

1. Classification
  - (a) K-Nearest Neighbor
  - (b) Adaboost
  - (c) Iterative Dichotomiser 3
  - (d) C4.5
  - (e) Naive Bayes
  - (f) Bagging
  - (g) Random Forest
2. Neural Networks
  - (a) Perceptron
  - (b) Back-Propagation
  - (c) Learning Vector Quantization
  - (d) Self Organizing Map
3. Clustering Algorithms
  - (a) Hierarchial Agglomerative Clustering
  - (b) Hierarchial Division Clustering
4. Regression Algorithms
  - (a) Lasso Regression
5. Deep Learning Algorithms
  - (a) Deep Q-Learning
6. Other Methods
  - (a) Gradient Descent

---

**Algorithm 1** k-Nearest Neighbor [Tay et al., 2014] link:36

---

**Input:** X: training data, Y: Class labels of X,  $x$ : unknown sample

**Output:** Class label of unknown sample

```

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

```

---

---

**Algorithm 2** Adaboost [Schapire, 2014]

---

**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

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

$$(2)$$

**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
```

---

---

**Algorithm 3** Adaboost [Hertzmann et al., 2015]

---

**Input:**Training data  $\{(x_i, y_i)_{i=1}^N$  where  $x_i \in \mathbb{R}^k$  and  $y_i \in \{-1, 1\}\}$ **Output:** Weighted sum that represents the final output of the boosted classifier

```
1: Given Training data  $\{(x_i, y_i) \text{ where } y_i \in \{-1, 1\}\}$ 
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
```

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

$$(4)$$

```
9:   renormalize
10: end for
11: output final classifier  $H_{final}(x) = \text{sign}(\sum \alpha_t h_t(x))$ 
12: end for
```

---

---

**Algorithm 4** Random forest [Bernstein, 2016]

---

**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  $i \in 1, \dots, B$  do
4:      $S^{(i)} \leftarrow$  A bootstrap sample from  $S$ 
5:      $h_i \leftarrow \text{RANDOMIZEDTREELEARN}(S^{(i)}, F)$ 
6:      $H \leftarrow H \cup \{h_i\}$ 
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 [., 2015]

---

**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:       $T_x = \text{ID3}(D_x, X - \{x\})$ 
18:    end if
19:    Add a branch from  $T$  to  $T_x$  labeled by  $x$ 
20:  end for
21:  return  $T$ 
end function
```

---

---

**Algorithm 6** Perceptron [Brownlee, 2015d]

---

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

**Output:**  $Weights$

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

---

---

**Algorithm 7** Back-propagation [Brownlee, 2015a]

---

**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 [Brownlee, 2015c]

---

**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:    $Pattern_i \leftarrow SelectInputPattern(InputPatterns)$ 
4:    $Bmu_i \leftarrow SelectBestMatchingUnit(Pattern_i, CodebookVectors)$ 
5:   for  $Bmu_i^{attribute} \in Bmu_i$  do
6:     if  $Bmu_i^{class} \equiv Pattern_i^{class}$  then
7:        $Bmu_i^{attribute} \leftarrow Bmu_i^{attribute} + learn_{rate} \times (Pattern_i^{attribute} - Bmu_i^{attribute})$ 
8:     else
9:        $Bmu_i^{attribute} \leftarrow Bmu_i^{attribute} - learn_{rate} \times (Pattern_i^{attribute} - Bmu_i^{attribute})$ 
10:    end if
11:  end for
12: end for
13: Return  $CodebookVectors$ 
```

---

---

**Algorithm 9** Self Organizing Map [Brownlee, 2015b]

---

**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:    $Learn_{rate}^i \leftarrow CalculateLearningRate(i, learn_{rate}^{init})$ 
4:    $neighborhood_{size}^i \leftarrow CalculateNeighborhoodSize(i, neighborhood_{init}^{size})$ 
5:    $Pattern_i \leftarrow SelectInputPattern(InputPatterns)$ 
6:    $Bmu_i \leftarrow SelectBestMatchingUnit(Pattern_i, CodebookVectors)$ 
7:    $Neighborhood \leftarrow Bmu_i$ 
8:    $Neighborhood \leftarrow SelectNeighbors(Bmu_i, CodebookVectors, neighborhood_{size}^i)$ 
9:   for  $Vector_i \in Neighborhood$  do
10:    for  $Vector_i^{attribute} \in Vector_i$  do
11:       $Vector_i^{attribute} \leftarrow Vector_i^{attribute} + learn_{rate} \times (Pattern_i^{attribute} - Vector_i^{attribute})$ 
12:    end for
13:  end for
14: end for
15: Return  $CodebookVectors$ 
```

