Association Rules

Question 1:

What is association rule mining?

Answer:

Given a database of transactions, where each transaction consists of a set of items, association discovery finds all the item sets that frequently occur together the so called frequent itemsets, and also the rules among them. An example of an association could be than, “35% of people who buy bread also buy cheese”. The association rules use two main concepts: support and confidence. The support counts is the frequency of an itemset occurred among the transactions, whereas, the confidence for an association rules X=>Y (“=> “ means gathered together) is the support(X)/support(X and Y). Where X and Y is the item(s).

Question 2:

Given a set of transaction below, find the 1, 2 and 3, large itemsets that have minimum support 75% and association rules that have minimum confidence 80%.

t100- crab, milk, cheese, bread

Answer:

The frequent itemsets are (minsup = 3/4):

F1 = milk (4), cheese (3), bread (4)
F2 = {milk,cheese} (3), {milk,bread} (4), {cheese,bread} (3)
F3 = {milk,cheese,bread}(3)

The rules with 80% confidence are:
milk and cheese -> bread (conf = 3/3 = 100%) supp = 3
cheese and bread -> milk (conf = 3/3 = 100%) supp = 3

Question 3:

Besides mining market basket database, list two possible uses of association rules:

Answer:

1. Fraud detection in financial
2. Clustering - e.g. clusters customers buying behavior
3. Web mining - finds the usage pattern of customers that visited a company web sites
**Question 4:**

What are the main drawbacks and advantages of Count Distributions algorithm?

**Answer:**

**Drawbacks:**
1. Synchronization at the end of each iteration to exchange the local support counts.
2. Redundancy since each processor produced exactly the same hash tree

**Advantages:**
1. No data movement is necessary, which save data communication cost
2. Direct implementations

**Question 5:**

Write the pseudocode and describe the serial Apriori algorithm?

**Answer:**

1. \( L_1 = \{ \text{large 1-itemsets} \} \)
2. for \( (k=2; L_{k-1} \neq \emptyset; k++) \) do
3. \( C_k = \text{apriori-gen}(L_{k-1}) \)
4. for all transactions \( t \in D \) do begin
5. \( C_t = \text{subset}(C_k, t); \)
6. for all candidates \( c \in C_t \) do
7. \( c\text{-count}++; \)
8. end
9. \( L_k = \{ c \in C_k \mid c\text{-count} \geq \text{minsupport} \} \)
10. end

The Apriori is an iterative algorithm that make multi-passes on the transaction data. Initially the frequent 1-itemsets \( L_1 \) is found (1.). The candidate itemset \( C_k \) for the next iteration is generated from the previous iteration large itemset \( L_{k-1} \) (3). The transaction data is scanned once (4) to update the candidate itemset support count if the candidate itemset \( c \) is subset of the transaction (5,6,7). The candidate itemsets is then check for minimum support constraint, and those satisfy this constraint is the large itemset for current iteration (9). The iteration is continue until no more large itemset.

**Question 6:**

Suggest two ways to overcome the memory management problems of maintaining the candidates itemset when the main memory space is not enough.

**Answer:**

1. Let \( m \) be the total candidate itemsets, and only \( n \ (< m) \) of them can be hold by the main memory space. The candidate generation algorithm could generate the \( n \) candidates itemset, make a pass to the database to find the support for these candidates itemset. After finding the large itemset from the \( n \) candidate itemset, these candidates itemset are discarded. The candidate generation algorithm then generates the remaining \((m-n)\)
candidates itemset and repeat the same process until all candidates itemset being
generated.
2. It is also possible to generate all candidates itemset at once, and the itemsets that cannot
be accommodated by the main memory is stored into the main memory.
3. The simplest way could be just discard the remaining candidate itemsets that cannot fit
into the main memory. This will results in lost of some of the potential large itemset.

**Question 7:**

Describes two algorithms that can be used to generate association rules from large itemset?

