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
    (b) Logistic Regression
5. Deep Learning Algorithms
    (a) Deep Q-Learning
6. Other Methods
    (a) Gradient Descent
    (b) Gaussian Process

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

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 [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

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

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:
        ε_m = (Σ_{i=1}^N w_i^m I(f_m(x_i)) ≠ y_i) / (x^2 + 2x + 1)
        where I(f_m(x_i) ≠ y_i) = 1 if f_m(x_i) ≠ y_i and 0 otherwise
        α_m = ln (1 − ε_m)/ε_m
    end for
    for all i do
        w_i^{m+1} = w_i^{(m)} e^{α_m I(f_m(x_i) ≠ y_i)}
    end for
end for
```

- **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 [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

- 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 $H_{final(x)} = sign(\sum \alpha_t h_t(x))$
- **end for**

Algorithm 4 Random forest [Bernstein, 2016] Link:39

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

function RANDOMFOREST(S, F)
    $H \leftarrow \emptyset$
    for $i \in 1, \ldots.B$ **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 [., 2015] Link:40

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
            $T_x = \mathrm{ID3}(D_x, X - \{x\})$
    end if
    Add a branch from T to T_x labeled by x
end for
return T
end function

Algorithm 6 Perceptron [Brownlee, 2015d] Link:65

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 [Brownlee, 2015a]

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 Back-propagation1

Input:
Training Set $x^{(1)}, y^{(1)}, , ..., (x^{(m)}, y^{(m)})$
Output:
Gradient of the cost function

$\Delta_{ij}^{(l)} = 0$(for all l,i,j)
for $i = 1$ to m **do**
  Set $a^{(1)} = x^{(i)}$
  Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3, ...L$
  Using $y^(i)$, compute $\delta^(L) = a^(L) - y^(i)$
  Compute $\delta^(L-1), \delta^(L-2), ...\delta^(2)$
  $\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$
end for
$D_{ij}^{(l)} := \frac{1}{m}\Delta_{ij} + \lambda\theta_{ij}^{(l)}$ if $j \neq 0$
$D_{ij}^{(l)} := \frac{1}{m}\Delta_{ij}$ if $j = 0$

Algorithm 9 Learning Vector Quantization [Brownlee, 2015c]

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 10 Self Organizing Map [Brownlee, 2015b]

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**
  $Learn^{i}_{rate} \leftarrow CalculateLearningRate(i, learn^{init}_{rate})$
  $neighborhood^{i}_{size} \leftarrow CalculateNeighborhoodSize(i, neighborhood^{size}_{init})$
  $Pattern_i \leftarrow SelectInputPattern(InputPatterns)$
  $Bmu_i \leftarrow SelectBestMatchingUnit(Pattern_i, CodebookVectors)$
  $Neighborhood \leftarrow Bmu_i$
  $Neighborhood \leftarrow SelectNeighbors(Bmu_i, CodebookVectors, neighborhood^{i}_{size})$
  for $Vector_i \in Neighborhood$ **do**
    for $Vector^{attribute}_i \in Vector_i$ **do**
      $Vector^{attribute}_i \leftarrow Vector^{attribute}_i + learn_{rate} \times (Pattern^{attribute}_i - Vector^{attribute}_i)$
    end for
  end for
end for
Return $CodebookVectors$

Algorithm 11 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

$C = \{\{v \mid v \in V\}\}$ ▷ 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 12 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

$C = \{V\}$ ▷ 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 13 C4.5 [Dai and Ji, 2014]

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

function C4.5(T)
  if T is $NULL$ **then**
    return failure
  end if
  if S is $NULL$ **then**
    return $Tree$ as a single node with most frequent class label in T
  end if
  if $|S| = 1$ **then**