---

---

**Algorithm 10** Hierarchial Agglomerative Algorithm [Stein, 2016a]

---

**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:  $C = \{\{v \mid v \in V\}\}$  ▷ Initial Clustering
2:  $V_t = \{v_C \mid C \in C\}, E_T = \emptyset$  ▷ Initial Dendogram
3: while  $|C| > 1$  do
4:    $update\_distance\_matrix(C, G, d_c)$ 
5:    $\{C, C'\} = \underset{\{C_i, C_j\} \in C: C_i \neq C_j}{argmin} d_c(C_i, C_j)$ 
6:    $C = (C \setminus \{C, C'\}) \cup \{C \cup C'\}$  ▷ 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  $T$ 
```

---

---

**Algorithm 11** Hierarchial Divisive Algorithm [Stein, 2016b]

---

**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:    $C = (C \setminus \{C, C'\}) \cup \{C' \cup C\}$  ▷ 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  $T$ 
```

---

---

**Algorithm 12** C4.5 [Dai and Ji, 2014]

---

**Input:** $T$  : Training dataset  
 $S$  : Attributes**Output:** decision tree  $Tree$ 

```
1: function C4.5( $T$ )
2:   if  $T$  is NULL then
3:     return failure
4:   end if
5:   if  $S$  is NULL then
6:     return  $Tree$  as a single node with most frequent class label in  $T$ 
7:   end if
8:   if  $|S| = 1$  then
9:     return  $Tree$  as a single node  $S$ 
10:  end if
11:  set  $Tree = \{\}$ 
12:  for  $a \in S$  do
13:    set  $Info(a, T) = 0$  and  $SplitInfo(a, T) = 0$ 
14:    compute  $Entropy(a)$ 
15:    for  $v \in values(a, T)$  do
16:      set  $T_{a,v}$  as the subset of  $T$  with attribute  $a = v$ 
17:       $Info(a, T) += \frac{|T_{a,v}|}{|T_a|} Entropy(a)$ 
18:       $SplitInfo(a, T) += -\frac{|T_{a,v}|}{|T_a|} \log \frac{|T_{a,v}|}{|T_a|}$ 
19:    end for
20:     $Gain(a, T) = Entropy(a) - Info(a, T)$ 
21:     $GainRatio(a, T) = \frac{Gain(a, T)}{SplitInfo(a, T)}$ 
22:  end for
23:  set  $a_{best} = \operatorname{argmax}\{GainRatio(a, T)\}$ 
24:   $a_{best}$  into  $Tree$ 
25:  for  $v \in values(a_{best}, T)$  do call C4.5( $T_{a,v}$ )
26:  end for
27:  return  $Tree$ 
28: end function
```

---

---

**Algorithm 13** Gradient Descent

---

**Input:** $f$   
starting value  $x_1$   
termination tolerances**Output:**  $x_{maxIters}$ 

```
1: for  $i = 1$  to  $maxIters$  do
2:   Compute the search direction  $d_t = -\delta f(x_t)$ 
3:   if  $|d_T| < \epsilon_g$  then
4:     return "Converged to critical point", output  $x_t$ 
5:     Find  $\alpha_t$  so that  $f(x_t + \alpha_t d_t) < f(x_t)$ 
6:   end if
7:   if  $|\alpha_t d_T| < \epsilon_x$  then
8:     return "Converged in x", output  $x_t$ 
9:     Find  $\alpha_t$  so that  $f(x_t + \alpha_t d_t) < f(x_t)$ 
10:  end if
11:  Let  $x_{t+1} = x_t + \alpha_t d_t$ 
12: end for
13: Return "Max number of iterations reached", output  $x_{maxIters}$ 
```

---

---

**Algorithm 14** Naive Bayes

---

**Input:**

$C$  : A fixed set of classes  
 $D$  : Documents

**Output:** Category(Class) of the Documents

```
1: function TRAINMULTINOMIALNB( $C, D$ )
2:    $V \leftarrow \text{EXTRACTVOCABULARY}(D)$ 
3:    $N \leftarrow \text{COUNTDOCS}(D)$ 
4:   for each  $c \in C$  do
5:      $N_c \leftarrow \text{COUNTDOCSINCLASS}(D, c)$ 
6:      $\text{prior}|c| \leftarrow N_c/N$ 
7:      $\text{text}_c \leftarrow \text{CONCATENATE TEXT OF ALL DOCS IN CLASS}(D, C)$ 
8:     for each  $t \in V$  do
9:        $\text{condprob}|t||c| \leftarrow \frac{T_{ct}+1}{\sum_{t'}(T_{ct'}+1)}$ 
10:    end for
11:  end for
12:  return  $V, \text{prior}, \text{condprob}$ 
13: end function
14: function APPLYMULTINOMIALNB( $C, D, \text{prior}, \text{condprob}, d$ )
15:    $W \leftarrow \text{EXTRACTTOKENS FROM DOC}(V, d)$ 
16:   for each  $c \in C$  do
17:      $\text{score}|c| \leftarrow \log \text{prior}|c|$ 
18:     for each  $t \in W$  do
19:        $\text{score}|c|+ \leftarrow \log \text{condprob}|t||c|$ 
20:     end for
21:   end for
22:   return  $\arg \max_{c \in C} \text{score}|c|$ 
23: end function
```

