Midterm Project

CS634

Fall 2021

Prepared for:

Dr. Jason Wang, Professor,
Department of Computer Science,
New Jersey Institute of Technology

Prepared by:

First Name:	Md Rakibul
Last Name:	Hasan
NJIT UCID:	31546095
Email:	mh629@njit.edu
Date:	9/30/2021

Table of Contents

Chapter 1 Introduction

Association rules help to find frequent patterns, associations, or correlations among sets of items or objects in transactional databases. It is often used for boosting sales. To find out frequent item set, we may use Apriori algorithm and association rule over a database. There are 2 principles of Apriori algorithm.

- Any subset of a frequent itemset must be frequent.
- Any superset of a non-frequent itemset must be non-frequent.

In this midterm project, Apriori algorithm will be implemented from scratch and will be applied on 5 different databases as well as the performance will be compared with Brute force method.

Chapter 2 Project Programming Environment

During this project completion, below mentioned environment were used.

- Programming Language: Python 3.9.7
- IDE: Jupyter Notebook
- OS: Windows 10
- Spreadsheet Application: Microsoft Excel CSV file

Chapter 3 Data Description

In this project, I have used 30 products purchased frequently from Walmart by men.

Category	Items
Apparel	Jacket, Pants, Hoodie, Sock, Jeans
Kitchen	Pan, Oven, Kettle, Blender, Knife
Electronics	Headphone, Mobile, Laptop, Printer, Scanner
Personal Care	Razor, Shampoo, Deodorant, Lotion, Soap
School	Pen, Pencil, Eraser, Calculator, Marker
Accessories	Tissue, Umbrella, Wallet, Watch, Sunglass

Using these products, 5 databases were created.

- Database 1:

Transaction ID	**Items**
1000	Tissue,Watch
1001	Watch,Umbrella
1002	Eraser,Pencil
1003	Oven,Blender,Knife
1004	Blender,Kettle,Pan,Knife
1005	Pan,Oven
1006	Pencil,Marker,Pen,Calculator
1007	Sunglass,Watch,Tissue
1008	Headphone,Laptop,Scanner
1009	Watch,Tissue,Umbrella,Sunglass
1010	Scanner,Printer,Headphone
1011	Headphone,Laptop
1012	Watch,Tissue,Umbrella
1013	Watch,Tissue,Umbrella,Wallet
1014	Blender,Knife
1015	Scanner,Laptop
1016	Calculator,Pencil,Eraser
1017	Marker,Calculator
1018	Blender,Knife,Pan,Oven
1019	Pencil,Calculator

- Database 2:

Transaction ID	Items
1000	Knife,Blender,Oven
1001	Pan,Knife,Kettle,Blender
1002	Laptop,Mobile,Headphone,Printer
1003	Mobile,Laptop
1004	Hoodie,Pants,Sock,Jackets
1005	Headphone,Mobile
1006	Laptop,Printer,Headphone
1007	Calculator,Pen,Pencil
1008	Pen,Eraser
1009	Marker,Calculator
1010	Pen,Pencil
1011	Jackets,Pants,Sock
1012	Hoodie,Jeans
1013	Sock,Pants,Hoodie
1014	Printer,Mobile,Scanner,Laptop

1015	Kettle,Pan,Oven
1016	Marker,Pencil,Pen,Calculator
1017	Blender,Knife,Kettle
1018	Eraser,Pencil
1019	Headphone,Mobile,Scanner

- Database 3:

Transaction ID	Items
1000	Deodorant,Lotion
1001	Kettle,Oven,Knife
1002	Oven,Knife,Blender
1003	Deodorant,Lotion,Razor
1004	Pen,Pencil
1005	Razor,Shampoo,Lotion,Deodorant
1006	Calculator,Pencil,Pen
1007	Jeans,Pants,Hoodie
1008	Pencil,Pen
1009	Marker,Pen,Calculator,Eraser
1010	Shampoo,Lotion,Soap
1011	Calculator,Pencil,Pen,Eraser
1012	Oven,Kettle,Knife,Pan
1013	Blender,Oven
1014	Oven,Kettle,Blender,Knife
1015	Pants,Jackets
1016	Shampoo,Razor,Deodorant
1017	Shampoo,Deodorant,Razor,Lotion
1018	Shampoo,Razor,Lotion
1019	Calculator,Marker,Pen

- Database 4:

Transaction ID	Items
1000	Eraser,Scanner,Jeans,Sunglass,Knife,Soap
1001	Lotion,Umbrella,Soap,Hoodie,Razor,Headphone,Watch
1002	Jackets,Pan,Watch,Knife,Sunglass,Deodorant,Scanner
1003	Kettle,Hoodie,Eraser,Pen,Lotion,Marker,Laptop,Pants,Oven
1004	Sock,Wallet,Watch,Lotion,Jackets,Calculator,Shampoo,Tissue,Headphone
1005	Lotion,Knife,Oven,Deodorant,Marker,Pen,Shampoo,Mobile,Tissue,Soap,Pencil
1006	Scanner,Umbrella,Printer,Soap,Watch,Mobile,Hoodie,Pants,Knife

1007	Oven,Sock,Pen,Watch,Lotion,Sunglass,Eraser,Mobile,Jackets,Marker,Scanner,Hoodie
1008	Pan,Kettle,Hoodie,Printer,Headphone,Blender,Marker,Laptop,Eraser,Pencil
1009	Pants,Umbrella,Soap,Tissue,Kettle,Marker,Sock,Laptop,Jackets,Knife,Pan,Eraser
1010	Sock,Oven,Deodorant,Hoodie,Printer,Jeans,Knife
1011	Headphone,Hoodie,Laptop,Mobile,Oven,Scanner
1012	Soap,Hoodie,Jackets,Knife,Razor,Umbrella,Eraser,Wallet,Marker
1013	Hoodie,Deodorant,Oven,Knife,Mobile,Blender,Calculator,Pan
1014	Tissue,Eraser,Scanner,Pencil,Lotion,Hoodie
1015	Pencil,Oven,Blender,Pants
1016	Jeans,Sunglass,Watch,Razor,Kettle
1017	Kettle,Deodorant,Knife,Wallet,Watch,Eraser,Laptop
1018	Headphone,Soap,Marker,Jeans,Deodorant,Hoodie,Razor,Printer,Mobile,Sunglass,Pan
1019	Mobile,Lotion,Oven,Shampoo,Deodorant,Printer,Eraser,Watch,Scanner,Sock

- Database 5:

Transaction ID	Items
1000	Hoodie,Umbrella,Printer,Kettle,Mobile,Pen,Pencil,Sunglass,Knife,Lotion
1001	Soap,Oven,Printer,Headphone,Laptop,Shampoo,Lotion
1002	Watch,Shampoo,Hoodie,Laptop,Pan,Lotion,Pants,Umbrella
1003	Umbrella,Blender,Kettle,Watch,Wallet,Printer,Pen,Knife
1004	Jackets,Marker,Blender,Knife,Soap,Sock,Hoodie,Pants,Mobile,Laptop
1005	Mobile,Sunglass,Umbrella,Kettle,Knife,Laptop,Scanner,Sock,Jackets,Pants
1006	Shampoo,Deodorant,Blender,Eraser,Oven,Headphone,Hoodie,Watch,Tissue,Razor,Lotion
1007	Soap,Jeans,Oven,Marker,Lotion,Sock,Shampoo,Jackets,Headphone,Eraser,Tissue,Pencil
1008	Lotion,Razor,Kettle,Blender,Marker,Pen,Mobile,Sock,Printer,Shampoo,Jeans
1009	Soap,Printer,Lotion,Eraser,Pen,Scanner,Mobile,Kettle,Pan,Headphone,Shampoo
1010	Kettle,Mobile,Pencil,Eraser,Umbrella,Razor,Calculator,Knife,Printer,Jeans,Pen,Headphone
1011	Calculator,Jackets,Knife,Razor,Laptop,Pan,Kettle,Sunglass,Headphone,Deodorant
1012	Wallet,Mobile,Printer,Sock,Hoodie,Jackets,Blender,Marker
1013	Lotion,Laptop,Watch,Mobile,Wallet,Jackets,Pencil,Soap,Sunglass,Knife,Marker
1014	Pencil,Laptop,Kettle,Jackets,Eraser,Jeans
1015	Sunglass,Pencil,Laptop,Hoodie,Knife,Tissue,Mobile,Headphone,Scanner
1016	Tissue,Sunglass,Pan,Marker,Razor,Watch,Kettle,Jeans,Printer
1017	Printer,Scanner,Wallet,Jeans,Jackets,Soap,Watch,Pen,Calculator,Deodorant
1018	Pan,Calculator,Shampoo,Tissue,Printer,Wallet,Oven,Laptop,Pencil
1019	Printer,Headphone,Jeans,Razor,Marker,Pants,Scanner

Chapter 4 Implementing Algorithms

4.0 Implementing Apriori Algorithm:

Here, step-by-step process of implementing Apriori algorithm will be discussed.

