1. Classification
   (a) K-Nearest Neighbor
   (b) Support Vector Machines
   (c) Adaboost
   (d) Iterative Dichotomiser 3
   (e) C4.5
   (f) Naive Bayes
   (g) Bagging
   (h) 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 ∈ 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

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 Adaboost4 [Hertzmann et al., 2015] Link:23,35,93,95

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

Set uniform example weights.
for each base learner do **do**
  Train base learner with weighted sample.
  Test base learner on all data.
  Set learner weight with weighted error.
  Set example weights based on ensemble predictions.
end for

Algorithm 5 Random forest [Bernstein, 2016] Link:42

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 6 Iterative Dichotomiser 3 [., 2015a] 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 = 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 7 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 8 Back-propagation [Brownlee, 2015a] Link:17,42

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 9 Back-propagation2 [Ng,]

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 $j = 0$
$\frac{\partial}{\partial\theta_{ij}^{(l)}} J(\theta) = D_{ij}^(l)$

Algorithm 10 Learning Vector Quantization [Brownlee, 2015c] Link : 50 and 58

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 11 Self Organizing Map [Brownlee, 2015b] Link:45

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 12 Hierarchial Agglomerative Algorithm [Stein, 2016a] Link:24,54

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 13 Hierarchial Agglomerative Algorithm 2 [**?**] Link:25,54

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

while More than one cluster remains **do**
  Compute the proximity graph if necessary
  repeat
  Merge the closest two clusters.
  Update the proximity matrix to reflect the proximity between the new cluster and the original clusters.
end while

Algorithm 14 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 15 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

while More than one cluster remains **do**
  Create a new cluster by breaking the link corresponding to the largest distance (smallest similarity).
end while

Algorithm 16 C4.5 [Dai and Ji, 2014] Link :22,55

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 17 Gradient Descent [., 2015b] Link : 59

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 18 Naive Bayes [., d] Link: 9, 56

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 19 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 20 Bagging [., a] Link:57

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 21 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 22 PageRank [., e]

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

Algorithm 23 K means

Input:

$E = \{e_1, e_2, ...e_k\}$
k(number of clusters)
$MaxIters$(limit of iterations)

Output:

$C = \{c_1, c_2, ...c_k\}$
$L = \{l(e)|e = 1, 2, ...n\}$ (set of cluster labels of E)

for each $c_i \in C$ **do**
    $c_i \leftarrow e_j \in E$(e.g random selection)
end for
for each $e_i \in E$ **do**
    $l(e_i) \leftarrow argminDistance(e_i, c_j)j \in \{1..k\}$
end for
$changed \leftarrow false$
$iter \leftarrow 0$
while $changed = trueanditer <= MaxIters$ **do**
    for $c_i \in C$ **do**
        $UpdateCluster(c_i)$
    end for
    for $c_i \in E$ **do**
        $minDist \leftarrow argminDistance(e_i, e_j)j \in \{1..k\}$
        if $minDist \neq l(e_i)$ **then**
            $l(e_i) \leftarrow minDist$
            $changed \leftarrow true$
        end if
    end for
    $iter$++
end while

Algorithm 24 DBSCAN link:42 [Ram et al., 2010]

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 $P'inNeighborPts$ **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 $P's$ ϵ $neighborhood$
end function

Algorithm 25 Principle Component Analysis [**?**]

Input:
$x_1, ...x_n$d length vector k
Output: Transform matrix R

$X \leftarrow n \times d$ data matrix with x_i in each row
$x \leftarrow \frac{1}{n}\sum_{i=1}^{n} x_i$
$X \leftarrow$ subtract x from each row $x_i inX$
$COV \leftarrow \frac{1}{n-1}X^T \times X$Compute eigenvalue $e_1, ..., e_d$ of COV, and sort them
Compute matrix V which satisfy $V^{-1} \times COV \times V = D$, D is the diagonal matrix of eigenvalue of COV
$R \leftarrow$ the first k column of V

Algorithm 26 Logistic Regression [., c]

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 27 Gaussian Process [., b]

