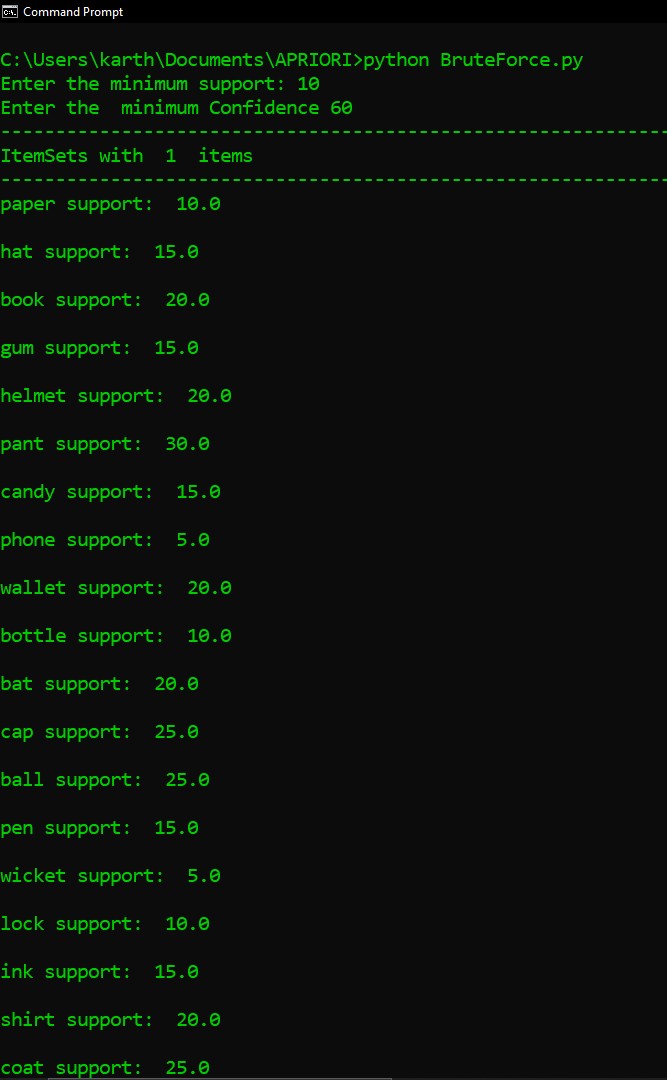
**print**()

**print**("Time taken for the Brute Force is ", time.time() - start)

The following are the screenshots of datasets after execution:

(Note: Unlike the Apriori algorithm, the item sets that do not qualify for the minimum support and confidences are not printed)

DataSet-1:

Graphical user interface, text

Description automatically generatedText

Description automatically generated

Time taken for execution: 0.81 sec

DataSet – 2:

Graphical user interface, text

Description automatically generated

Graphical user interface, text

Description automatically generatedGraphical user interface, text

Description automatically generated

Time taken for execution: 1.02 sec

DataSet-3:

Graphical user interface, text

Description automatically generated

Graphical user interface, text

Description automatically generatedGraphical user interface, text

Description automatically generated

Time taken for execution: 0.88 sec

DataSet-4:

Graphical user interface, text

Description automatically generated

Text

Description automatically generatedGraphical user interface, text

Description automatically generated

Time taken for execution: 1.01 sec

DataSet – 5:

Text

Description automatically generated

Text

Description automatically generatedText

Description automatically generated

Time taken for the execution: 0.70 sec

The following are the time differences of the execution times of the datasets using both the algorithms:

**DataSet-1:**

Apriori:0.31 sec

B-F:0.81 sec

Difference: Apriori is 0.50 sec faster

**DataSet-2:**

Apriori:0.29 sec

B-F:1.02 sec

Difference: Apriori is 0.71 sec faster

**DataSet-3:**

Apriori: 0.37 sec

B-F: 0.88 sec

Difference: Apriori is 0.55 sec faster

**DataSet-4:**

Apriori: 0.28 sec

B-F: 1.01 sec

Difference: Apriori is 0.73 sec faster

**Dataset-5:**

Apriori: 0.32 sec

B-F: 0.70 sec

Difference: Apriori is 0.38 sec faster