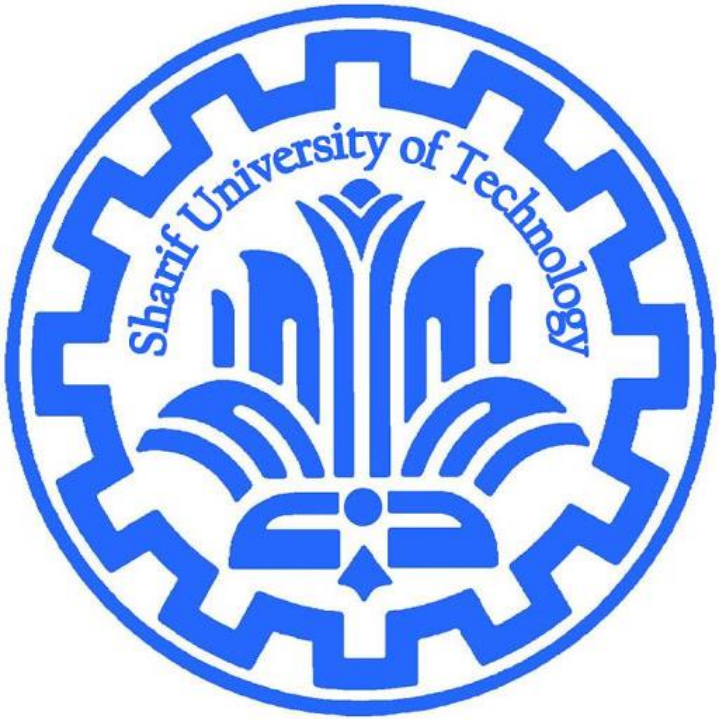

Massive Data Analysis HW2

Ahmadreza Tavana: 98104852

Professor: Dr.Gholampour

Question 1

Part a)

At first we write down the formula in a formal case:

$$conv(A \to B) = \frac{1 - S(B)}{1 - conf(A \to B)} = \frac{P(B^c)}{1 - P(B|A)} = \frac{P(B^c)}{P(B^c|A)}$$

In the other hand of A and B be independent we have:

$$P(B|A) = P(B) \to P(B^c|A) = P(B^c)$$

Now we can understand that $P(B^c|A)$ is equal to being Association rule wrong. Now according to these equations we can say that $conv(A \to B)$ is equal to being Association rule wrong when A and B be independent divide to probability of being what we saw.

For example, is conv be equal to 1.5, it means that if A and B be independent, this rule will have 50 percent of more wrongness. (it means that we have less wrongness in compare with independency.)

Part b)

conf and conv are asymmetric but lift is symmetric. For lift we have:

$$lift\ (A \to B) = \frac{conf(A \to B)}{S(B)} = \frac{P(B|A)}{P(B)} = \frac{P(B \cap A)}{P(B)P(A)} = \frac{conf(B \to A)}{S(A)} = lift\ (B \to A)$$

Generally, we know that $P(B|A) \neq P(A|B)$. Now for conv, we prove that it's not symmetric with an example. Assume that $A = B^c$, so we have:

$$conv(A \to B) = \frac{P(B^c).P(A)}{P(B^c \cap A)} = P(A)$$

$$conv(B \to A) = \frac{P(A^c).P(B)}{P(A^c \cap A)} = P(B)$$

But we can choose B in order that P(A) = P(B).

Part c)

The weakness of this definition is that we did not consider the probability of B. if we don't change A and make B larger and larger only P(B|A) gets larger. But this getting larger doesn't mean that the conf is well. This means that with getting probability of result bigger, conf will be increased which shows us that conf is not a proper parameter for measuring the power of $A \rightarrow B$.

Question 2

From supposes that we have in the question, we can assume that we have 100 subsets and the items are integer numbers from 1 to 100 because all of these numbers are repeated in baskets.

Now we write down subsets which as support of more than 5 and then figure out whether that subset is maximal or not. The subsets are as below. (I wrote down only 20 most frequent subsets that their supports are more than 5 and refuse to write subsets which have less than 5 supports.)