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

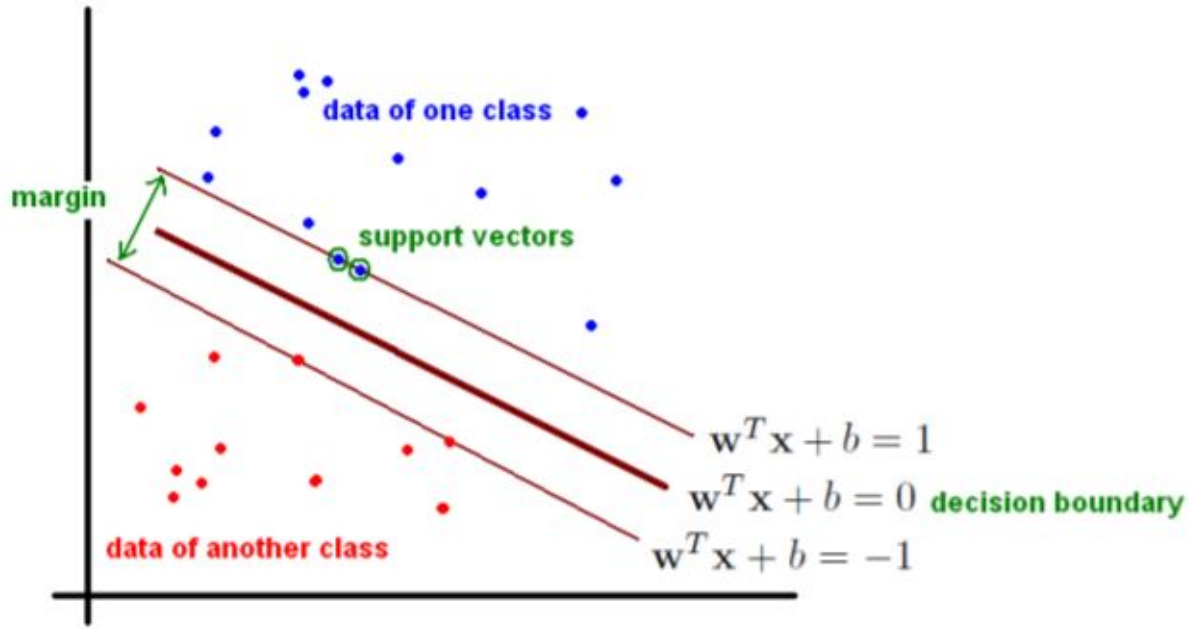
3: $f_* = k_*^T \alpha \in \mathbb{R}$

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

Algorithm 28 Support Vector Machines [Gavrilov,]

Input:

Set of N input-output pairs $\{x, y\}^{N_1}$ x: input vectors of the same dimension and y : set of output target labels $y_i = \{0, 1\}$



1 Simple case : linearly-separable data,binary classification

Goal:we want to find the hyperplane(i.e. decision boundary)linearly separating our classes. Our boundary will have the equation: $\mathbf{w^T x} + \mathbf{b} = 0$

Anything above the decision boundary should have label 1 i.e., $\mathbf{w^T x}_i + b > 0$ will have corresponding $y_i = 1$

Similarly, anything below the decision boundary should have label -1 i.e. $\mathbf{w^T x}_i + b < 0$ will have corresponding $y_i = -1$

The reason for this labeling scheme is that it lets us condense for the decision function to

$$f(x) = sign(\mathbf{w^T} + b)$$

since $f(x) = +1$ for all x above the boundary , and $f(x) = -1$ for all x below the boundary.

Thus, we can figure out if an instance has been classified properly by checking that $y(\mathbf{w^T} + b) \geq 1$ (which will be the case as long as either both $y, \mathbf{w^T} + b > 0$ or else $y, \mathbf{w^T} + b < 0$)

You'll notice that we will now have some space between our decision boundary and the nearest data points of either class. Thus,let's rescale the data such that anything on or above the boundary $\mathbf{w^T x} + b = 1$ is of one class(with label 1), and anything on or below the boundary $\mathbf{w^T x} + b = -1$ is of another class (with label -1)

What is the distance between these newly added boundaries?

First note that the two lines are parallel,and thus share their parameters w, b.Pick an arbitary point x_1 to lie on line $\mathbf{w^T x} + b = -1$. Then the closest point on line $\mathbf{w^T x} + b = 1$ is the point $\mathbf{x}_2 = \mathbf{x}_1 + \lambda\mathbf{w}$(since the closest point will always lie on the perpendicular;recall that the vector $\mathbf{w}$ is perpendicular to both lines). Using this formulation, $\lambda\mathbf{w}$ will be the line segment segment connecting $\mathbf{x}_1$ and $\mathbf{x}_2$, and thus, $\lambda \|\mathbf{w}\|$, the distance between $\mathbf{x}_1$ and $\mathbf{x}_2$ is the shortest distance between the two lines/boundaries.