**Answer:**

**Simple Algorithm**

for all large itemsets $l_k$, $k \geq 2$
do 
call genrules($l_k$, $l_k$);

//The genrules generates all valid rules $a \Rightarrow (l_k-a)$, for all $a \subset a_m$
procedure genrules($l_k$: large $k$-itemset, $a_m$: large $m$-itemset)
$A= \{ (m-1)$-itemsets $a_{m-1} \mid a_{m-1} \subset a_m \}$;
for all $a_{m-1} \in A$ do begin
   conf = support($l_k$)/support($a_{m-1}$);
   if(conf $\geq$ minconf) then begin
      output the rule $a_{m-1} \Rightarrow (l_k-a_{m-1})$, with confidence = conf and support =
support($l_k$);
      if($m-1 > 1$) then
         call genrules($l_k$, $a_{m-1}$); //to generate rules with subsets of $a_{m-1}$ as the
antececdents
   end
end for

**Faster algorithm**

For all large $k$-itemsets $l_k$, $k \geq 2$ do begin
   $H_1 = \{ \text{ consequents of rules derived from } l_k \text{ with one item in the consequent } \}$;
   call ap-genrules($l_k$, $H_1$)
end

procedure ap-genrules($l_k$: large $k$-itemset, $H_m$: set of $m$-item consequents)
if($k > m+1$) then begin
   $H_{m+1} = \text{apriori-gen}(H_m)$;
for all $h_{m+1} \in H_{m+1}$ do begin
   conf = support($l_k$)/support($l_k=h_{m+1}$);
   if(conf $\geq$ minconf) then
      output the rule($l_k-h_{m+1}$) with confidence = conf and support = support ($l_k$);
else
   delete $h_{m+1}$ from $H_{m+1}$
end
   call ap-genrules($l_k$, $H_{m+1}$);
end

genrules() is Apriori candidate generation algorithm.
Question 8:

How can we use a database sampling method to speed up association rules generation?

Answer:

Scanning the whole database is the most expensive operations in the Apriori association rules algorithm. Many attempts have been made to reduce the time of database scanning step. In sampling method, instead of using the whole database to find the support counts of candidate itemset, the algorithm may simply draw let say 60% random examples from the database to use as to represents the whole database. These samples are then used to find the support counts of candidate itemsets, which will significantly reduced the database scanning time. Further time can be reduced if these samples can fit into main memory because scanning the samples in main memory is much faster that in disc. The main drawback of this method is that the samples may not be large enough to represents the actual statistics in the whole database. Some large itemset may be left out.

Question 9:

Besides the database sizes, what are the other factors that effect the times to generate association rules?

Answer:

1. Average number of items in a transaction - if more items are available in the transactions, the number of level will be increased, thus more iteration is needed to generate the completes association rules.

2. Total number of difference items (product ID) - if the total number of difference items (product ID) is increased and the average transaction length is keep constant, the probability of the same items repeated in difference transaction will be becomes lower. In the first iteration, the number of 1-large itemset may be increased, but when the number of level (k) increase, the support counts for larger length itemset will be decreased, thus the execution time will be decreased.

3. Minimum support - if minimum support is decrease, the number of candidates itemset generated will be increased, thus more time is required to generate the association rules.

Question 10:

Describes how hash tree data structure can be used for fast support counting? What are the advantages of hash tree as compared to the scanning all the candidate itemset method for support counting?

Answer:

The hash tree data structure is use to stored the candidate itemset in each iteration. A candidate itemset is added to the hash tree by hash on the d-th item at level d of the hash tree. When it reached the leaf node, the candidate is added the candidates list. During support counting, we need to find all candidates itemset that is contained in a transaction. We hash on the items in the transaction to traverse the hash tree until we reached the leaf nodes that may contains the candidates that is supported by this transaction.

As compared to the support updating method by scanning through each candidate itemset, the hash tree structure saves time by reducing the numbers of candidate itemset that needs to be check. The hash tree structure ensures only the potential candidates itemset are checked by traversing to the correct part to the leaf node.
Sequential rules

Question 1:

Derived that, during updating the support counts, if we reached an interior node by hashing on an item \( x \) whose transaction time is \( t \) of a customer sequence \( q \), then we only need to apply the hash function to items in \( q \) with transaction time in \([t - \text{wsize}, t + \text{max}(\text{wsize}, \text{maxgap})] \). Where \( \text{wsize} \) is the window size, \( \text{maxgap} \) is maximum gap and \( \text{max} \) is the maximum value function.

