1. Classification
    (a) K-Nearest Neighbor
2. Ensemble Algorithms
    (a) Adaboost

Algorithm 1 k-Nearest Neighbor

Input: X: training data, Y:Class labels of X, x: unknown sample
Output: Class with the highest number of occurrence

function Classify(X, Y, x)
    for $i = 1$ to m **do**
        Compute distance $d(X_i, x)$
    end for
    Compute set I containing indices for the k smallest distances $d(X_i, x)$
    Return majority label $\{Y_i$ where $i \in I\}$
end function

Algorithm 2 Adaboost

Input:
Training data $\{(x_i, y_i)_{i=1}^{N}$ where $x_i \in \mathbb{R}^k$ and $y_i \in \{-1, 1\}\}$
Large number of classifiers denoted by $f_m(x) \in \{-1, 1\}$
0-1 loss function I defined as

$$I(f_m(x, y)) = \begin{cases} 0, & \text{if } f_m(x_i) = y_i & (1) \\ 1, & \text{if } f_m(x_i) \neq y_i & (2) \end{cases}$$

Output: The final classifier

for $i = 1$ to N **do**
    for $i = 1$ to M **do**
        Fit weak classifier m to minimize the objective function:
        $\epsilon_m = \frac{\sum_{i=1}^{N} w_i^m I(f_m(x_i)) \neq y_i}{x^2+2x+1}$
        where $I(f_m(x_i) \neq y_i) = 1$ if $f_m(x_i) \neq y_i$ and 0 otherwise
        $\alpha_m = \ln \frac{1-\epsilon_m}{\epsilon_m}$
    end for
    for all i **do**
        $w_i^{m+1} = w_i^{(m)} e^{\alpha_m I(f_m(x_i) \neq y_i)}$
    end for
end for

Algorithm 3 Adaboost

Input:

Training data $\{(x_i, y_i)_{i=1}^N$ where $x_i \in \mathbb{R}^k$ and $y_i \in \{-1, 1\}\}$

Output: The final classifier

Given Training data $\{(x_i, y_i)$ where $y_i \in \{-1, 1\}\}$
initialize D_1 = uniform distribution on training examples
for $t = 1$ to T **do**
  Train weak classifier h_t on D_t
  choose $\alpha_t > 0$
  compute new distribution D_{t+1}:
  for all i **do**
    multiply $D_t(x)$ by

$$\begin{cases} e^{-\alpha_t}, & (< 1) \text{ if } y_i = h_t(x_i) \quad (3) \\ e^{\alpha_t}, & (> 1) \text{ if } y_i \neq h_t(x_i) \quad (4) \end{cases}$$

    renormalize
  end for
  output final classifier $H_f inal(x) = sign(\sum \alpha_t h_t(x))$
end for

Algorithm 4 Random forest

Input: S: training set, F:Features and number of trees in forest B

Output: Constructed tree

function RANDOMFOREST(S, F)
  $H \leftarrow \emptyset$
  for $i \in 1, ...B$ **do**
    $S^{(i)} \leftarrow$ A bootstrap sample from S
    $h_i \leftarrow RANDOMIZEDTREELEARN(S^i, F)$
    $H \leftarrow H \bigcup \{h_i\}$
  end for
  return H
end function
function RANDOMIZEDTREELEARN(S, F)
  At each node:
  $f \leftarrow$ a very small subset of F
  Split on best feature in f
  return The learned tree
end function

Algorithm 5 Iterative Dichotomiser 3

Input: D : Training Data, X : Set of Input Attributes
Output: A decision tree

function ID3(D, X)
  Let T be a new tree
  if all instances in D have the same class c **then**
    Label (T) = c; Return T
  end if
  if $X = \emptyset$ or no attribute has positive information gain **then**
    Label (T) = most common class in D; Return T
  end if
  $X \leftarrow$ attribute with highest information gain
  Label(T) = X
  for each value x of X **do**
    $D_x \leftarrow$ instances in D with $X = x$
    if D_x is empty **then**
      Let T_x be a new tree
      Label(T_x) = most common class in D
    else
      T_x = ID3($D_x, X - \{x\}$)
  end if
  Add a branch from T to T_x labeled by x
end for
return T
end function

Algorithm 6 Perceptron

Input: $ProblemSize, InputPatterns, iterations_max, learn_rate$
Output: $Weights$

for $i = 1$ to $iterations_{max}$ **do**
  $Pattern_i \leftarrow SelectInputPattern(InputPatterns)$
  $Activation_i \leftarrow ActivateNetwork(Pattern_i, Weights)$
  $Output_i \leftarrow TransferActivation(Activation_i)$
  $UpdateWeights(Pattern_i, Output_i, learn_{rate})$
end for
Return $Weights$

Algorithm 7 Back-propagation

Input: $ProblemSize, InputPatterns, iterations_max, learn_rate$
Output: $Network$

$Network \leftarrow ConstructNetworkLayers()$
$Network_weights \leftarrow InitializeWeights(Network, ProblemSize)$
for $i = 1$ to $iterations_{max}$ **do**
  $Pattern_i \leftarrow SelectInputPattern(InputPatterns)$
  $Output_i \leftarrow ForwardPropagate(Pattern_i, Network)$
  $BackwardPropagateError(Pattern_i, Output_i, Network)$
  $UpdateWeights(Pattern_i, Output_i, Network, learn_{rate})$
end for
Return $Network$

