EE4483 CA-Project3 Report

a. What is the maximum number of possible itemsets (including all 1-itemsets) and association rules (including all rules that have zero support and/or zero confidence) that can be generated from this dataset?

The total number of possible itemsets is: 22C1 + 22C2 + 22C3 + ... + 22C21 + 22C22 = 4194303

Source code:

```
from math import factorial as f

def nCr(n,r):
    return f(n)/f(r)/f(n-r)

total = 0
for i in range(1,11):
    total+= nCr(22,i)*2
total+=nCr(22,11) + 1
print(total)
```

Association rules are in the form of X -> Y. The size of X is in [1,21]. When the size of X is i, the size of Y is in [1, 22-i]. The total number of possible association rules is 31368476700.

Source code:

```
from math import factorial as f

def nCr(n,r):
    return f(n)/f(r)/f(n-r)

total = 0
for i in range(1,22):
    combinations_x = nCr(22,i)
    total_y=0
    for j in range(1,22-i):
        total_y+=nCr(22-i, j)
    total+=total_y*combinations_x
print(total)
```

b. What is the maximum size (in terms of number of items) of frequent itemsets (including those with minsup > 0) that can be extracted from this dataset?

Since there are 22 items. The maximum size is 22.

c. Which 2-itemset(s) and 3-itemset(s) have the largest support among all the frequent 2-itemsets and 3-itemsets, respectively?

The maximum support of 2-itemset is: 19
The 2-itemsets having support of 19 are:
Fish, Quiche

Fish, Ham

The maximum support of 3-itemset is: 10
The 3-itemsets having support of 10 are:
Egg, Ham, Milk
Fish, Ham, Quiche

Source code:

```
def support (d,m, n, prefix, suffix):
    '''
    d: dictionary: hash table to store the support of each combination
    m: number of item that has already been chosen
    n: int: the size of the itemset
    prefix: str: a string of all the chosen item.
            If we have chosen Apple and Olive, prefix will be 'AppleOlive'
    suffix: list: the item that can be chosen
    return: list: [most frequent itemset, support of most frequent itemset]
    '''
    if m==n:
        if prefix in d:
            d[prefix]+=1
        else:
            d[prefix]=1
    else:
        for i in range(len(suffix)-(n-m)+1):
            if m+1<n:
                support(d,m+1,n,prefix+suffix[i]+", ", suffix[i+1:] )
            else:
                support(d,m+1,n,prefix+suffix[i], suffix[i+1:] )

transactions = []
i = 0
with open('grocery.basket.txt','r') as f:
    for line in f:
        transactions.append([])
        for word in line.split(','):
            w = word
            if w[-1]=='\n':
                w=w[:-1]
            transactions[i].append(w)
        i+=1
for transaction in transactions:
    transaction.sort()
d = {}
for transaction in transactions:
    if  len(transaction)<2:
        continue
    support(d,0,2,"",transaction)
max_2_support = max(d.values())
print("The maximum support of 2-itemset is: "+ str(max_2_support))
print("The 2-itemset(s) having support of "+str(max_2_support)+" are:")
for k,v in d.items():
    if v==max_2_support:
        print(k)

print()
d = {}
```

```
for transaction in transactions:
    if  len(transaction)<3:
        continue
    support(d,0,3,"",transaction)
max_3_support = max(d.values())
print("The maximum support of 3-itemset is: "+ str(max_3_support))
print("The 3-itemset(s) having support of "+str(max_3_support)+" are:")
for k,v in d.items():
    if v==max_3_support:
        print(k)
```

d. How many frequent itemsets have the minimum support of 12%, 10%, and 5%, respectively?

The numbers of frequent itemsets have the minimum support of 12%, 10% and 5% are 26, 46 and 240 respectively.

Add all the frequent itemsets to the hash table first.