Answer:

Proof:

Consider a candidate sequence \( s \) with two consecutive items \( x \) and \( y \). Let \( x \) be contained in a transaction in \( d \) whose transaction-time is \( t \). For \( d \) to contain \( s \), the transaction-time corresponding to \( y \) must be in \([t - \text{wsize}, t + \text{wsize}] \) if \( y \) is part of the same element as \( x \), or in the interval \((t, t + \text{maxgap}] \) if \( y \) is a separated element. Hence if we reached this node by hashing on an item \( x \) with transaction time \( t \), \( y \) must be contained in a transaction whose transaction time is in the interval \([t - \text{wsize}, t + \text{max}(\text{wsize}, \text{maxgap})] \) for the data sequence \( d \) to support the sequence.

Question 2:

Write down the Sequential rules candidates generation algorithms using the variables and function definitions below.

```cpp
list<Sequence *> large_sequence; /* list contains large sequences for the k-th iteration */
HashTable hashtable; /* hash table entries contains all large sequences in iteration < k */
bool Can_Join(Sequence & sq1, Sequence & sq1);
Function Can_Join() returns true if two sequence can be joined together, otherwise return false.
You can create your own functions or variables. No detail implementation is necessary.
```

Answer:
```c
list<Sequence*>*::iterator seqbeg1 = large_sequence->begin();
list<Sequence*>*::iterator seqend1 = large_sequence->end();
list<Sequence*>*::iterator seqbeg2;
list<Sequence*>* cand_sequence;
Sequence* contiguous_subsequence, new_candidate;
bool flag;

while(seqbeg1 != seqend1)
{
    seqbeg2 = large_sequence->begin();
    while(seqbeg2 != seqend1)
    {
        if(seqbeg2 == seqbeg1)
        {
            seqbeg2++; continue
        }

        if(Can_Join(**seqbeg1, **seqbeg2))
        {
            new_candidate = new_cand(*seqbeg1, *seqbeg2);
            flag = true;
            while((contiguous_subsequence = Gencontiguous_subseq(new_candidate)) != NULL)
            {
                if(hashtable.lookup(contiguous_subsequence) != true)
                {
                    flag =false;
                    break;
                }

                if(flag == true)
                {
                    cand_sequence.push_back(new_candidate);
                }
            }

            seqbeg2++;
        }

    seqbeg1++;
}
```

**Question 3:**

Discuss how MPI functions can be used to achieve parallelism in the count distribution sequential rules.

**Answer:**

In the count distribution algorithm, there are two main steps that can be achieved using MPI functions. That is, finding the large 1-sequence, and exchanging sequences local support count at the end of each iteration. In the first, we can use MPI_Allreduce() function with MPI_SUM binary operator to sum up the candidate 1-sequence support count from all processor.

Secondly, by using MPI_Allreduce() again, we can perform the same effect to find the global candidate sequences support counts. In this way, we create a parallelism by distribute the workload of complete data to only a partial local data partition.
Question 4:

In which implementation environment does the data distribution is preferred to the count distribution algorithm?

Answer:

The main drawback of data distribution sequential rules is their large communication overhead result from sending the complete local partition to all other processor. Thus, this algorithm is only suitable in the implementation environment where there have higher resources of buffer available for asynchronous communication and fast communication network. On the other hand, the count distribution algorithm involves less data movement.

Question 5:

Objective: to study how the data distribution work. Create a data parallel program that receives/sends data from/to other processor. You must use the non-blocking point-to-point communication, that is MPI_Isend() and MPI_Irecv() MPI functions. Assume initially one data file is created for each processor. At the end of the execution, each processor should have a complete data from other processor that saved into a file.

Answer:

Please refer to the DataDistribute() module for the data distribution algorithm.

Question 6:

Describe how the array of lists data structure could be used to fast find the existence of an element/itemset in a data sequence after time $t$?

