1) Decision Tree :

ALGORITHM 3.2

```
Input:
   T     //Decision tree
   D     //Input database
Output:
   M     //Model prediction
DTProc algorithm:
             //Simplistic algorithm to illustrate prediction
               technique using DT
   for each t ∈ D do
      n = root node of T;
      while n not leaf node do
         Obtain answer to question on n applied to t;
         Identify arc from t, which contains correct answer;
         n = node at end of this arc;
      Make prediction for t based on labeling of n;
```

2) Naïve Bayes :

```
Input:
T // Decision tree
D // Input database

Output:
M // Model prediction

NaiveBayesProc Algorithm:
// Simplified Naive Bayes prediction algorithm

// Step 1: Calculate Class and Feature Probabilities
CalculateProbabilities(D);

// Step 2: Make Predictions
for each data point t in D do
    CalculateClassProbabilities(t);
    M[t] = Argmax(P(Y | X)) // Select the class with the highest probab

// Output the final predictions
Output M
```

3) K-Means :

ALGORITHM 5.6

```
Input:
   D = {t_1, t_2, ..., t_n}    //Set of elements
   k      //Number of desired clusters
Output:
   K      //Set of clusters
K-means algorithm:
   assign initial values for means m_1, m_2, ..., m_k;
   repeat
      assign each item t_i to the cluster which has the closest mean;
      calculate new mean for each cluster;
   until convergence criteria is met;
```

The K-means algorithm is illustrated in Example 5.4.

4) K-Mediods(PAM) :

ALGORITHM 5.8

```
Input:
   D = {t_1, t_2, ..., t_n}    //Set of elements
   A      //Adjacency matrix showing distance between elements
   k      //Number of desired clusters
Output:
   K      //Set of clusters
PAM algorithm:
   arbitrarily select k medoids from D;
   repeat
      for each t_h not a medoid do
         for each medoid t_i do
            calculate TC_ih;
      find i, h where TC_ih is the smallest;
      if TC_ih < 0, then
         replace medoid t_i with t_h;
   until TC_ih ≥ 0;
   for each t_i ∈ D do
      assign t_i to K_j, where dis(t_i, t_j) is the smallest over all medoids;
```

5) Hierarchical Clustering – Single :

```
ALGORITHM 5.2
Input:
   D = {t1, t2, ..., tn}    //Set of elements
   A     //Adjacency matrix showing distance between elements
Output:
   DE    // Dendrogram represented as a set of ordered triples
MST single link algorithm:
   d = 0
   k = n
   K = {{t1}, ..., {tn}}
   DE = ⟨d, k, K⟩; // Initially dendrogram contains each element in
         its own cluster.
   M = MST(A);
   repeat
      oldk = k;
      Ki, Kj = two clusters closest together in MST;
      K = K − {Ki} − {Kj} ∪ {Ki ∪ Kj};
      k = oldk − 1;
      d = dis(Ki, Kj);
      DE = DE ∪ ⟨d, k, K⟩; // New set of clusters added to dendrogram.
      dis(Ki, Kj) = ∞
   until k = 1
```

6) Hierarchical Clustering – Average :

```
ALGORITHM 5.3
Input:
   D = {t1, t2, ..., tn}    //Set of elements
   A     //Adjacency matrix showing distance between elements
Output:
   DE    // Dendrogram represented as a set of ordered triples
Average link algorithm:
   d = 0;
   k = n;
   K = {{t1}, ..., {tn}};
   DE = ⟨d, k, K⟩; // Initially dendrogram contains each element
                   in its own cluster.
   repeat
      oldk = k;
      d = d + 0.5;
      for each pair of Ki, Kj ∈ K do
         ave = average distance between all ti ∈ Ki and tj ∈ Kj;
         if ave ≤ d, then
            K = K − {Ki} − {Kj} ∪ {Ki ∪ Kj};
            k = oldk − 1;
            DE = DE ∪ ⟨d, k, K⟩; // New set of clusters added
                                    to dendrogram.
      until k = 1
```

7) Hierarchical Clustering – Complete :

```
Input:
D = {t1, t2, ..., tn} // Set of elements
A // Adjacency matrix showing distance between elements

Output:
DE // Dendrogram represented as a set of ordered triples

Complete Linkage Hierarchical Clustering Algorithm:
d = 0
k = n
K = {{t1}, ..., {tn}}
DE = (d, k, K); // Initially, dendrogram contains each element in its o

// Calculate the complete-linkage distance matrix
D_complete = CalculateCompleteLinkageDistanceMatrix(A)

repeat
    oldk = k

    // Find two clusters with the smallest complete-linkage distance
    (Ki, Kj) = FindClustersWithMinCompleteLinkageDistance(D_complete)

    // Merge the two clusters
    K = K - {Ki} - {Kj} U {Ki U Kj}
    k = oldk - 1
    d = dis(Ki, Kj)

    // Update the dendrogram
    DE = DE U (d, k, K)

    // Set the distance between merged clusters to infinity to avoid re
    dis(Ki, Kj) = ∞

until k = 1
```

