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
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 $\{Y_i$ 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
```

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

```
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 $\{(x_i, y_i)$ 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
```

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

```
 9:         renormalize
10:     end for
11:     output final classifier $H_{final(x)} = sign(\sum \alpha_t h_t(x))$
12: 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
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 [., 2015] Link:40

```
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 [Brownlee, 2015d] Link:65

```
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 [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 Back-propagation1

```
Input:
    Training Set $x^{(1)}, y^{(1)}, , ...., (x^{(m)}, y^{(m)})$
Output:
    Gradient of the cost function
1: $\Delta_{ij}^{(l)} = 0$(for all l,i,j)
2: for $i = 1$ to $m$ do
3:     Set $a^{(1)} = x^{(i)}$
4:     Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3, ....L$
5:     Using $y^(i)$, compute $\delta^(L) = a^(L) - y^(i)$
6:     Compute $\delta^(L-1), \delta^(L-2), ....\delta^(2)$
7:     $\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$
8: end for
9: $D_{ij}^{(l)} := \frac{1}{m}\Delta_{ij} + \lambda\theta_{ij}^{(l)}$ if $j \neq 0$
10: $D_{ij}^{(l)} := \frac{1}{m}\Delta_{ij}$ if $j = 0$
```

Algorithm 9 Learning Vector Quantization [Brownlee, 2015c]

```
Input: $ProblemSize, InputPatterns, iterations_{max}, CodebookVectors_{num}, learn_{rate}$
Output: $CodebookVectors$
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 10 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:     $Learn^{i}_{rate} \leftarrow CalculateLearningRate(i, learn^{init}_{rate})$
 4:     $neighborhood^{i}_{size} \leftarrow CalculateNeighborhoodSize(i, neighborhood^{size}_{init})$
 5:     $Pattern_i \leftarrow SelectInputPattern(InputPatterns)$
 6:     $Bmu_i \leftarrow SelectBestMatchingUnit(Pattern_i, CodebookVectors)$
 7:     $Neighborhood \leftarrow Bmu_i$
 8:     $Neighborhood \leftarrow SelectNeighbors(Bmu_i, CodebookVectors, neighborhood^{i}_{size})$
 9:     for $Vector_i \in Neighborhood$ do
10:         for $Vector^{attribute}_{i} \in Vector_i$ do
11:             $Vector^{attribute}_{i} \leftarrow Vector^{attribute}_{i} + learn_{rate} \times (Pattern^{attribute}_{i} - Vector^{attribute}_{i})$
12:         end for
13:     end for
14: end for
15: 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
 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:     $C = (C \backslash \{C, C'\}) \cup \{C \cup C'\}$                   ▷ 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 $T$
```

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
 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 $T$
```

Algorithm 13 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 $a_{best} = argmax\{GainRatio(a,T)\}$
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 14 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 15 Naive Bayes

```
Input:
    $C$ : A fixed set of classes
    $D$ : Documents
Output: Category(Class) of the Documents
 1: function TRAINMULTINOMIALNB($C, D$)
 2:     $V \leftarrow EXTRACTVOCABULARY(D)$
 3:     $N \leftarrow COUNTDOCS(D)$
 4:     for each $c \in C$ do
 5:         $N_c \leftarrow COUNTDOCSINCLASS(D, c)$
 6:         $prior|c| \leftarrow N_c/N$
 7:         $text_c \leftarrow CONCATENATETEXTOFALLDOCSINCLASS(D, C)$
 8:         for each $t \in V$ do
 9:             $condprob|t||c| \leftarrow \frac{T_{ct}+1}{\sum_{t'}(T_{ct'+1})}$
10:         end for
11:     end for
12:     return $V, prior, condprob$
13: end function
14: function APPLYMULTINOMIALNB($C, D, prior, condprob, d$)
15:     $W \leftarrow EXTRACTTOKENSFROMDOC(V, d)$
16:     for each $c \in C$ do
17:         $score|c| \leftarrow \log\ prior|c|$
18:         for each $t \in W$ do
19:             $score|c| + = \log condprob|t||c|$
20:         end for
21:     end for
22:     return $arg\ max_{c \in C} score|c|$
23: end function
```

Algorithm 16 Lasso Regression

```
Input:
    $ipy$ : Inner product vector, $ipy_i =< y, X_{\cdot i} >$
    $ipx$ : Inner product matrix, $ipx_{ij} =< X_{\cdot i}, X_{\cdot j} >$
    $\lambda$ : Penalty parameter
    $N$ : Number of samples
Output: $beta$ : Regression parameter vector
 1: function FASTLASSO($ipy, ipx, \lambda, N$)
 2:     stop_thr                                  ▷ Threshold for stopping iteration
 3:     $p \leftarrow length(ipy)$
 4:     $beta \leftarrow 0$ with length $p$
 5:     $gc \leftarrow 0$ with length $p$
 6:     while $difBeta_{max} \geq$ stop_thr do
 7:         $difBeta_{max} \leftarrow 0$
 8:         for $j = 1 \leftarrow p$ do
 9:             $z \leftarrow (ipy|j| - gc|j|)/N + beta|j|$
10:             beta_tmp $\leftarrow max(0, z - \lambda) - max(0, -z - \lambda)$
11:             $difBeta \leftarrow$ beta_tmp $- beta|j|$
12:             $difabs \leftarrow abs(difBeta)$
13:             if $difabs > 0$ then
14:                 $beta|j| \leftarrow$ beta_tmp
15:                 $gc \leftarrow gc + ipx|j| \times difBeta$
16:                 $difBeta_{max} = max(difBeta_{max}, difabs)$
17:             end if
18:         end for
19:     end while
20: end function
```

Algorithm 17 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:         $examples_i \leftarrow$ a bootstrap sample of $examples$
 4:     end for
 5:     Compute set $I$ containing indices for the $k$ smallest distances $d(X_i, x)$
 6:     $h_i \leftarrow$ apply$L to examples_i$
 7:     Return $h_1, h_2, ...h_B$
 8: 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
 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 $y_j =$
```

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

```
 9:         Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to the following equation
10:
```

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

```
11:     end for
12: end for
```

Algorithm 19 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.

[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
    $\epsilon$: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 $P$in 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($P, \epsilon$)
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
    $\sigma^2$noise 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: $cov(f_*) = k(x_*, x_*) - k_*^T [K + \sigma_2 \mathbb{I}n]^{-1} k_*$
```

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 Simplest 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: $\mathbf{w^T x + b} = 0$

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. Solving for λ :
$\mathbf{w^T x}_2 + b = 1$ where $\mathbf{x}_2 = \mathbf{x}_1 + \lambda\mathbf{w}$
$\mathbf{w^T}(\mathbf{x}_1\lambda\mathbf{w}) + b = 1$
$\mathbf{w^T x}_1 + b + \lambda\mathbf{w^T w} = 1$ 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 $\mathbf{x}_i)$