Answer:

Assume the maximum item ID for all the data sequences is $L$. An array of size $L$ will be created, where the $i$-th index of the array belongs to item $i$. Each array entry is a list that chained the transaction time where this item occurred. The list records are sorted in ascending order by time $t$. By organizing this way, we can easily find the existent of an item $j$ at time $t_1$ by locating the $j$ entry of the array and then traverse the list looking for time $t_1$.

An element consists of one or more items. Thus, to look whether an element $e$ exists or not in a data sequence after a time $t_1$, we first create an array of list for this sequence. Then for each item in $e$, we find the time after time $t_1$. If we can find all the items after time $t_1$, then $e$ exists in the sequence, otherwise it does not.
**Question 7:**

Given frequent 3-sequences below, determine the large candidates 4-sequences.

\(<(1,2) (3)>\), \(<(1,2) (4)>\), \(<(1) (3,4)>\)  \(<(1,3) (5)>\), \(<(2) (3, 4)>\), \(<(2) (3) (5)>\)

**Answer:**

\(<(1,2) (3,4)>\)

**Question 8:**

When performing candidate support counting, we need to check whether a candidate sequence contains in a data sequence. Describe the two phases of the contains test?

**Answer:**

The contains test is performed in two phases, the forward phase, and the backward phase. Let \(d\) is the data sequence and \(s\) the elements in the data sequence.

**Forward phase**

The algorithm finds the successive elements \(s\) in \(d\) as long as the difference between the end-time and start-time of the previous element is less than max gap. If the difference is more than max gap, the algorithm is switched to backward phase. If an element is not found, the data sequence \(d\) does not contain \(s\).

**Backward phase**

Let \(s_{i+1}\) is the current element that does not satisfy the max-gap constraint. The algorithm backtracks to find the existence of the previous element \(s_i\) (that has found in the data sequence) in the data sequence \(d\) after time \(t_{i}\)-max-gap. Where \(t_i\) is the end-time of element \(s_{i+1}\). If we found the element \(s_i\) in \(d\), we may need to backtrack on the element \(s_{i+1}\) again because the max gap constraint between \(s_i\) and \(s_{i+1}\) may be violated. We stop backtracking until the max gap constraint between the last backtrack element and it previous element is satisfied or there is no more element to backtrack. If one of the backtrack element is not found, then \(d\) does not contains \(s\). We then switch to forward phase start from the next element after the element last backtrack.

**Question 9:**

How does the Hash Tree data structure help to reduce the number of candidate sequences that needs to be count?

**Answer:**

The hash tree data structure is used to store the candidate sequences. A hash tree consists of a root node, interior nodes, and leaf nodes. An interior node points to child nodes. It contains the hash table structure where the hash table entries store the pointer to the child nodes. The leaf node is where the candidate sequences resided. It uses the list data structure to store the candidate sequence.
To update the support count of the candidate sequences, we traverse from the root node until reached the leaf node. We reduce all the possible traverse paths by hashing on the correct items in a data sequence. By doing so, we reduce the number of candidate sequences that needs to be count. Thus saving time of candidate counts.

**Question 10:**

Give the definitions of Maximum gap, Minimum gap, and Window size in the Generalized Sequential Patterns algorithm.

**Answer:**

Maximum gap - the maximum allowed time difference between the latest occurrence of an event in an event-set and the earliest occurrence of an event in its immediately preceding event-set.

Minimum gap - the minimum required time difference between the earliest occurrence of an event in an event-set and the latest occurrence of an event in its immediately proceeding event-set.

Window size – The maximum allowed time difference between the latest and earliest occurrences of events in any event set.
Classification

Question 1:

Give two attribute selection criteria for the decision tree construction.

Answer:

Gain criterion, and Gini index.

Gain criterion find the best split point by measuring the information gain using the entropy measure, whereas, the Gini index measures the purity of the data after split.

Question 2:

Describe the variation of serial SPRINT algorithm from the C4.5 algorithm.

Answer:

*Continuous attributes*
In C4.5, the continuous attributes are sorted in each node to find the best split point. This results in high computational overhead when the number of training data is large and many continuous attribute in the training data set.