8) Apriori :

ALGORITHM 6.3

```
Input:
    I       //Itemsets
    D       //Database of transactions
    s       //Support
Output:
    L       //Large itemsets
Apriori algorithm:
    k = 0;  //k is used as the scan number.
    L = ∅;
```

```
C_1 = I;        //Initial candidates are set to be the items.
repeat
   k = k+1;
   L_k = Ø;
   for each I_i ∈ C_k do
      c_i = 0;    // Initial counts for each itemset are 0.
   for each t_j ∈ D do
      for each I_i ∈ C_K do
         if I_i ∈ t_j then
            c_i = c_i +1;
   for each I_i ∈ C_k do
      if c_i ≥ (s× | D |) do
         L_k = L_k ∪ I_i;
   L = L ∪ L_k;
   C_{k+1} = Apriori-Gen(L_k)
until C_{k+1} = Ø;
```

9) FP-Tree

Algorithm: **FP_growth.** Mine frequent itemsets using an FP-tree by pattern fragment growth.

Input:

- D, a transaction database;
- *min_sup*, the minimum support count threshold.

Output: The complete set of frequent patterns.

Method:

1. The FP-tree is constructed in the following steps:

   (a) Scan the transaction database D once. Collect F, the set of frequent items, and their support counts. Sort F in support count descending order as L, the *list* of frequent items.

   (b) Create the root of an FP-tree, and label it as "null." For each transaction *Trans* in D do the following. Select and sort the frequent items in *Trans* according to the order of L. Let the sorted frequent item list in *Trans* be $[p|P]$, where p is the first element and P is the remaining list. Call insert_tree($[p|P]$, T), which is performed as follows. If T has a child N such that $N.item\text{-}name = p.item\text{-}name$, then increment N's count by 1; else create a new node N, and let its count be 1, its parent link be linked to T, and its node-link to the nodes with the same *item-name* via the node-link structure. If P is nonempty, call insert_tree(P, N) recursively.

2. The FP-tree is mined by calling FP_growth(*FP_tree*, *null*), which is implemented as follows.

procedure FP_growth(*Tree*, α)
(1) **if** *Tree* contains a single path P **then**
(2) **for each** combination (denoted as β) of the nodes in the path P
(3) generate pattern $\beta \cup \alpha$ with $support\_count = minimum\ support\ count\ of\ nodes\ in\ \beta$;
(4) **else for each** a_i in the header of *Tree* {
(5) generate pattern $\beta = a_i \cup \alpha$ with $support\_count = a_i.support\_count$;
(6) construct β's conditional pattern base and then β's conditional FP_tree $Tree_\beta$;
(7) **if** $Tree_\beta \neq \emptyset$ **then**
(8) call FP_growth($Tree_\beta$, β); }

10) Page Rank :

Input:

W // Web pages and their links represented as a directed graph

q // Query page

Output:

A, H // Sets of authority and hub pages

PageRank Algorithm:

// Simplified PageRank algorithm

// Step 1: Initialize PageRank Scores

InitializePageRankScores(W);

// Step 2: Perform Iterative PageRank Calculation

```
for i = 1 to max_iterations do
   for each page p in W do
      PR(p) = (1 - d) + d * Σ (PR(q) / L(q)) for all pages q linking to p
```

// Step 3: Identify Authority and Hub Pages

A = FindTopPagesByPageRank(PR, k); // Select top k pages based on PageRank scores as authority pages

H = FindTopPagesByPageRank(1 / PR, k); // Select top k pages based on the inverse of PageRank scores as hub pages

// Output the final sets of authority and hub pages

Output A, H

11) HITS

ALGORITHM 7.1

Input:

```
   W         //WWW viewed as a directed graph
   q         //Query
   s         //Support
```

Output:

```
   A         //Set of authority pages
   H         //Set of hub pages
```

HITS algorithm

$R = SE(W, q)$

$B = R \cup$ {pages linked to from R} $\cup$ {pages that link to pages in R};

$G(B, L) =$ Subgraph of W induced by B;

$G(B, L^1) =$ Delete links in G within same site;

$x_p = \sum_q$ where $(q,p) \in L^1$ y_q; // Find authority weights;

$y_p = \sum_q$ where $(p,q) \in L^1$ x_q; // Find hub weights;

$A = \{p \mid p$ has one of the highest $x_p\}$;

$H = \{p \mid p$ has one of the highest $y_p\}$;

For Information package and Designing schema refer Exp. 2

For OLAP operations refer Exp. 7