Source code:

```
def support(d,m, n, prefix, suffix):
    if m==n:
        if prefix in d:
            d[prefix]+=1
        else:
            d[prefix]=1
    else:
        for i in range(len(suffix)-(n-m)+1):
            if m+1<n:
                support(d,m+1,n,prefix+suffix[i]+", ", suffix[i+1:] )
            else:
                support(d,m+1,n,prefix+suffix[i], suffix[i+1:] )

transactions = []
i = 0
with open('grocery.basket.txt','r') as f:
    for line in f:
        transactions.append([])
        for word in line.split(','):
            w = word
            if w[-1]=='\n':
                w=w[:-1]
            transactions[i].append(w)
        i+=1
for transaction in transactions:
    transaction.sort()
d = {}
for transaction in transactions:
    for i in range(1,len(transaction)):
        support(d,0,i,"",transaction)
twelve, ten, five=0,0,0
for k,v in d.items():
    if v>= 0.12*150:
        twelve+=1
    if v>=0.1*150:
        ten+=1
    if v>=0.05*150:
        five+=1
print(twelve,ten, five)
```

e. What are the respective percentages of frequent 4-itemsets, 3-itemsets, and 2-itemsets, with respect to all possible itemsets, which have a minimum support of 2%?

The percentages are 0.250, 0.575 and 0.140 respectively.

Source code:

```
def support(d,m, n, prefix, suffix):
    if m==n:
        if prefix in d:
            d[prefix]+=1
        else:
            d[prefix]=1
    else:
        for i in range(len(suffix)-(n-m)+1):
            if m+1<n:
                support(d,m+1,n,prefix+suffix[i]+", ", suffix[i+1:] )
            else:
                support(d,m+1,n,prefix+suffix[i], suffix[i+1:] )

transactions = []
i = 0
with open('grocery.basket.txt','r') as f:
    for line in f:
        transactions.append([])
        for word in line.split(','):
            w = word
            if w[-1]=='\n':
                w=w[:-1]
            transactions[i].append(w)
        i+=1
for transaction in transactions:
    transaction.sort()
d = {}
for transaction in transactions:
    for i in range(1,len(transaction)):
        support(d,0,i,"",transaction)
total = 0
four = 0
three = 0
two = 0
for k,v in d.items():
    if v>=0.02*150:
        total+=1
        if k.count(',')==3:
            four+=1
        if k.count(',')==2:
            three+=1
        if k.count(',')==1:
            two+=1
print(four/total,three/total,two/total)
```

f. How many association rules have a minimum confidence of 50% and

a minimum support of 5% and 10%, respectively?

Association rules that have a minimum confidence of 50% and a minimum support of 5%:
Jam,Olive->Ketchup
Milk,Olive->Fish
Coffee,Milk->Ginger
Egg,Milk->Ham
Ginger,Milk->Coffee
Quiche,Veg->Olive
Jam,Ketchup->Olive
Fish,Milk->Ham
Fish,Milk->Ketchup
Ham,Milk->Fish
Ginger,Yogurt->Milk
Fish,Milk->Olive
Fish,Milk->Egg
Fish,Quiche->Ham
Coffee,Ginger->Milk
Olive,Veg->Quiche
Ketchup,Milk->Fish
Ham,Tea->Fish
Ketchup,Olive->Jam
Ham,Milk->Egg
Fish,Tea->Ham
Fish,Ketchup->Jam
Fish,Jam->Ketchup
Milk,Yogurt->Ginger
Egg,Ham->Milk
Ginger->Milk
Egg,Fish->Milk
Fish,Ketchup->Milk
Ginger,Milk->Yogurt
Fish,Ham->Quiche
Ham,Quiche->Fish
Jam,Ketchup->Fish

Association rules that have a minimum confidence of 50% and a minimum support of 10%:
Ginger->Milk