    return $Tree$ as a single node S
  end if
  set $Tree = \{\}$
  for $a \in S$ **do**
    set $Info(a, T) = 0$ and $SplitInfo(a, T) = 0$
    compute $Entropy(a)$
    for $v \in values(a, T)$ **do**
      set $T_{a,v}$as the subset of T with attribute $a = v$
      $Info(a, T) += \frac{|T_{a,v}|}{|T_a|} Entropy(a)$
      $SplitInfo(a, T) += -\frac{|T_{a,v}|}{|T_a|} \log \frac{|T_{a,v}|}{|T_a|}$
    end for
    $Gain(a, T) = Entropy(a) - Info(a, T)$
    $GainRatio(a, T) = \frac{Gain(a,T)}{SplitInfo(a,T)}$
  end for
  set $a_{best} = argmax\{GainRatio(a, T)\}$
  a_{best}into $Tree$
  for $v \in values(a_{best}, T)$ **do** call $C4.5(T_{a,v})$
  end for
  return $Tree$
end function

Algorithm 14 Gradient Descent

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

for $i = 1$ to $maxIters$ **do**
  Compute the search direction $d_t = -\delta f(x_t)$
  if $|d_T| < \epsilon_g$ **then**
    return "Converged to critical point", output x_t
    Find α_t so that $f(x_t + \alpha_t d_t) < f(x_t)$
  end if
  if $|\alpha_t d_T| < \epsilon_x$ **then**
    return "Converged in x", output x_t
    Find α_t so that $f(x_t + \alpha_t d_t) < f(x_t)$
  end if
  Let $x_{t+1} = x_t + \alpha_t d_t$
end for
Return "Max number of iterations reached", output $x_{maxIters}$

Algorithm 15 Naive Bayes

Input:

C : A fixed set of classes

D : Documents

Output: Category(Class) of the Documents

function TrainMultinomialNB(C, D)

  $V \leftarrow EXTRACTVOCABULARY(D)$

  $N \leftarrow COUNTDOCS(D)$

  for each $c \in C$ **do**

    $N_c \leftarrow COUNTDOCSINCLASS(D, c)$

    $prior|c| \leftarrow N_c/N$

    $text_c \leftarrow CONCATENATETEXTOFALLDOCSINCLASS(D, C)$

    for each $t \in V$ **do**

      $condprob|t||c| \leftarrow \frac{T_{ct}+1}{\sum_{t'}(T_{ct'+1})}$

    end for

  end for

  return $V, prior, condprob$

end function

function ApplyMultinomialNB($C, D, prior, condprob, d$)

  $W \leftarrow EXTRACTTOKENSFROMDOC(V, d)$

  for each $c \in C$ **do**

    $score|c| \leftarrow \log\ prior|c|$

    for each $t \in W$ **do**

      $score|c| + = \log condprob|t||c|$

    end for

  end for

  return $arg\ max_{c \in C} score|c|$

end function

Algorithm 16 Lasso Regression

Input:

ipy : Inner product vector, $ipy_i =< y, X_{\cdot i} >$

ipx : Inner product matrix, $ipx_{ij} =< X_{\cdot i}, X_{\cdot j} >$

λ : Penalty parameter

N : Number of samples

Output: $beta$: Regression parameter vector

function FastLasso(ipy, ipx, λ, N)

  `stop_thr` ▷ Threshold for stopping iteration

  $p \leftarrow length(ipy)$

  $beta \leftarrow 0$ with length p

  $gc \leftarrow 0$ with length p

  while $difBeta_{max} \geq$ `stop_thr` **do**

    $difBeta_{max} \leftarrow 0$

    for $j = 1 \leftarrow p$ **do**

      $z \leftarrow (ipy|j| - gc|j|)/N + beta|j|$

      `beta_tmp` $\leftarrow max(0, z - \lambda) - max(0, -z - \lambda)$

      $difBeta \leftarrow$ `beta_tmp` $- beta|j|$

      $difabs \leftarrow abs(difBeta)$

      if $difabs > 0$ **then**

        $beta|j| \leftarrow$ `beta_tmp`

        $gc \leftarrow gc + ipx|j| \times difBeta$

        $difBeta_{max} = max(difBeta_{max}, difabs)$

      end if

    end for

  end while

end function

Algorithm 17 Bagging

Input:

B: the number of bags or base hypotheses

L: Base Learning Algorithm

Output: New Training Sets

function BAGGING($examples, B, L$)
  for $i = 1$ to B **do**
    $examples_i \leftarrow$ a bootstrap sample of $examples$
  end for
  Compute set I containing indices for the k smallest distances $d(X_i, x)$
  $h_i \leftarrow$ apply$L$to$examples_i$
  Return $h_1, h_2, ...h_B$
end function

Algorithm 18 Deep Q-Learning with Experience Replay [Mnih et al., 2013]

Input:

D: data set