Subset	Support	Maximal
(1)	100	No
(1,2)	50	No
(1,3)	33	No
(1,2,4)	25	No
(1,5)	20	No
(1,2,3,6)	16	No
(1,7)	14	No
(1,2,4,8)	12	No
(1,3,9)	11	No
(1,2,5,10)	10	No
(1,11)	9	Yes
(1,2,3,4,6,12)	8	Yes
(1,13)	7	Yes
(1,2,7,14)	7	Yes
(1,3,5,15)	6	Yes
(1,2,4,8,16)	6	Yes
(1,17)	5	Yes
(1,2,3,6,9,18)	5	Yes
(1,19)	5	Yes
(1,2,4,5,10,20)	5	Yes

As it can be seen, itemsets of 11 to 20 are maximal because superset of each of them has support less than 5 (because each itemset number more than 20 has support less than 5) and also their support is more than 5.

Question 3

In this question we use A-Priori algorithm for finding most frequent itemsets. For this question at first we find the most single frequent items by map reduce by filtering them on the support value. If the count of the itemset is below the support, we remove it from the whole dataset. The result for most single itemsets is as below:

```
The most single frequent items:
('DAI62779', 6667)
('FRO40251', 3881)
('ELE17451', 3875)
('GRO73461', 3602)
('SNA80324', 3044)
('ELE32164', 2851)
('DAI75645', 2736)
('SNA45677', 2455)
('FRO31317', 2330)
('DAI85309', 2293)
('ELE26917', 2292)
('FRO80039', 2233)
('GRO21487', 2115)
('SNA99873', 2083)
('GRO59710', 2004)
('GRO71621', 1920)
('FRO85978', 1918)
('GRO30386', 1840)
('ELE74009', 1816)
('GRO56726', 1784)
('DAI63921', 1773)
('GRO46854', 1756)
('ELE66600', 1713)
('DAI83733', 1712)
('FRO32293', 1702)
('ELE66810', 1697)
('SNA55762', 1646)
('DAI22177', 1627)
('FRO78087', 1531)
```

Now for the most 2-member frequent itemsets, we create the pairs with most single frequent itemsets and then try to find the most 2-member frequent itemsets. The result of this approach is as below:

```
The most 2-member frequent itemsets:
(('DAI62779', 'ELE17451'), 1592)
(('FRO40251', 'SNA80324'), 1412)
(('DAI75645', 'FRO40251'), 1254)
(('FRO40251', 'GRO85051'), 1213)
(('DAI62779', 'GRO73461'), 1139)
(('DAI75645', 'SNA80324'), 1130)
(('DAI62779', 'FRO40251'), 1070)
(('DAI62779', 'SNA80324'), 923)
(('DAI62779', 'DAI85309'), 918)
(('ELE32164', 'GRO59710'), 911)
(('FRO40251', 'GRO73461'), 882)
(('DAI62779', 'DAI75645'), 882)
(('DAI62779', 'ELE92920'), 877)
(('FRO40251', 'FRO92469'), 835)
(('DAI62779', 'ELE32164'), 832)
(('DAI75645', 'GRO73461'), 712)
(('DAI43223', 'ELE32164'), 711)
(('DAI62779', 'GRO30386'), 709)
(('ELE17451', 'FRO40251'), 697)
(('DAI85309', 'ELE99737'), 659)
(('DAI62779', 'ELE26917'), 650)
(('GRO21487', 'GRO73461'), 631)
(('DAI62779', 'SNA45677'), 604)
(('ELE17451', 'SNA80324'), 597)
(('DAI62779', 'GRO71621'), 595)
(('DAI62779', 'SNA55762'), 593)
(('DAI62779', 'DAI83733'), 586)
(('ELE17451', 'GRO73461'), 580)
(('GRO73461', 'SNA80324'), 562)
```