On the other hand, SPRINT algorithm performs continuous attributes sorting at the root node. The attribute lists entries consist of records with element, record id, target class, and value. The record ID plays an importance role to keep track the complete attribute values of a training example. When splitting continuous attributes, the child node still preserve the sorted order of the continuous attribute. Consequently, the continuous attribute didn’t needs to be re-sorted at each child node.

*Multi-ways Vs Two-ways branching*
C4.5 algorithm uses multi-way branching. That means, if the split discrete attribute consisted of n attribute values, n child nodes will be created that split on one attribute value.

SPRINT algorithm only support binary branching. For discrete attributes, two subsets of the attribute values are created, where each branch split on the first subset, and second subset for the other branch. Both algorithms perform binary split on the continuous attribute.

Question 3:

What is the main data structure used in the SPRINT algorithm?

There are three main data structures in SPRINT algorithm: the attribute lists, count matrix and histograms. The count matrix is an n × c matrix where the rows are the attribute values for a discrete attribute, and c is the total number of target classes. The i-th row and j-th column represent the frequency of training examples that contain attribute value of the i-th row and j-th class. This class frequency information is required to calculate the Gini index value for a discrete attribute.
The histograms data structure consist of Cabove and Cbelow. Each structure is a matrix of size $1 \times c$. The Cbelow matrix is used to indicate how many training examples in a node that is less or equal than a cut point value for a continuous attribute. The columns are class frequencies of examples that have continuous attribute less than the cut point value.

On the contrary, the Cabove matrix stores the class frequencies of training example that has the continuous attribute values greater than the cut point value.

In conclusion, the histograms and the count matrix are used to calculate the Gini index value for the candidate split attribute.

**Question 4:**

Beside the synchronous tree construction, what is other possible parallel tree construction architecture?

**Answer:**

Another way to parallel construct decision tree is through a tree-partitioned technique. In this approach, different processors work on different parts of the classification tree. In particular, if more than one processor cooperates to expand a node, then these processors are partitioned to expand the successors of this node.

**Question 5:**

How the parallel SPRINT algorithm differs from serial SPRINT in treating Discrete and Continuous attributes?

**Answer:**

*Discrete attribute*

In each processor, the local tree nodes have only the local class distribution information. In order to find the best split subset of attribute values the local processor needs to global class distribution. Recalled that the class distribution for discrete attribute is stored in a count matrix structure. The find the split, each processor will gather all the count matrix entries from all nodes to find the global class distribution. Then each processor can continue independently on calculating the best split subset.

*Continuous attribute*

Before find the best split point for an continuous attribute, we needs to initialize the Cabove and Cbelow histograms with the class distribution information from other processor. The Cbelow histogram of processor rank $i$ is initialized with the Cabove histograms from processor of lower rank $j$ ($j < i$), whereas the Cbelow histogram is initialized with Cabove histogram from processor of higher rank $j$ ($j > i$). By doing so, the local histograms of processor will reflect the other part of the data. Each processor can then independently find the cut point in local partition. After evaluating all the cut-points in local partition, the processors perform all exchanged to exchange their local cut points and their Gini index values. After a processor collected the completed set of cut points, they can independently find the best cut point for a continuous attribute.
**Question 6:**

What are some of the difficulties or tricks in the implementations of the parallel SPRINT algorithms? Give examples?

**Answer:**

The MPI uses the Single Instruction Multiple Program (SIMP) paradigm to perform parallelism. In the coding, the if-then-else statement is used to switch between the executions of different statements (if necessary) in different processor. We must ensure each processor will eventually reach the same execution point either follow the same or different execution path. Otherwise, for example all processor execution the MPI_Allgather() statement except one, then the whole program will be blocked. Every possible path of the program must be considered and to avoid any error.

Since, the parallel SPRINT algorithm performs parallel sorting at the root node, it is rather hard to ensure balancing partitioning on the continuous attributes records. On the worst case, one processor may get zero record. Thus, the trick is to think “globally” during implementation and we must consider other parts of data that in the same node in other processor. We must always use the global version of class distribution information to find the best split attribute.