References

[., a] . (.a). Bagging.

[., b] . (.b). Gaussian process.

[., c] . (.c). Logistic regression.

[., d] . (.d). Naive bayes.

[., e] . (.e). Page rank.

[., 2015a] . (2015a). Decision trees.

[., 2015b] . (2015b). Gradient descent.

[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.

[Gavrilov,] Gavrilov, Z. (.). Support vector machines.

[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.

[Ng,] Ng, A. Coursera machine learning.

[Ram et al., 2010] Ram, A., Jalal, S., Jalal, A. S., and Kumar, M. (2010). A density based algorithm for discovering density varied clusters in large spatial databases. *International Journal of Computer Applications*, 3(6):1–4.

[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 28 Support Vector Machines (continued)

Solving for λ : $\mathbf{w^T x}_2 + b = 1$ where $\mathbf{x}_2 = \mathbf{x}_1 + \lambda \mathbf{w}$
$\mathbf{w^T}(\mathbf{x}_1 \lambda \mathbf{w}) + b = 1$ $\mathbf{w^T}(\mathbf{x}_1 \lambda \mathbf{w}) + b = 1$
$\mathbf{w^T x}_1 + b + \lambda \mathbf{w^T w} = 1$ where $\mathbf{w^T x}_1 + b = -1$
$-1 + \lambda \mathbf{w^T w} = 1$
$\lambda \mathbf{w^T w} = 2$
$\lambda = \frac{2}{\mathbf{w^T w}} = \frac{2}{\|\mathbf{w}\|^2}$
And, so the distance $\lambda \|w\| is \frac{2}{\|\mathbf{w}\|^2} \|w\| = \frac{2}{\|w\|} = \frac{2}{\sqrt{\mathbf{w^T w}}}$

It's intuitive that we would want to maximize the distance between the two boundaries demarcating the classes(Why?We want to be as sure that we are not making classification mistakes and thus we want our data points from the two classes to lie as far away from each other as possible).This distance is called the margin,so went to obtain the maximal margin.

Thus, we want to maximize $\frac{2}{\sqrt{\mathbf{w^T w}}}$, which is equivalent to minimizing $\frac{\sqrt{\mathbf{w^T w}}}{2}$ which is in turn equivalent to minimizing $\frac{\mathbf{w^T w}}{2}$(since square root is a monotonic function)

This quadratic programming problem is expressed as :
$min_{w,b} \frac{\mathbf{w^T w}}{2}$
subject to : $y_i(\mathbf{w}^T \mathbf{x} + b) \geq 1 (\forall$ data points $\mathbf{x}_i)$

2 Soft-margin extension

Consider the case that your data isn't linearly separable.For instance, maybe you aren't guaranteed that all your data points are correctly labelled, so you want to allow some data points of one class to appear on the other side of the boundary.

We can introduce *slackvariables*an $- \epsilon_i \geq 0$. Our quadratic programming problem becomes:
$min_{\mathbf{w},b,\epsilon} \frac{\mathbf{w^T w}}{2} + C \sum_i \epsilon_i$
subject to : $y_i(\mathbf{w^T x}_i + b) \geq 1 - \epsilon$

3 Nonlinear decision boundary

Mapping your data vectors, x_i,into a higher-dimension(even infinite) feature space may make them linearly separable in that space (whereas they may not be linearly separable in the original space). The formation of the quadratic programmatic problem is as above,but with all $\mathbf{x}_i$ replaced with $\phi(\mathbf{x}_i)$, where ϕprovides the higher-dimensional mapping.So we have the standard SVM formulation:
$min_{\mathbf{w},b,\epsilon} \frac{\mathbf{w}^T \mathbf{w}}{2}$
subject to : $y_i(\mathbf{w}^T \phi(\mathbf{x}_i) + b) \geq 1 - \epsilon$and $\epsilon_i \geq 0 (\forall$ data points $\mathbf{x}_i)$