Source code:

```
def support(d,m, n, prefix, suffix):
    if m==n:
        if prefix in d:
            d[prefix]+=1
        else:
            d[prefix]=1
    else:
        for i in range(len(suffix)-(n-m)+1):
            if m+1<n:
                support(d,m+1,n,prefix+suffix[i]+",", suffix[i+1:] )
            else:
                support(d,m+1,n,prefix+suffix[i], suffix[i+1:] )
```

```
def combinations_recursive(prefix,suffix,r, c):
    if r==len(prefix):
        c.append(prefix)
    else:
        for i in range(len(suffix)-(r-len(prefix))+1):
            combinations_recursive(prefix+[suffix[i]], suffix[i+1:],r,c)

def combinations(n,r):
    c = []
    combinations_recursive([],[i for i in range(n)],r,c)
    return c


def confidence(itemset, d, confidence_table, minsup=0):
    if d[itemset]<minsup:
        return
    itemList = itemset.split(',')
    if len(itemList)==1:
        return
    for r in range(1,len(itemList)):
        X_index = combinations(len(itemList),r)
        X_sets = []
        Y_sets = []
        for index in X_index:
            X_sets.append([])
            Y_sets.append([])
            for i in range(len(itemList)):
                if i in index:
                    X_sets[-1].append(itemList[i])
                else:
                    Y_sets[-1].append(itemList[i])
        for i in range(len(X_sets)):
            key = ','.join(X_sets[i])+'->'+','.join(Y_sets[i])
            value = d[itemset]/d[','.join(X_sets[i])]
            confidence_table[key] = value

transactions = []
i = 0
with open('grocery.basket.txt','r') as f:
    for line in f:
        transactions.append([])
        for word in line.split(','):
            w = word
            if w[-1]=='\n':
                w=w[:-1]
            transactions[i].append(w)
        i+=1
for transaction in transactions:
    transaction.sort()
d = {}
for transaction in transactions:
    for i in range(1,len(transaction)):
        support(d,0,i,"",transaction)
print("Association rules that have a minimum confidence of 50% and a minimum support of 5%:")
confidence_table={}
for k,v in d.items():
    confidence(k, d, confidence_table, minsup=0.05*150)
for k,v in confidence_table.items():
    if v>=0.5:
        print(k)
print()
print("Association rules that have a minimum confidence of 50% and a minimum support of 10%:")
confidence_table={}
for k,v in d.items():
    confidence(k, d, confidence_table, minsup=0.1*150)
for k,v in confidence_table.items():
    if v>=0.5:
        print(k)
```

g. What are the two other items that customers most likely would buy when they buy Milk and Nuts?

Egg and Ham. The confidence of the association rule Milk,Nuts->Egg,Ham is 0.3333, it is the highest among all the association rules where X is Milk, Nuts and Y has size of 2.

Source code:

```
def support(d,m, n, prefix, suffix):
    if m==n:
        if prefix in d:
            d[prefix]+=1
        else:
            d[prefix]=1
    else:
        for i in range(len(suffix)-(n-m)+1):
            if m+1<n:
                support(d,m+1,n,prefix+suffix[i]+",", suffix[i+1:] )
            else:
                support(d,m+1,n,prefix+suffix[i], suffix[i+1:] )

def combinations_recursive(prefix,suffix,r, c):
    if r==len(prefix):
        c.append(prefix)
    else:
        for i in range(len(suffix)-(r-len(prefix))+1):
            combinations_recursive(prefix+[suffix[i]], suffix[i+1:],r,c)

def combinations(n,r):
    c = []
    combinations_recursive([],[i for i in range(n)],r,c)
    return c


def confidence(itemset, d, confidence_table, minsup=0):
    if d[itemset]<minsup:
        return
    itemList = itemset.split(',')
    if len(itemList)==1:
        return
    for r in range(1,len(itemList)):
        X_index = combinations(len(itemList),r)
        X_sets = []
        Y_sets = []
        for index in X_index:
            X_sets.append([])
            Y_sets.append([])
            for i in range(len(itemList)):
                if i in index:
                    X_sets[-1].append(itemList[i])
                else:
                    Y_sets[-1].append(itemList[i])
        for i in range(len(X_sets)):
            key = ','.join(X_sets[i])+'->'+','.join(Y_sets[i])
            value = d[itemset]/d[','.join(X_sets[i])]
            confidence_table[key] = value

transactions = []
i = 0
with open('grocery.basket.txt','r') as f:
    for line in f:
        transactions.append([])
        for word in line.split(','):
            w = word
            if w[-1]=='\n':
                w=w[:-1]
            transactions[i].append(w)
        i+=1
for transaction in transactions:
    transaction.sort()
d = {}
for transaction in transactions:
    for i in range(1,len(transaction)):
        support(d,0,i,"",transaction)
confidence_table={}
for k,v in d.items():
    confidence(k, d, confidence_table)
largest = list(confidence_table.keys())[0]
for k,v in confidence_table.items():
    i = k.find('->')
    X = k[:i].split(',')
    Y = k[i+2:].split(',')
    if X==['Milk','Nuts'] and len(Y)==2 and v>confidence_table[largest]:
        largest=k
print(largest,confidence_table[largest])
```