Now for the most 3-member frequent itemsets, we create the pairs with 2-member frequent itemsets and then try to find the most 3-member frequent itemsets. The result of this approach is as below:

```
The most 3-member frequent itemsets:
(('DAI75645', 'FRO40251', 'SNA80324'), 550)
(('DAI62779', 'FRO40251', 'SNA80324'), 476)
(('FRO40251', 'GRO85051', 'SNA80324'), 471)
(('DAI62779', 'ELE92920', 'SNA18336'), 432)
(('DAI62779', 'DAI75645', 'SNA80324'), 421)
(('DAI62779', 'ELE17451', 'SNA80324'), 417)
(('DAI62779', 'DAI75645', 'FRO40251'), 412)
(('DAI62779', 'ELE17451', 'FRO40251'), 406)
(('DAI75645', 'FRO40251', 'GRO85051'), 395)
(('DAI62779', 'FRO40251', 'GRO85051'), 381)
(('ELE17451', 'FRO40251', 'SNA80324'), 353)
(('DAI62779', 'ELE17451', 'ELE92920'), 345)
(('FRO40251', 'FRO92469', 'SNA80324'), 343)
(('DAI62779', 'DAI85309', 'ELE17451'), 339)
(('DAI62779', 'DAI75645', 'ELE17451'), 328)
(('DAI62779', 'FRO40251', 'GRO73461'), 315)
(('DAI62779', 'ELE32164', 'GRO59710'), 301)
(('DAI75645', 'ELE17451', 'SNA80324'), 300)
(('DAI75645', 'FRO40251', 'GRO73461'), 293)
(('DAI75645', 'ELE17451', 'FRO40251'), 292)
(('DAI43223', 'ELE32164', 'GRO59710'), 287)
(('DAI43223', 'DAI62779', 'ELE32164'), 287)
(('DAI62779', 'ELE17451', 'ELE32164'), 277)
(('DAI62779', 'DAI85309', 'ELE99737'), 272)
(('DAI62779', 'DAI75645', 'GRO73461'), 261)
```

Question 4

In this question after setting up the pre requires we map the data with key of ('Final_CAR_KEY', 'PASS_DAY_TIME') and value of ('DEVICE_CODE') which shows the pathway of the cars. Now we have a RDD which has rows that each row contains pathway of each date with license plate of each passed car. The result of take 5 of the rdd is as below:

```
[(('10477885', datetime.date(2021, 6, 1)), 200501),
 (('87625017', datetime.date(2021, 6, 1)), 155),
 (('8652928', datetime.date(2021, 6, 1)), 631757),
 (('8548123', datetime.date(2021, 6, 1)), 631757),
 (('24715264', datetime.date(2021, 6, 1)), 631757)]
```

Now we group_by the above RDD by its key for each car in every date to have all pathway that each car passed in a list. Notice that in this part we use set instead of list to not having repeated pathways. This approach base on the usage of this part does not have huge effect on the whole of the problem. The result of take 5 of the rdd is as below:

```
[(('69939810', datetime.date(2021, 6, 1)), {206602, 631830, 900233}),
 (('11046172', datetime.date(2021, 6, 1)), {206602}),
 (('29077699', datetime.date(2021, 6, 1)),
  {119,
   128,
   206602,
   208602,
   210602,
   631367,
   631763,
   631829,
   900135,
   900233,
   900234,
   900240,
   900246,
   900266,
   22010111,
   22010123}),
 (('40682798', datetime.date(2021, 6, 1)), {206602}),
 (('48823778', datetime.date(2021, 6, 1)), {206602})]
```

A-Priori Function

NOTE: I DID ALL MY PROCESSES WITH A SAMPLE OF THE WHOLE DATASET AND TA ACCEPTED THAT. TNX!

Now we define A-Priori function. The steps of writing down the function is as below:

For having baskets, we don't need to keep license plates, so we get baskets without them. Then we count the number of baskets and then with using threshold, we get minimum support. I explain about MIN_COUNT in the end of this notebook and report.

I explain some other functions are written and I explain them in the following and cells and then I write the original function for A-Priori function.

generate_combinations

This function gets frequent itemsets of its previous step and generates candidate of the next step. At first it gets the frequent remaining items from whole of the frequent itemssets and then sort them. Then for each of frequent itemset, it gets the largest element and then compare to remaining items of the collection and if that item be larger than that one, it adds it to the end of that collection and generates a new candidate. In this function it doesn’t need to add smaller elements to the collections.