- In first step, database will be taken as input from CSV file.

```
In [101]: import csv
          def input_database(file_name):

              with open(file_name) as csv_file:
                  read_csv = csv.reader(csv_file, delimiter=',')
                  next(read_csv)
                  rows = []
                  for row in read_csv:
                      rows.append(row[1])

              return rows
```

- In this step, all distinct items will be returned.

```
In [102]: def all_distinct_items(rows):
              items_in_row = []
              for row in rows:
                  items_in_row.extend(row.split(","))

              return list(set(items_in_row))
```

- Here, all string values will be converted to integer for making the process simpler.

```
In [103]: def string_to_int(all_rows, distinct_items):
              rows = []
              for row in all_rows:
                  rows.append(sorted([distinct_items.index(item) for item in row.split(",")]))
              return rows
```

- Now, user input for minimum support and confidence value will be taken.

```
In [111]: def user_input():
              min_support = input("Please provide minimum Support value (Only Value): ")
              min_confidence = input("Please provide minimum Confidence value (Only Value): ")
              return min_support, min_confidence
```

- Here, a all_possible_subset function has been written to find out all possible subset of a given set input.

```
In [100]: def all_possible_subset(s):
              len_s = len(s)

              all_set = []

              for i in range(1 << len_s):

                  all_set.append([s[j] for j in range(len_s) if (i & (1 << j))])

              return all_set
```

- For each itemset, support value needs to be checked. Here, we will check each row and count number of rows that are superset of itemset.

```
In [104]: def find_support_from_items(data, items):
              support_count = 0
              for row in data:
                  if set(items).issubset(row):
                      support_count += 1
              return support_count
```

- Here, at comparing_support_value function, we pass a list of elements and compare support value for each candidate. In each level k, If candidate element's support value is less than minimum support value, we will discard those items. We will take remaining candidate element to make list of itemsets for (k+1)th level.
- According to Apriori Algorithm, any superset of a non-frequent itemset must be non-frequent. check_non-frequent_item method checks if candidate contains non-frequent itemset or not.
- After that, next level items have been created. To generate (k+1)-th level candidate items, we need to use k-th level frequent itemsets. I implemented a method which take unions of two itemsets if k-2 items are matched. And if resultant itemset does not contain any non-frequent itemset, then that itemset is added to (k+1)-th level candidate itemsets.

```
In [106]: def comparing_support_value(data, elements, min_support):
              frequent_items = []
              non_frequent_items = []
              if elements is not None:
                  for e in elements:
                      if(find_support_from_items(data, e) >= min_support):
                          frequent_items.append(e)
                      else:
                          non_frequent_items.append(e)
              return frequent_items, non_frequent_items
```

```
In [107]: def check_non_frequent_item(element, non_freq_elements):
              all_pos = all_possible_subset(element)
              for item in all_pos:
                  if item in non_freq_elements:
                      return True
              return False
```

```
In [108]: def next_level_items(pos_freq_elements, neg_freq_elements):
              next_level_items = {}
              total_pos_items = len(pos_freq_elements)
              if total_pos_items == 0:
                  return []
              len_each_item = len(pos_freq_elements[0])
              for left in range(0, total_pos_items):
                  for right in range(left+1, total_pos_items):
                      merged = tuple(sorted(set(pos_freq_elements[left]).union(set(pos_freq_elements[right]))))
                      if len(merged) == len_each_item + 1 and not check_non_frequent_item(merged, neg_freq_elements):
                          next_level_items[merged] = 1
              return [list(i) for i in next_level_items.keys()]
```