Algorithm 8 Learning Vector Quantization

Input: $ProblemSize, InputPatterns, iterations_{max}, CodebookVectors_{num}, learn_{rate}$

Output: $CodebookVectors$

$CodebookVectors \leftarrow InitializeCodebookVectors(CodebookVectors_{num}, ProblemSize)$
for $i = 1$ to $iterations_{max}$ **do**
  $Pattern_i \leftarrow SelectInputPattern(InputPatterns)$
  $Bmu_i \leftarrow SelectBestMatchingUnit(Pattern_i, CodebookVectors)$
  for $Bmu_i^{attribute} \in Bmu_i$ **do**
    if $Bmu_i^{class} \equiv Pattern_i^{class}$ **then**
      $Bmu_i^{attribute} \leftarrow Bmu_i^{attribute} + learn_{rate} \times (Pattern_i^{attribute} - Bmu_i^{attribute})$
    else
      $Bmu_i^{attribute} \leftarrow Bmu_i^{attribute} - learn_{rate} \times (Pattern_i^{attribute} - Bmu_i^{attribute})$
  end if
end for
end for
Return $CodebookVectors$

Algorithm 9 Self Organizing Map

Input: $InputPatterns, iterations_{max}, learn_{rate}, Grid_width, Grid_height$

Output: $CodebookVectors$

$CodebookVectors \leftarrow InitializeCodebookVectors(Grid_{width}, Grid_{height}, InputPatterns)$
for $i = 1$ to $iterations_{max}$ **do**
  $Learn_{rate}^{i} \leftarrow CalculateLearningRate(i, learn_{rate}^{init})$
  $neighborhood_{size}^{i} \leftarrow CalculateNeighborhoodSize(i, neighborhood_{init}^{size})$
  $Pattern_i \leftarrow SelectInputPattern(InputPatterns)$
  $Bmu_i \leftarrow SelectBestMatchingUnit(Pattern_i, CodebookVectors)$
  $Neighborhood \leftarrow Bmu_i$
  $Neighborhood \leftarrow SelectNeighbors(Bmu_i, CodebookVectors, neighborhood_{size}^{i})$
  for $Vector_i \in Neighborhood$ **do**
    for $Vector_i^{attribute} \in Vector_i$ **do**
      $Vector_i^{attribute} \leftarrow Vector_i^{attribute} + learn_{rate} \times (Pattern_i^{attribute} - Vector_i^{attribute})$
    end for
  end for
end for
Return $CodebookVectors$

Algorithm 10 Hierarchial Agglomerative Algorithm

Input:
  $\langle V, E, w \rangle$.Weighted graph
  d_c.Distance measure for two clusters

Output: $\langle V_T, E_T \rangle$.Cluster hierarchy or dendogram

$C = \{\{v \mid v \in V\}\}$ ▷ Initial Clustering
$V_t = \{v_C \mid C \in C\}, E_T = \emptyset$ ▷ Initial Dendogram
while $|C| > 1$ **do**
  $update\_distance\_matrix(C, G, d_c)$
  $\{C, C'\} = \underset{\{C_i, C_j\} \in C : C_i \neq C_j}{argmin} d_c(C_i, C_j)$
  $C = (C \backslash \{C, C'\}) \cup \{C \cup C'\}$ ▷ Merging
  $V_T = V_T \cup \{v_{C,C'}\}, E_T = E_T \cup \{\{v_{C,C'}, v_C\}, \{v_{C,C'}, v_C\}\}$ ▷ Dendogram
end while
Return T

Algorithm 11 Hierarchial Divisive Algorithm

Input:

$\langle V, E, w\rangle$.Weighted graph

d_c.Distance measure for two clusters

Output: $\langle V_T, E_T\rangle$.Cluster hierarchy or dendogram

$C = \{V\}$ ▷ Initial Clustering

$V_t = \{v_C \mid C \in C\}, E_T = \emptyset$ ▷ Initial Dendogram

while $\exists C_x : (C_x \in C \wedge |C| > 1)$ **do**

  $update\_distance\_matrix(C, G, d_c)$

  $\{C, C'\} = \underset{\{C_i, C_j\}: C_i \cup C_j = C_x \wedge \ C_i \cap C_j = \emptyset}{argmax} d_c(C_i, C_j)$

  $C = (C \backslash \{C, C'\}) \cup \{C \cup C'\}$ ▷ Merging

  $V_T = V_T \cup \{v_{C,C'}\}, E_T = E_T \cup \{\{v_{C,C'}, v_C\}, \{v_{C,C'}, v_C\}\}$ ▷ Dendogram

end while

Return T

Algorithm 12 The Apiori Algorithm

Input: T : transaction datasets, ϵ : support threshold

Output: Frequent itemsets

while $L_k \neq \emptyset$ **do**

  for $i = 1$ to m **do**

    Compute distance $d(X_i, x)$

  end for

  Compute set I containing indices for the k smallest distances $d(X_i, x)$

  Return majority label $\{Y_i$ where $i \in I\}$

end while