For example, suppose [b,c,d] collection which was frequent in the previous step and the remaining collection of frequent is [a,b,c,d,z] (these elements are sorted). From these remaining elements, z is the only element which is larger than other elements so a new candidate is [b,c,d,z]. But [a,b,c,d] is not a candidate. Assume that it is frequent. We know that all subsets of a frequent set are also frequent.

Now if [a,b,c] was frequent, so it would be available in the frequent list and create its combination in checking [a,b,c] but if it isn't frequent, so [a,b,c,d] wouldn't be frequent and was not created correctly.

get_new_frequents

This function gets all the baskets as input and gets the most frequents itemsets of the next step. In this function we get help from a new function which is defined as "count_freq" which gets a basket as input and for each candidate of that step, if that candidate exists on that basket, it creates an output as (candidate,1). Finally with counting all candidates, we can get most frequent itemsets. The total code of this part is as below:

```
def apriori(baskets_rdd, MIN_COUNT, verbose=False):
  frequent_items_rdd = baskets_rdd.flatMap(lambda basket: [((item,),1) for item in basket]).reduceByKey(lambda x,y: x+y).filter(lambda x: x[1] > MIN_COUNT)

  def generate_combinations(old_combinations):
    """
    Input old_frequent_itemsets, and create new candidates from it
    """

    def generate_combinations_util(old_combination):
      """
      lambda function that maps an old combination to a number of new candidate combinations
      """
      old_combination = old_combination[0]
      old_combination_max_item = old_combination[-1]

      # Can do here numpy way
      bigger_items = remaining_items[remaining_items>old_combination_max_item]
      new_candidates = []
      for x in bigger_items:
        new_candidates.append( old_combination + (x,) )

      return new_candidates

    remaining_items = np.array(old_combinations.flatMap(lambda x: x[0]).distinct().sortBy(lambda x: x).collect())

    new_combinations = old_combinations.flatMap(generate_combinations_util)
    return new_combinations

  def get_new_frequents(candidates):
    _candidates = candidates.collect()
    def count_freq(basket):
      candidates_present = []
      for candidate in _candidates:
        if set(candidate) <= set(basket):
          candidates_present.append( (candidate,1) )
```

```
  frequent_itemsets_rdd = baskets_rdd.flatMap(count_freq).reduceByKey(lambda x,y: x+y).filter(lambda x: x[1]>MIN_COUNT)

  return frequent_itemsets_rdd

if verbose:
  print('MIN_COUNT is: ', MIN_COUNT)
k = 1
frequent_itemsets_rdd = frequent_items_rdd
frequent_itemsets_rdds = []
while frequent_itemsets_rdd.count() != 0:
  frequent_itemsets_rdds.append(frequent_itemsets_rdd)
  if verbose:
    print('-----------------loop start----------------------')
  k += 1
  candidates = generate_combinations(frequent_itemsets_rdd)
  frequent_itemsets_rdd = get_new_frequents(candidates)
  if verbose:
    print('Itemsets of size ', k, ', count: ', frequent_itemsets_rdd.count())
    print('sample: ')
    print(frequent_itemsets_rdd.take(10))

return frequent_itemsets_rdds
```

The result of A-Priori algorithm for a sample is as below:

```
MIN_COUNT is:  15
-----------------loop start----------------------



Itemsets of size  2 , count:  811
sample:
[((101301, 900101), 30), ((900102, 100701100), 36), ((900182, 100701100), 33), ((145, 100700841), 20), ((900222, 900228), 28), ((900222, 100700868), 171),
((100700866, 100700868), 47), ((900234, 900276), 34), ((900235, 100700871), 41), ((209103, 900235), 18)]
-----------------loop start----------------------



Itemsets of size  3 , count:  184
sample:
[((900212, 900244, 22009977), 23), ((631765, 900164, 900276), 17), ((631765, 900164, 100700820), 35), ((631765, 900276, 100700820), 21), ((900101, 900259,
100700841), 35), ((900155, 900222, 100700868), 50), ((205802, 900215, 900234), 19), ((142, 900215, 900234), 16), ((205802, 212802, 900233), 16), ((900215, 900234,
900256), 23)]
-----------------loop start----------------------



Itemsets of size  4 , count:  11
sample:
[((22010087, 22010088, 22010094, 22010095), 28), ((900101, 900212, 900244, 100700839), 16), ((900193, 900212, 900244, 100700839), 16), ((900102, 900142, 900212,
900244), 18), ((900142, 900202, 900212, 900244), 16), ((900142, 900212, 900244, 900249), 17), ((900142, 900212, 900244, 100700853), 54), ((209103, 900265,
100700804, 100700834), 21), ((900142, 900152, 900212, 900244), 18), ((231, 900236, 900255, 100700841), 20)]
-----------------loop start----------------------
Itemsets of size  5 , count:  0
sample:
[]
```