- To generate frequent itemset from Apriori algorithm, at first support value for every single itemset had been calculated for 1st level. Then, frequent itemsets from 1st level has been used to generate candidate itemsets for 2nd level. Using comparing_support_value method non-frequent items had been filtered out and a list of frequent itemsets have been prepared. This process is continued until there is a level with empty list of frequent itemsets.

```
In [109]: def apriori_algorithm(data, item_size, min_support, min_confidence):
    freq_items = []
    non_freq_items = []
    items = range(0, item_size)
    min_support_value = int(len(data) * min_support / 100.0)
    new_freq_elements = [[i] for i in items]
    pos_freq_elements, neg_freq_elements = comparing_support_value(data, new_freq_elements, min_support_value)
    freq_items.extend(pos_freq_elements)
    non_freq_items.extend(neg_freq_elements)

    for k in range(2,item_size+1):
        new_freq_elements = next_level_items(pos_freq_elements, non_freq_items)
        pos_freq_elements, neg_freq_elements = comparing_support_value(data, new_freq_elements, min_support_value)
        if not pos_freq_elements:
            break
        freq_items.extend(pos_freq_elements)
        non_freq_items.extend(neg_freq_elements)

    return freq_items
```

- To generate association rules, each frequent itemsets has been divided into left and right subpart and checked confidence value for each combination. If confidence value meets required confidence value, then that rule was added to a list.

```
In [105]: def find_association_from_items(data, freq_items, distinct_items, min_confidence):
    association_list = []
    for item in freq_items:
        all_set = all_possible_subset(item)
        for p in all_set:
            left_hand_side = set(p)
            right_hand_side = set(item) - left_hand_side
            if len(left_hand_side) and len(right_hand_side):
                support_lhs = find_support_from_items(data, left_hand_side)
                if support_lhs == 0:
                    continue
                confidence = find_support_from_items(data, item) * 100 / support_lhs
                if confidence < min_confidence:
                    continue
                lhs_items = [distinct_items[i] for i in left_hand_side]
                rhs_items = [distinct_items[i] for i in right_hand_side]
                association_list.append((lhs_items, rhs_items, confidence))
    return association_list
```

- For each database, value of minimum support and confidence have been taken from user. Then association rules were calculated based on these values. And finally, the generated association rules have been printed out.

```
In [133]: def print_association_rule(file_name, min_support, min_confidence):
    all_rows = input_database(file_name) # Load_database()
    distinct_items = all_distinct_items(all_rows)
    num_distinct_items = len(distinct_items)
    data = string_to_int(all_rows, distinct_items)

    freq_items = apriori_algorithm(data, num_distinct_items, min_support, min_confidence)
    association_list = find_association_from_items(data, freq_items, distinct_items, min_confidence)
    print("Total number of items in association list is {}".format(len(association_list)))
    for item in association_list:
        lhs_items, rhs_items, confidence = item
        print("{} -->> {} : {}".format(lhs_items, rhs_items, confidence))
```

```
In [*]: for index in range(1,6):
    print("For Database Number # {} ".format(index))
    file_name = 'database_{}.csv'.format(index)
    min_support, min_confidence = user_input()
    print("Below are the Association rules :")
    print_association_rule(file_name, float(min_support), float(min_confidence))
    print()

For Database Number # 1
Please provide minimum Support value (Only Value): 20

Please provide minimum Confidence value (Only Value): 50
```

- Association rules for Database 1:

```
In [*]: for index in range(1,6):
    print("For Database Number # {} ".format(index))
    file_name = 'database_{}.csv'.format(index)
    min_support, min_confidence = user_input()
    print("Below are the Association rules :")
    print_association_rule(file_name, float(min_support), float(min_confidence))
    print()
```

```
For Database Number # 1
Please provide minimum Support value (Only Value): 20
Please provide minimum Confidence value (Only Value): 50
Below are the Association rules :
Total number of items in association list is 6
['Blender'] -->> ['Knife'] : 100.0
['Knife'] -->> ['Blender'] : 100.0
['Umbrella'] -->> ['Watch'] : 100.0
['Watch'] -->> ['Umbrella'] : 66.66666666666667
['Tissue'] -->> ['Watch'] : 100.0
['Watch'] -->> ['Tissue'] : 83.33333333333333
```

- Association rules for Database 2:

```
For Database Number # 2
Please provide minimum Support value (Only Value): 15
Please provide minimum Confidence value (Only Value): 70
Below are the Association rules :
Total number of items in association list is 10
['Pen'] -->> ['Pencil'] : 75.0
['Pencil'] -->> ['Pen'] : 75.0
['Blender'] -->> ['Knife'] : 100.0
['Knife'] -->> ['Blender'] : 100.0
['Sock'] -->> ['Pants'] : 100.0
['Pants'] -->> ['Sock'] : 100.0
['Printer'] -->> ['Laptop'] : 100.0
['Laptop'] -->> ['Printer'] : 75.0
['Laptop'] -->> ['Mobile'] : 75.0
['Headphone'] -->> ['Mobile'] : 75.0
```