---

---

**Algorithm 15** Lasso Regression

---

**Input:**

$\text{ipy}$  : Inner product vector,  $\text{ipy}_i = \langle y, X_{\cdot i} \rangle$   
 $\text{ipx}$  : Inner product matrix,  $\text{ipx}_{ij} = \langle X_{\cdot i}, X_{\cdot j} \rangle$   
 $\lambda$  : Penalty parameter  
 $N$  : Number of samples

**Output:**  $\text{beta}$  : Regression parameter vector

```
1: function FASTLASSO( $\text{ipy}, \text{ipx}, \lambda, N$ )
2:   stop_thr ▷ Threshold for stopping iteration
3:    $p \leftarrow \text{length}(\text{ipy})$ 
4:    $\text{beta} \leftarrow 0$  with length  $p$ 
5:    $gc \leftarrow 0$  with length  $p$ 
6:   while  $\text{difBeta}_{\max} \geq \text{stop\_thr}$  do
7:      $\text{difBeta}_{\max} \leftarrow 0$ 
8:     for  $j = 1 \leftarrow p$  do
9:        $z \leftarrow (\text{ipy}[j] - gc[j])/N + \text{beta}[j]$ 
10:       $\text{beta\_tmp} \leftarrow \max(0, z - \lambda) - \max(0, -z - \lambda)$ 
11:       $\text{difBeta} \leftarrow \text{beta\_tmp} - \text{beta}[j]$ 
12:       $\text{difabs} \leftarrow \text{abs}(\text{difBeta})$ 
13:      if  $\text{difabs} > 0$  then
14:         $\text{beta}[j] \leftarrow \text{beta\_tmp}$ 
15:         $gc \leftarrow gc + \text{ipx}[j] \times \text{difBeta}$ 
16:         $\text{difBeta}_{\max} = \max(\text{difBeta}_{\max}, \text{difabs})$ 
17:      end if
18:    end for
19:  end while
20: end function
```

---

---

**Algorithm 16** Bagging

---

**Input:**

B: the number of bags or base hypotheses

L: Base Learning Algorithm

**Output:** New Training Sets

```
1: function BAGGING(examples, B, L)
2:   for i = 1 to B do
3:     examplesi ← a bootstrap sample of examples
4:   end for
5:   Compute set I containing indices for the k smallest distances  $d(X_i, x)$ 
6:   hi ← apply L to examplesi
7:   Return h1, h2, ... hB
8: end function
```

---

---

**Algorithm 17** Deep Q-Learning with Experience Replay [Mnih et al., 2013]

---

**Input:**

D: data set

Q: Action-Value Function

**Output:** New Training Sets

```
1: for i = 1 to M do
2:   Initialise sequence  $s_1 = \{x_1\}$  and preprocessed sequenced  $\phi = \phi(s_1)$ 
3:   for i = 1 to T do
4:     With probability  $\epsilon$  select a random action  $a_t$  otherwise select  $a_t = \max_a Q * (\phi(s_t).a : \theta)$ 
5:     Execute action  $a_t$  in emulator and observe reward r and image  $x_{t+1}$ 
6:     Set  $s_{t+1} = s_t, a_t, x_{t+1}$  and preprocess  $\phi_{t+1} = \phi(s_{t+1})$ 
7:     Store transition  $(\phi_t, a_t, r_t, \phi_{t+1})$  in D
8:     Set  $y_j =$ 
```

$$\begin{cases} r_j, & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta), & \text{for terminal } \phi_{j+1} \end{cases} \quad (5)$$

$$\quad (6)$$

```
9:   Perform a gradient descent step on  $(y_j - Q(\phi_j, a_j; \theta))^2$  according to the following equation
```

```
10:
```

$$\Delta_{\theta} L_i(\theta_i) = \mathbb{E}_{s, a \sim \rho(\cdot); s' \sim \epsilon[(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i)) \Delta_{\theta_i} Q(s, a; \theta_i)]}$$

```
11:   end for
```

```
12: end for
```