Now in the following we represent SON algorithm.

SON

For SON algorithm, we use the function which I was defined in the last part. For sample in below cell and the support, the application and explanation is the same. In this part, we deploy a random number between 0 to number of partitions-1 which number of partitions shows that RDD will be divided to several parts. Then with filtering on deployed number, RDD will be divided to almost equal partitions. The minimum support of each of these partitions can be reached base on their size which are almost equal. Notice that because the size of each partition is small, the minimum support would be small too. For more confidence we multiple the support to 0.9 to not miss anything.

Now we deploy A-Priori algorithm in the all parts and then we union the output of these algorithms to get frequent itemsets of the algorithms. But these itemsets may contain false positive because of that we pass through all of the data one more time and count the candidates and filter them. We do this work exactly like A-Priori algorithm.

The result of this algorithm was exactly as the same as A-Priori algorithm (Actually we expected this happen because SON algorithm does'nt have false positive and false negative.)

Now in the next part we compare the results and explain the way of selecting support and other used parameters.

Result, Comparison, Selecting support and other parameters

With setting threshold to 0.02 or in the other hand by setting support equal to 30408 and on the whole data of the sample, frequent items are equal to 27 which a 10 sample of that is as below:

((900107,), 38371), ((900155,), 45958), ((900207,), 35631), ((22010119,), 30749), ((900139,), 31477), ((900191,), 31379), ((100700824,), 34662), ((900244,), 61075), ((900164,), 37506), ((900236,), 41652)

The most frequent 2-member-set is ((900212,900244),42733) which the right number of each set is the repeat of that set.

For testing the code for bigger data, I get a sample of 0.01 from the data and put the threshold to 0.001 and the result is as below:

SAMPLE_SIZE = 0.01

THRESHOLD = 0.001

MIN_COUNT is: 15

Itemsets of size 2, count: 811

sample:

[((101301, 900101), 30), ((900102, 100701100), 36), ((900182, 100701100), 33), ((145, 100700841), 20), ((900222, 900228), 28), ((900222, 100700868), 171), ((100700866, 100700868), 47), ((900234, 900276), 34), ((900235, 100700871), 41), ((209103, 900235), 18)]

Itemsets of size 3, count: 184

sample:

[((900212, 900244, 22009977), 23), ((631765, 900164, 900276), 17), ((631765, 900164, 100700820), 35), ((631765, 900276, 100700820), 21), ((900101, 900259, 100700841), 35), ((900155, 900222, 100700868), 50), ((205802, 900215, 900234), 19), ((142, 900215, 900234), 16), ((205802, 212802, 900233), 16), ((900215, 900234, 900256), 23)]

Itemsets of size 4, count: 11

sample:

[((22010087, 22010088, 22010094, 22010095), 28), ((900101, 900212, 900244, 100700839), 16), ((900193, 900212, 900244, 100700839), 16), ((900102, 900142, 900212, 900244), 18), ((900142, 900202, 900212, 900244), 16), ((900142, 900212, 900244, 900249), 17), ((900142, 900212, 900244, 100700853), 54), ((209103, 900265, 100700804, 100700834), 21), ((900142, 900152, 900212, 900244), 18), ((231, 900236, 900255, 100700841), 20)]

How will pair of far cameras be removed in the process and don’t appear in the output?

In the map reduce the dataset, if the cameras are far the device code would not be near so the specific cars would not pass the same way so the far cameras data will be removed automatically.