- Association rules for Database 3:

```
For Database Number # 3
Please provide minimum Support value (Only Value): 18
Please provide minimum Confidence value (Only Value): 75
Below are the Association rules :
Total number of items in association list is 29
['Calculator'] -->> ['Pen'] : 100.0
['Pencil'] -->> ['Pen'] : 100.0
['Kettle'] -->> ['Oven'] : 100.0
['Kettle'] -->> ['Knife'] : 100.0
['Knife'] -->> ['Kettle'] : 75.0
['Blender'] -->> ['Oven'] : 100.0
['Shampoo'] -->> ['Lotion'] : 80.0
['Shampoo'] -->> ['Razor'] : 80.0
['Razor'] -->> ['Shampoo'] : 80.0
['Deodorant'] -->> ['Lotion'] : 80.0
['Deodorant'] -->> ['Razor'] : 80.0
['Razor'] -->> ['Deodorant'] : 80.0
['Oven'] -->> ['Knife'] : 80.0
['Knife'] -->> ['Oven'] : 100.0
```

```
['Razor'] -->> ['Lotion'] : 80.0
['Kettle'] -->> ['Knife', 'Oven'] : 100.0
['Kettle', 'Oven'] -->> ['Knife'] : 100.0
['Knife'] -->> ['Kettle', 'Oven'] : 75.0
['Knife', 'Kettle'] -->> ['Oven'] : 100.0
['Knife', 'Oven'] -->> ['Kettle'] : 75.0
['Shampoo', 'Deodorant'] -->> ['Razor'] : 100.0
['Razor', 'Shampoo'] -->> ['Deodorant'] : 75.0
['Razor', 'Deodorant'] -->> ['Shampoo'] : 75.0
['Lotion', 'Shampoo'] -->> ['Razor'] : 75.0
['Razor', 'Shampoo'] -->> ['Lotion'] : 75.0
['Lotion', 'Razor'] -->> ['Shampoo'] : 75.0
['Lotion', 'Deodorant'] -->> ['Razor'] : 75.0
['Razor', 'Deodorant'] -->> ['Lotion'] : 75.0
['Lotion', 'Razor'] -->> ['Deodorant'] : 75.0
```

- Association rules for Database 4:

```
For Database Number # 4
Please provide minimum Support value (Only Value): 20
Please provide minimum Confidence value (Only Value): 72
Below are the Association rules :
Total number of items in association list is 15
['Printer'] -->> ['Hoodie'] : 80.0
['Headphone'] -->> ['Hoodie'] : 80.0
['Kettle'] -->> ['Eraser'] : 80.0
['Kettle'] -->> ['Laptop'] : 80.0
['Laptop'] -->> ['Kettle'] : 80.0
['Umbrella'] -->> ['Soap'] : 100.0
['Laptop'] -->> ['Eraser'] : 80.0
['Hoodie', 'Eraser'] -->> ['Marker'] : 80.0
['Hoodie', 'Marker'] -->> ['Eraser'] : 80.0
['Eraser', 'Marker'] -->> ['Hoodie'] : 80.0
['Kettle'] -->> ['Eraser', 'Laptop'] : 80.0
['Eraser', 'Kettle'] -->> ['Laptop'] : 100.0
['Laptop'] -->> ['Eraser', 'Kettle'] : 80.0
['Kettle', 'Laptop'] -->> ['Eraser'] : 100.0
['Eraser', 'Laptop'] -->> ['Kettle'] : 100.0
```