**Question 7:**

What is the greedy algorithm, and suggest how to implements it?

**Answer:**

The greedy algorithm is an incremental algorithm that adds one object to a subset from another subset that is initially the complete subset of that object. This algorithm has been applied to the feature selection algorithm, in which one feature in discarded one at a time and evaluate the performance on the remaining. In SPRINT algorithm, since only binary split was considered, we need to create two subset from all attribute values of a discrete attribute. The first subset is initially empty, and the second subset consists of all attribute values. One attribute from the second subset is added to the first subset and evaluated their performance. This process is repeated until there is no improvement on the performance.

One way to implements the greedy algorithm, we can create two indices array from subset 1 and subset 2. The size of each array is total number of attribute values of the discrete attribute. The entries on the indices array are the index of the discrete attribute values. Let the attribute values are indexed as 1,...,V. Initially, indices1 = φ, indices2 = {1,...,V}. The pseudo code is as follow,

```plaintext
do
  evaluate the performance of subsets in indices1 and indices2
  if current performance is less than previous iteration performance
    stop
  draw one index from indices2 and add to indices1 array
while(indices2 is not empty)
  use subsets in indices1 and indices2 as the attribute values subset
```

EDPNMO006/2001 (D.Taniar & K.Smith) – page 12
Question 8:

What is the main drawback of the SPRINT algorithm? Give examples to support your comment?

Answer:

From the implementation viewpoint, the main drawback is the imbalance of the decision tree in difference processor. In one processor, there may be 10 examples in a node, while it is 0 example at the same node in other processor. Thus, it is wasted of resource while we still split the nodes that have empty examples. The nodes will be most of the time waiting for other node in other processor to complete local operations. The algorithm also does not scale well when the tree are deep, this is due to high synchronization cost at lower level of the tree.

Question 9:

How do we use the MPI functions to exchange hash table entries?

Answer:

The hash table data structure is used to store the branching information of the records in the split attribute list. Due to parallel sorting on continuous attribute, the continuous and discrete attributes records for the same training example will not partition to the same processor. When performing splitting, one attribute record will not no which child it should goes because the split attribute record for the same training example is in other processor. This processor must get this information from the other processor on where the assign this attribute record.

The main disadvantage of the MPI functions is the send/receive buffer of array type. Thus there is no direct way we can exchange the hash table entries. The first step is to save the hash table entries into an array and perform all gather.

There are few MPI functions that need to be used for exchanging hash table, that is the MPI_Allgather(), MPI_Allgatherv(). The pseudocode is as follow:

```c
HshCell * sendbuf, *recvbuf;
int *recvcount;
int size, totsize;
...

size = Copy_entries(Hashtab, sendbuf);

MPI_Allgather(&size, 1, MPI_INT, recvcount, MPI_INT,1, comm);
totsize =0
for all processor indexes i
    totsize += recvcount[i]
allocate memory of size totsize of type HshCell to recvbuf

Find_recv_displacements(recvcounts)
```
MPI_Allgatherv(sendbuf, size, hash_type, recvbuf, recv_disp, hash_type, comm);

    destroy old hash table entries
Copy the recvbuf entries into hash table entries

As an exercise, implements your own algorithm.

**Question 10:**

From your opinion, why have decision tree type algorithms gained wide popularity as compared to other algorithm like neural network?

**Answer:**

Neural network uses connectionism approach to perform function approximation of the underlying training data. The connection weights between difference layers are updated using the gradient descent training approach.

There are several advantages of decision tree as compared to neural network,

1. The decision rules produced by the decision tree are more comprehensible as compared to neural network weights. Although there are few attempts to extract rules from neural network, but most algorithm still lacks of practical usefulness to large data set and is not direct.
2. It is rather hard to determine the optimal parameters in the neural network training, for example number of hidden units in the hidden layer for multi-layered network. In decision tree, the tree structure is generated automatically from the algorithm.
3. Neural network needs long training time and a few training parameters like momentum, learning rate that needs trial and error. Some of the network like Radial Basis Function network has difficulty to determine the optimal number of class centers/hiden units. Decision tree has less tuning parameters, for example the minimum examples in a node and the percentage of most frequent class in a leaf node.

