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

Algorithm 1 k-Nearest Neighbor [Tay et al., 2014]

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 [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:
    $\epsilon_m = \frac{\sum_{i=1}^N w_i^m I(f_m(x_i)) \neq y_i}{x^2+2x+1}$
    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

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.

[Schapire, 2014] Schapire, R. (2014). Machine learning algorithms for classification.

[Stein, 2016a] Stein, B. (2016a). Unit hierarchial cluster analysis.

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: 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 [Bernstein, 2016]

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 [., 2015]

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 [Brownlee, 2015d]

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 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 9 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**
  $Learn_{rate}^{i} \leftarrow CalculateLearningRate(i, learn_{rate}^{init})$
  $neighborhood_{size}^{i} \leftarrow CalculateNeighborhoodSize(i, neighborhood_{init}^{size})$
  $Pattern_i \leftarrow SelectInputPattern(InputPatterns)$
  $Bmu_i \leftarrow SelectBestMatchingUnit(Pattern_i, CodebookVectors)$
  $Neighborhood \leftarrow Bmu_i$
  $Neighborhood \leftarrow SelectNeighbors(Bmu_i, CodebookVectors, neighborhood_{size}^{i})$
  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 [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**
  $update\_distance\_matrix(C, G, d_c)$
  $\{C, C'\} = \underset{\{C_i, C_j\} \in C : C_i \neq C_j}{argmin} d_c(C_i, C_j)$
  $C = (C \setminus \{C, C'\}) \cup \{C \cup C'\}$ ▷ 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 [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 12 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 13 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 14 Naive Bayes

Input:

C : A fixed set of classes

D : Documents

Output: a predicted class $c \in C$

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 15 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 ← length(ipy)
    beta ← 0 with length p
    gc ← 0 with length p
    while difBeta_max ≥ stop_thr do
        difBeta_max ← 0
        for j = 1 ← p do
            z ← (ipy|j| − gc|j|)/N + beta|j|
            beta_tmp ← max(0, z − λ) − max(0, −z − λ)
            difBeta ← beta_tmp − beta|j|
            difabs ← abs(difBeta)
            if difabs > 0 then
                beta|j| ← beta_tmp
                gc ← gc + ipx|j| × difBeta
                difBeta_max = max(difBeta_max, difabs)
            end if
        end for
    end while
end function
```


[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.