h. List three association rules that have the highest support with 100% confidence?

Apple,Ham,Milk->Egg. The support of this association rule is 6 (4%). And it is the highest among all the association rules with 100% confidence.

Source code:

```
def support(d,m, n, prefix, suffix):
    if m==n:
        if prefix in d:
            d[prefix]+=1
        else:
            d[prefix]=1
    else:
        for i in range(len(suffix)-(n-m)+1):
            if m+1<n:
                support(d,m+1,n,prefix+suffix[i]+",", suffix[i+1:] )
            else:
                support(d,m+1,n,prefix+suffix[i], suffix[i+1:] )
```

```
def combinations_recursive(prefix,suffix,r, c):
    if r==len(prefix):
        c.append(prefix)
    else:
        for i in range(len(suffix)-(r-len(prefix))+1):
            combinations_recursive(prefix+[suffix[i]], suffix[i+1:],r,c)

def combinations(n,r):
    c = []
    combinations_recursive([],[i for i in range(n)],r,c)
    return c


def confidence(itemset, d, confidence_table, minsup=0):
    if d[itemset]<minsup:
        return
    itemList = itemset.split(',')
    if len(itemList)==1:
        return
    for r in range(1,len(itemList)):
        X_index = combinations(len(itemList),r)
        X_sets = []
        Y_sets = []
        for index in X_index:
            X_sets.append([])
            Y_sets.append([])
            for i in range(len(itemList)):
                if i in index:
                    X_sets[-1].append(itemList[i])
                else:
                    Y_sets[-1].append(itemList[i])
        for i in range(len(X_sets)):
            key = ','.join(X_sets[i])+'->'+','.join(Y_sets[i])
            value = d[itemset]/d[','.join(X_sets[i])]
            confidence_table[key] = value

transactions = []
i = 0
with open('grocery.basket.txt','r') as f:
    for line in f:
        transactions.append([])
        for word in line.split(','):
            w = word
            if w[-1]=='\n':
                w=w[:-1]
            transactions[i].append(w)
        i+=1
for transaction in transactions:
    transaction.sort()
d = {}
for transaction in transactions:
    for i in range(1,len(transaction)):
        support(d,0,i,"",transaction)
confidence_table={}
for k,v in d.items():
    confidence(k, d, confidence_table,6)
for k,v in confidence_table.items():
    if v==1:
        print(k,v)
```

i. Do you find any “interesting” rules? What are they? Briefly explain why.

When the minimum support is higher, it takes much shorter time to calculate the confidence of the association rules that meet the minimum support. When I set the minimum support 5% or 10%, my program returned the result immediately. And it takes a few seconds when there is no minimum support requirement.

Although 5% and 10% seem to be small, only a few association rules meet this requirement. That is why the running time is much shorter when this minimum requirement is set.

j. State what software tool(s) and functions you use to complete this homework.

Python3.5 and its built-in math library.