Neural network normally performs better than the decision tree in terms of classification rate, but it is not significant if you view from the scalability and practical usefulness. There were also attempts to parallelize neural network training.
Clustering

**Question 1:**

What is clustering?

**Answer:**

Clustering distributes data into several groups so that similar objects fall into same group. The objective is to create clusters where the intra-dissimilarity is minimized and inter-dissimilarity is maximized.

**Question 2:**

Give the possible ways to initialize the cluster centers in the k-means algorithm.

- Random subset selection, or
- Assigning the i-th partition to the i-th cluster, or
- Supplied by the user or back-ground knowledge

**Question 3:**

Describe how parallel k-means distributed memory algorithm works.

**Answer:**

The parallel k-means algorithm employs the data distribution paradigm, in which each processor has 1/P of the data partition. Each processor initially initializes their local clusters center using their local data. They exchange all the cluster means values to calculate the global cluster center. The clustering algorithm is as follow:

```plaintext
while not reached maximum iteration and clusters not stable
    for all data points in local partition
        assign the data point to the nearest cluster
        update the number of points and means of the cluster center
    end for
    exchange all clusters center information
    update clusters center with the global cluster means
    determine the performance of the clusters
end while
```
**Question 4:**

Give several quality measurements on the clusters created by clustering algorithm.

Answer:

- Mean Squared Error in each cluster, or
- confusion matrix, or
- entropy measure

**Question 5:**

Explain the serial k-means algorithm.

Answer:

The k-means algorithm is an iterative algorithm that iteratively assigns and reassigns the training data to the nearest cluster. The nearest cluster is measured using some distance measure in the feature space such as the well-known Euclidean distance. At each iteration, the clusters mean is recalculated. The total number of cluster is fixed by the user. The k-means algorithm is outlined below,

Let d is the data dimension, m is the cluster means,

1. **Initialization**
   
   Select a set of k starting points \( \{ m_j \}_{j=1}^k \) in \( \mathbb{R}^d \). The selection may be done in a random manner or according to some heuristic.

2. **Distance Calculation**
   
   For each data point \( X_i, 1 \leq i \leq n \), compute its Euclidean distance to each cluster mean \( m_j, 1 \leq j \leq k \), and then find the closest cluster centroid.

3. **Mean recalculation**
   
   For each \( 1 \leq j \leq k \), recompute cluster mean \( m_j \) as the average of data points assigned to it.

4. **Convergence condition**
   
   Repeat steps 2 and 3 until convergence

**Question 6:**

What are some of the applications of clustering?

Answer:

- Image segmentation in computer vision.
- Documents clustering or information retrieval
- Object and character recognition
Question 7:
What are the components of clustering task?

Answer:
1. pattern representation (optional),
2. definition of a pattern proximity measure appropriate to the data domain,
3. clustering or grouping,
4. data abstraction (optional), and
5. assessment of output (optional)

Question 8:
Give few drawbacks of the k-means algorithm.

Answer:
1. Sensitive to initial cluster center – difference cluster center initialization method will result in difference final clusters.
2. Outlier deletion – the k-means algorithm assigns the data point to the closest cluster center. It doesn’t take into consideration whether this data point is an outlier (too far from the closest cluster) or not.
3. Case order dependence – the k-means algorithm updates the cluster center when there is changes in the data points. Thus, it is sensitive the ordering of training data present to the clustering algorithm.
4. Randomized trials – needs trial and error methods to find the best clusters or number of clusters.

Question 9:
K-means clustering algorithm creates a sphere boundaries for each cluster. What are the other types of boundaries shape that is possible?

Answer:
- Hyperbox in Fuzzy clustering which create boundaries using the min and max points.
- Hyper-Ellipsoidal boundaries using Expectation Maximization clustering algorithm
- Arbitrary boundaries shape

Question 10:
What are some of the requirements of the partitioning clustering method?

Answer:
- Each group/cluster must contain at least one object,
- Each object must belong to exactly on group.

These conditions imply that there are at most as many clusters as there are objects.