4 Reformulating as a Lagrangian

We can introduce Lagrange multipliers to represent the condition:
$y_i(\mathbf{w}^T \phi(\mathbf{x}_i + b))$ must be as close to 1 as possible. This condition is captured by: $max_{ai \geq 0} \alpha_i [1 - y_i(\mathbf{w}^T \phi(\mathbf{x}_i + b))]$ This ensures that when $y_i(\mathbf{w}^T \phi(\mathbf{x}_i + b)) \geq 1$, the expression above is maximal when $\alpha_i = 0$(since [1-$y_i(\mathbf{w}^T \phi(\mathbf{x}_i + b)]$ ends up being negative). Otherwise, $y_i(\mathbf{w}^T \phi(\mathbf{x}_i + b)) < 1$, so $[1 - y_i(\mathbf{w}^T \phi(\mathbf{x}_i + b))]$ is a positive value, and the expression is maximal when $a_i \to \infty$. This has the effect of penalizing any misclassified data points, while assigning 0 penalty to properly classified instances.

We thus have the following formulation:
$min_{w,b} [\frac{\mathbf{w}^T \mathbf{w}}{2} + \sum_i max_{\alpha \geq 0} \alpha_i [1 - \mathbf{w}^T \phi(\mathbf{x}_i + b)]]$
To allow for slack(soft-margin), preventing the α variables from going to ∞, we can impose constraints on the Lagrange multipliers to lie within: $0 \leq \alpha_i \leq C$. We can define the dual problem by interchanging the max and min as follows (i.e minimize after fixing alpha):
$max_{alpha \geq zero} [min_{w,b} J(\mathbf{w}, b; \alpha)]$where $= \frac{\mathbf{w}^T \mathbf{w}}{2} + \sum_i \alpha_i [1 - y_i(\mathbf{w}^T \phi(x_i) + b)]$

Algorithm 28 Support Vector Machines (continued)

Since, we're solving an optimization problem, we set $\frac{\partial J}{\partial \mathbf{w}} = 0$ and discover that the optimal setting of $\mathbf{w}$ is $\sum_i \alpha_i y_i \phi(x_i)$,while seeting $\frac{\partial J}{\partial \mathbf{b}} = 0$ yields the constraint $\sum_i \alpha_i y_i = 0$

Thus, after substituting and simplifying,we get:
$min_{w,b} J(\mathbf{w}, b, \alpha) = \sum_i \alpha_i \alpha_j y_i y_j \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$ And thus our dual is:
$max_{\alpha \geq 0} [\sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_j y_i y_j \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)]$
Subject to: $\sum_i \alpha_i y_i = 0$ and $0 \leq \alpha_i \leq C$

5 Kernel trick

Because we're working in a higher-dimension space(and potentially even an infinite-dimensional space),calculating $\phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$ may be intractable.However, it turns out there are special *kernel* functions that operate on the lower dimension vectors $\mathbf{x}_i$ and $\mathbf{x}_j$ to produce a value equivalent to the dot-product of the higher dimensional vectors. For instance, consider the function $\phi: \mathbb{R}^3 \longmapsto \mathbb{R}^{10}, where$: $\phi(x) = (1, \sqrt{2}x^{(1)}, \sqrt{2}x^{(2)}, \sqrt{2}x^{(3)}, [\mathbf{x}^{(1)}]^2, [\mathbf{x}^{(2)}]^2, [\mathbf{x}^{(3)}]^2, \sqrt{2}x^{(1)(2)}, \sqrt{2}x^{(1)(3)}, \sqrt{2}x^{(2)(3)})$ Instead, we have the the kernel trick, which tells us that $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^2 = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$ for the given ϕ. Thus, we can simplify our calculations. Re-writing the dual in terms of the kernel yields: $max_{\alpha \geq 0} [\sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j]$

6 Decision function

To classify a novel instance $\mathbf{x}$ once you've learned the optimal α_i parameters, all you have to do is calculate $f(\mathbf{x}) = sign(\mathbf{w}^T \mathbf{x} + b) = \sum_i \alpha_i y_i \phi(\mathbf{x}_i$ and using the kernel trick). Note that α_i is only non-zero for instances $\phi(\mathbf{x}_i)$ on or near the boundary-those are called the *supportvector* since they alone specify the decision boundary. We can toss out the other data points once training is complete. Thus, we only sum over the x_i which constitute the support vectors.