Q: Action-Value Function

Output: New Training Sets

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

$$\begin{cases} r_j, & \text{for terminal } \ \phi_{j+1} \quad (5) \\ r_j + \gamma max_{a'} Q(\phi_{j+1,a';\theta}), & \text{for terminal } \phi_{j+1} \quad (6) \end{cases}$$

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

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

  end for
end for

Algorithm 19 PageRank

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

function PageRank(G, *iteration*)
  $d \leftarrow 0.85$ ▷ damping factor: 0.85
  $oh \leftarrow G$ ▷ get outlink hash from G
  $ih \leftarrow G$ ▷ get inlink hash from G
  $N \leftarrow G$ ▷ get number of pages from G
  for all p in the graph **do**
    $opg[p] \leftarrow \frac{1}{N}$
  end for
  while $iteration > 0$ **do**
    $dp \leftarrow 0$
    for all p that has no out-links **do**
      $dp \leftarrow dp + d * \frac{opg[p]}{N}$
    end for
    for all p in the graph **do**
      $npg[p] \leftarrow dp + \frac{[1-d]}{N}$
      for all ip in $ih[p]$ **do**
        $npg[p] \leftarrow dp + \frac{d*opg[ip]}{oh[ip]}$
      end for
    end for
    $opg \leftarrow npg$
    $iteration \leftarrow iteration - 1$
  end while
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.

[Hertzmann et al., 2015] Hertzmann, 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 20 DBSCAN link:42

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

function DBSCAN($D, \epsilon, minPts$)
  $C = 0$
  for each point Pin dataset$D$ **do**
    if P is visited **then**
      continue next point
    end if
    mark P as visited
    $NeighborPts = regionQuery(P, \epsilon)$
    if $sizeof(NeighborPts) < MinPts$ **then**
      mark P as NOISE
    else
      C = next cluster
      $expandCluster(P, NeighborPts, C, \epsilon, MinPts)$
  end if
end for
end function
function expandCluster($P, NeighborPts, C, \epsilon, MinPts$)
add P to Cluster C
for each point $P' in NeighborPts$ **do**
  if P' is not visited **then**
    mark P' as visited
    $NeighborPts' = regionQuery(P', eps)$
    if $sizeof(NeighborPts) >= MinPts$ **then**
      $NeighborPts = NeighborPts$ joined with $NeighborPts'$
    end if
  end if
  if P' is not yet member of any cluster **then**
    add P' to cluster C
  end if
end for
end function
function regionQuery(P, ϵ)
return all points within $P's\ \epsilon\ neighborhood$
end function

Algorithm 21 Logistic Regression

Input:

Training data of the form $\{(x_1, 1), (x_2, 0), ..\}$

x: unknown sample

Output: The output is a probability that the given input point belongs to a certain class

1: $0 \leftarrow \beta$

2: Compute y by setting its elements to

$$y = \begin{cases} 1, & \text{if } g_i = 1 \quad (7) \\ 0, & \text{if } g_i = 2 \quad (8) \end{cases}$$

i = 1,2,..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,..N

4: Compute the diagonal matrix W. The ith 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

Algorithm 22 Gaussian Process

Input:

$X = \begin{bmatrix} x_1^T \\ \dots \\ x_n^T \end{bmatrix} \in \mathbb{R}^{n \times D}$, m training inputs

$y = \begin{bmatrix} y_1^T \\ .. \\ y_n^T \end{bmatrix} \in \mathbb{R}^n$

$k(.,.) : \mathbb{R}^{D \times D}$

x_*test input

σ^2noise level on the observations

$$[y(x) = f(x) + \epsilon, \epsilon \sim N(0, \sigma^2)]$$

Output:

f_*

$cov(f_*)$

1: $K \in \mathbb{R}^{n \times n}$Gram matrix.$K_{ij} = k(x_i, x_j)$

$$k(x_*) = k_* = k(X, x_*) = \begin{bmatrix} k(x_1, x_*) \\ \dots\dots\dots \\ k(x_n, x_*) \end{bmatrix} \in \mathbb{R}$$

2: $\alpha = (K + \sigma_2 \mathbb{I}_n)^{-1} y$

3: $f_* = k_*^T \alpha \in \mathbb{R}$

4: $cov(f_*) = k(x_*, x_*) - k_*^T [K + \sigma_2 \mathbb{I}n]^{-1} k_*$