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
1: **function** CLASSIFY(X, Y, x)
2:     **for** $i = 1$ to m **do**
3:         Compute distance $d(X_i, x)$
4:     **end for**
5:     Compute set I containing indices for the k smallest distances $d(X_i, x)$
6:     Return majority label $\{Y_i \text{ where } i \in I\}$
7: **end function**

---

1. Ensemble Algorithm

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

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

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

---

1. Another Pseudocode for Adaboost

1. Machine Learning Algorithms Pseudocode

1. Machine Learning Algorithms Pseudocode

1. Machine Learning Algorithms Pseudocode

1. Machine Learning Algorithms Pseudocode

1. Machine Learning Algorithms Pseudocode

1. Machine Learning Algorithms Pseudocode

1. Machine Learning Algorithms Pseudocode

1. Machine Learning Algorithms Pseudocode

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

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

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

9:         renormalize
10:     **end for**
11:     output final classifier $H_f inal(x) = sign(\sum \alpha_t h_t(x))$
12: **end for**

Algorithm 4 Random forest

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

Output: Constructed tree

1: **function** RANDOMFOREST(S, F)
2:     $H \leftarrow \emptyset$
3:     **for** $i \in 1, ...B$ **do**
4:         $S^{(i)} \leftarrow$ A bootstrap sample from S
5:         $h_i \leftarrow RANDOMIZEDTREELEARN(S^i, F)$
6:         $H \leftarrow H \bigcup \{h_i\}$
7:     **end for**
8:     return H
9: **end function**
10: **function** RANDOMIZEDTREELEARN(S, F)
11:     At each node:
12:     $f \leftarrow$ a very small subset of F
13:     Split on best feature in f
14:     return The learned tree
15: **end function**

Algorithm 5 Iterative Dichotomiser 3

Input: D : Training Data, X : Set of Input Attributes
Output: A decision tree
 1: **function** ID3(D, X)
 2:    Let T be a new tree
 3:    **if** all instances in D have the same class c **then**
 4:        Label $(T) = c$; Return T
 5:    **end if**
 6:    **if** $X = \emptyset$ or no attribute has positive information gain **then**
 7:        Label (T) = most common class in D; Return T
 8:    **end if**
 9:    $X \leftarrow$ attribute with highest information gain
10:    Label$(T) = X$
11:    **for** each value x of X **do**
12:        $D_x \leftarrow$ instances in D with $X = x$
13:        **if** D_x is empty **then**
14:            Let T_x be a new tree
15:            Label(T_x) = most common class in D
16:        **else**
17:            $T_x = $ ID3($D_x, X - \{x\}$)
18:        **end if**
19:    Add a branch from T to T_x labeled by x
20: **end for**
21: return T
22: **end function**

Algorithm 6 Perceptron

Input: $ProblemSize, InputPatterns, iterations_max, learn_rate$
Output: $Weights$
 1: **for** $i = 1$ to $iterations_{max}$ **do**
 2:    $Pattern_i \leftarrow SelectInputPattern(InputPatterns)$
 3:    $Activation_i \leftarrow ActivateNetwork(Pattern_i, Weights)$
 4:    $Output_i \leftarrow TransferActivation(Activation_i)$
 5:    $UpdateWeights(Pattern_i, Output_i, learn_{rate})$
 6: **end for**
 7: Return $Weights$

Algorithm 7 Back-propagation

Input: $ProblemSize, InputPatterns, iterations_max, learn_rate$
Output: $Network$
 1: $Network \leftarrow ConstructNetworkLayers()$
 2: $Network_weights \leftarrow InitializeWeights(Network, ProblemSize)$
 3: **for** $i = 1$ to $iterations_{max}$ **do**
 4:    $Pattern_i \leftarrow SelectInputPattern(InputPatterns)$
 5:    $Output_i \leftarrow ForwardPropagate(Pattern_i, Network)$
 6:    $BackwardPropagateError(Pattern_i, Output_i, Network)$
 7:    $UpdateWeights(Pattern_i, Output_i, Network, learn_{rate})$
 8: **end for**
 9: Return $Network$

Algorithm 8 Learning Vector Quantization

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

Output: $CodebookVectors$

1: $CodebookVectors \leftarrow InitializeCodebookVectors(CodebookVectors_{num}, ProblemSize)$
2: **for** $i = 1$ to $iterations_{max}$ **do**
3: $\quad Pattern_i \leftarrow SelectInputPattern(InputPatterns)$
4: $\quad Bmu_i \leftarrow SelectBestMatchingUnit(Pattern_i, CodebookVectors)$
5: $\quad$ **for** $Bmu_i^{attribute} \in Bmu_i$ **do**
6: $\quad\quad$ **if** $Bmu_i^{class} \equiv Pattern_i^{class}$ **then**
7: $\quad\quad\quad Bmu_i^{attribute} \leftarrow Bmu_i^{attribute} + learn_{rate} \times (Pattern_i^{attribute} - Bmu_i^{attribute})$
8: $\quad\quad$ **else**
9: $\quad\quad\quad Bmu_i^{attribute} \leftarrow Bmu_i^{attribute} - learn_{rate} \times (Pattern_i^{attribute} - Bmu_i^{attribute})$
10: $\quad\quad$ **end if**
11: $\quad$ **end for**
end for
Return $CodebookVectors$

Algorithm 9 Self Organizing Map

Input: $InputPatterns, iterations_{max}, learn_{rate}, Grid_{width}, Grid_{height}$

Output: $CodebookVectors$

1: $CodebookVectors \leftarrow InitializeCodebookVectors(Grid_{width}, Grid_{height}, InputPatterns)$
2: **for** $i = 1$ to $iterations_{max}$ **do**
3: $\quad Learn_{rate}^i \leftarrow CalculateLearningRate(i, learn_{rate}^{init})$
4: $\quad neighborhood_{size}^i \leftarrow CalculateNeighborhoodSize(i, neighborhood_{init}^{size})$
5: $\quad Pattern_i \leftarrow SelectInputPattern(InputPatterns)$
6: $\quad Bmu_i \leftarrow SelectBestMatchingUnit(Pattern_i, CodebookVectors)$
7: $\quad Neighborhood \leftarrow Bmu_i$
8: $\quad Neighborhood \leftarrow SelectNeighbors(Bmu_i, CodebookVectors, neighborhood_{size}^i)$
9: $\quad$ **for** $Vector_i \in Neighborhood$ **do**
10: $\quad\quad$ **for** $Vector_i^{attribute} \in Vector_i$ **do**
11: $\quad\quad\quad Vector_i^{attribute} \leftarrow Vector_i^{attribute} + learn_{rate} \times (Pattern_i^{attribute} - Vector_i^{attribute})$
12: $\quad\quad$ **end for**
13: $\quad$ **end for**
14: **end for**
15: Return $CodebookVectors$

Algorithm 10 Hierarchial Agglomerative Algorithm

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

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

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

4

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

1:   $C = \{V\}$ ▷ Initial Clustering

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

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

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

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

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

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

8:   **end while**

9:   Return T