---



---

**Algorithm 18** PageRank

---

**Input:** $G$ : inlink file $iteration$ : Number of iteration**Output:** PageRank

```
1: function PAGERANK( $G, iteration$ )
2:    $d \leftarrow 0.85$  ▷ damping factor: 0.85
3:    $oh \leftarrow G$  ▷ get outlink hash from G
4:    $ih \leftarrow G$  ▷ get inlink hash from G
5:    $N \leftarrow G$  ▷ get number of pages from G
6:   for all  $p$  in the graph do
7:      $opg[p] \leftarrow \frac{1}{N}$ 
8:   end for
9:   while  $iteration > 0$  do
10:     $dp \leftarrow 0$ 
11:    for all  $p$  that has no out-links do
12:       $dp \leftarrow dp + d * \frac{opg[p]}{N}$ 
13:    end for
14:    for all  $p$  in the graph do
15:       $npg[p] \leftarrow dp + \frac{[1-d]}{N}$ 
16:      for all  $ip$  in  $ih[p]$  do
17:         $npg[p] \leftarrow dp + \frac{d * opg[ip]}{oh[ip]}$ 
18:      end for
19:    end for
20:     $opg \leftarrow npg$ 
21:     $iteration \leftarrow iteration - 1$ 
22:  end while
23: end function
```

---

## References

- [., 2015] . (2015). Decision trees.
- [Bernstein, 2016] Bernstein, M. (2016). Random forests.
- [Brownlee, 2015a] Brownlee, J. (2015a). Back-propagation.
- [Brownlee, 2015b] Brownlee, J. (2015b). Clever algorithms: Nature-inspired programming recipes.
- [Brownlee, 2015c] Brownlee, J. (2015c). Learning vector quantization.
- [Brownlee, 2015d] Brownlee, J. (2015d). Perceptron.
- [Dai and Ji, 2014] Dai, W. and Ji, W. (2014). A mapreduce implementation of c4.5 decision tree algorithm.
- [Hertzmman et al., 2015] Hertzmman, A., Fleet, D., and Brubaker, M. (2015). Adaboost.
- [Mnih et al., 2013] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. A. (2013). Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602.
- [Schapire, 2014] Schapire, R. (2014). Machine learning algorithms for classification.
- [Stein, 2016a] Stein, B. (2016a). Unit hierarchial cluster analysis.
- [Stein, 2016b] Stein, B. (2016b). Unit hierarchial cluster analysis.
- [Tay et al., 2014] Tay, B., Hyun, J., and Sejong, O. (2014). A machine learning approach for specification of spinal cord injuries using fractional anisotropy values obtained from diffusion tensor images.

---

**Algorithm 19** DBSCAN link:42

---

**Input:***D*: Data $\epsilon$ : Threshold distance*MinPts* : Minimum number of points required to form a cluster**Output:** Clustered Data

```
1: function DBSCAN(D,  $\epsilon$ , minPts)
2:   C = 0
3:   for each point P in dataset D do
4:     if P is visited then
5:       continue next point
6:     end if
7:     mark P as visited
8:     NeighborPts = regionQuery(P,  $\epsilon$ )
9:     if sizeof(NeighborPts) < MinPts then
10:      mark P as NOISE
11:    else
12:      C = next cluster
13:      expandCluster(P, NeighborPts, C,  $\epsilon$ , MinPts)
14:    end if
15: end for
end function
function EXPANDCLUSTER(P, NeighborPts, C,  $\epsilon$ , MinPts)
18: add P to Cluster C
19: for each point P' in NeighborPts do
20:   if P' is not visited then
21:     mark P' as visited
22:     NeighborPts' = regionQuery(P',  $\epsilon$ )
23:     if sizeof(NeighborPts')  $\geq$  MinPts then
24:       NeighborPts = NeighborPts joined with NeighborPts'
25:     end if
26:   end if
27:   if P' is not yet member of any cluster then
28:     add P' to cluster C
29:   end if
30: end for
end function
function REGIONQUERY(P,  $\epsilon$ )
33: return all points within P's  $\epsilon$  neighborhood
end function
```

---

---

**Algorithm 20** Logistic Regression

---

**Input:**Training data of the form  $\{(x_1, 1), (x_2, 0), \dots\}$  $x$ : unknown sample**Output:** The output is a probability that the given input point belongs to a certain class1:  $0 \leftarrow \beta$ 2: Compute  $y$  by setting its elements to

$$y = \begin{cases} 1, & \text{if } g_i = 1 \\ 0, & \text{if } g_i = 2 \end{cases} \quad (7)$$

(8)

 $i = 1, 2, \dots, N$ 3: Compute  $p$  by setting its elements to

$$p(x_i, \beta) = \frac{e^{\beta^T x_i}}{1 + e^{\beta^T x_i}}$$

 $i = 1, 2, \dots, N$ 4: Compute the diagonal matrix  $W$ . The  $i$ th diagonal element is  $p(x_i, \beta)(1 - p(x_i, \beta))$ 5:  $z \leftarrow X\beta + W^{-1}(y - p)$ 6:  $\beta \leftarrow (X^T W X)^{-1} X^T W z$ 7: If the stopping criteria, stop; otherwise go back to step 3

---