- Association rules for Database 5:

```
For Database Number # 5
Please provide minimum Support value (Only Value): 25
Please provide minimum Confidence value (Only Value): 65
Below are the Association rules :
Total number of items in association list is 13
['Pen'] -->> ['Kettle'] : 83.33333333333333
['Pen'] -->> ['Printer'] : 100.0
['Sunglass'] -->> ['Knife'] : 83.33333333333333
['Kettle'] -->> ['Printer'] : 66.66666666666667
['Shampoo'] -->> ['Lotion'] : 85.71428571428571
['Lotion'] -->> ['Shampoo'] : 75.0
['Knife'] -->> ['Mobile'] : 75.0
['Mobile'] -->> ['Knife'] : 66.66666666666667
```

```
['Jeans'] -->> ['Printer'] : 71.42857142857143
['Pen'] -->> ['Kettle', 'Printer'] : 83.33333333333333
['Pen', 'Kettle'] -->> ['Printer'] : 100.0
['Pen', 'Printer'] -->> ['Kettle'] : 83.33333333333333
['Kettle', 'Printer'] -->> ['Pen'] : 83.33333333333333
```

4.1 Implementing Brute Force Method:

- Before implementing brute force method, an orientation method needs to be implemented which will generates all possible combinations of n elements.

```
In [120]: def orientation(items, n):
              if n == 0:
                  return [[]]

              lst = []
              for i in range(0, len(items)):

                  m = items[i]
                  rest = items[i + 1:]

                  for p in orientation(rest, n-1):
                      lst.append([m]+p)

              return lst
```

- Now, brute force method will be implemented. At first all items were enumerated to generate all possible 1-itemset and 2-temsets. There are 30 items, so there are 435 possible 2-itemsets totally. All items with minimum support value are added to frequent itemset list. Then all possible 3-itemsets have been generated. There are 4060 possible 3-itemsets in total. Frequent itemset from these itemsets have been filtered based on minimum support value. This process was continued until there are no possible k-itemsets as frequent, at which point the brute force method terminates.

```
In [131]: def brute_force_method(data, item_size, min_support, min_confidence):
              freq_items = []
              items = range(0, item_size)
              min_support_value = int(len(data) * min_support / 100.0)

              for k in range(1,item_size+1):
                  new_freq_elements = list(orientation(items,k))
                  print(len(new_freq_elements))
                  new_freq_items = list(filter(lambda x: find_support_from_items(data, x) >= min_support_value, new_freq_elements))
                  if not new_freq_items:
                      break
                  freq_items.extend(new_freq_items)
              return freq_items
```

4.2 Comparing Apriori & Brute Force Method:

- Now, required time for both Apriori algorithm and Brute Force Method with same user defined minimum support and confidence value will be compared for all databases.

```
In [136]: import time
          def compare_bruteforce_apriori(file_name, min_support, min_confidence):
              all_rows = input_database(file_name)
              unique_items = all_distinct_items(all_rows)
              num_unique_items = len(unique_items)
              data = string_to_int(all_rows, unique_items)

              start = time.time()
              freq_items = brute_force_method(data, num_unique_items, min_support, min_confidence)
              association_list = find_association_from_items(data, freq_items, unique_items, min_confidence)
              end = time.time()
              print("Required time for brute force method {}".format(end - start))

              start = time.time()
              freq_items = apriori_algorithm(data, num_unique_items, min_support, min_confidence)
              association_list = find_association_from_items(data, freq_items, unique_items, min_confidence)
              end = time.time()
              print("Required time for apriori Algorithm {}".format(end - start))

              print("---------------------------------------------------------------")
```

```
In [*]: for index in range(1,6):
            print("For Database Number # {} :".format(index))
            file_name = 'database_{}.csv'.format(index)
            min_support, min_confidence = user_input()
            compare_bruteforce_apriori(file_name, float(min_support), float(min_confidence))

        For Database Number # 1 :
        Please provide minimum Support value (Only Value): 15
        Please provide minimum Confidence value (Only Value): 60
        19
        171
        969
        3876
        Required time for brute force method 0.056813716888427734
        Required time for apriori Algorithm 0.0019948482513427734
```

- Results of time comparison are in the below table.

Database	Min Support	Min Confidence	Time for Brute Force Method (sec)	Time for Apriori Algorithm (sec)
Database-1	15	60	0.056813717	0.001994848
Database-2	20	55	0.003989458	0.000997305
Database-3	18	57	0.059969902	0.002993345
Database-4	16	54	2.498095036	0.055394411
Database-5	17	57	11.72953892	0.12955451

Chapter 5 Conclusion

Apriori & Brute force method were implemented from scratch. Both methods were applied on same database with same minimum support and confidence value and produced same association rules. In every case, Apriori algorithm had performed faster than the brute force.

Reference:

My code has been uploaded in below url along with databases and item list.

https://github.com/ronypy/CS634.git