Decision Tree:

$$H(V) = \sum_{V} -P(H=V)\log_2 P(H=V).$$

$$Gain(S,A) = H(S) - \sum_{v\in Value(A)} P(V)\cdot H(V)$$

Point Estimation.

$$\hat{\theta}_{MLE} = \frac{\alpha_H}{\alpha_H+\alpha_T}. \quad \hat{\theta}_{MAP} = \frac{\alpha_H+\beta_H-1}{\alpha_H+\beta_H+\alpha_T+\beta_T-2}$$

Naive Bayes:

$$P(Y=y_i|X=X_k) = \frac{P(X=X_k|Y=y_i)\cdot P(Y=y_i)}{\sum_j P(X=X_k|Y=y_j)\cdot P(Y=y_j)}$$

Logistic Regression.

$$P(Y=1|X) = \frac{1}{1+\exp(w_0+\sum_i^n w_i X_i)}.$$

Perception. Training Rule:

$$\begin{cases} \Delta w_i = \eta(t-o)\cdot x_i \\ w_i = w_i + \Delta w_i. \end{cases}$$

To guarantee convergence; use gradient descent:

minimize $E[\vec{w}] = \frac{1}{2}\sum_{d\in D}(t_d - o_d)^2$

$\nabla E[\vec{w}] = [\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots, \frac{\partial E}{\partial w_n}]$.

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i}\cdot\frac{1}{2}\sum_{d\in D}(t_d-o_d)^2$$

$$= \sum_{d\in D}(t_d-o_d)(-x_{i,d})$$

$\vec{\Delta w} = -\eta\cdot\nabla E[\vec{w}]$. therefore,

$$\Delta w_i = -\eta\cdot\sum_{d\in D}(t_d-o_d)(-x_{i,d})$$

$$= \eta\cdot\sum_{d\in D}(t_d-o_d)\cdot x_{i,d}.$$

```
Batch_Gradient_descent(D; η):
1. $\vec{w} = \langle w_0, w_1, \cdots, w_n\rangle \leftarrow$ small random number.
2. Until convergence do.
      i). $\vec{\Delta w} = \langle \Delta w_0, \cdots, \Delta w_n\rangle \leftarrow 0$
      ii). for $d\in D$, where $d = \langle \vec{x}, y\rangle$, do
              $o_d = \vec{w}^T\cdot\vec{x}$
              for $\Delta w_i \in \vec{\Delta w}$, do
                    $\Delta w_i \leftarrow \Delta w_i + \eta(y-o_d)\cdot x_i$
      iii). for $w_i \in \vec{w}$, do
              $w_i \leftarrow w_i + \Delta w_i$
3. return $\vec{w}$.
```

```
Incremental_Gradient_Descent(D, η)
1. $\vec{w} = \langle w_0, w_1, \cdots, w_n\rangle \leftarrow$ small random value
2. Until meet the termination condition do
      for $d\in D$ do
         i) $o_d = \vec{w}^T\cdot\vec{x}$.
         for $\Delta w_i \in \vec{\Delta w}$ do
               $\Delta w_i = \eta\cdot x_i\cdot(y-o_d)$
         for $w_i \in \vec{w}$, do
               $w_i = w_i + \Delta w_i$
3. Return $\vec{w}$.
```

If $o_d = \text{Sigmoid}(\vec{w}\cdot\vec{x}) = \sigma(\vec{w}\cdot\vec{x}) = \frac{1}{1+e^{-\vec{w}\cdot\vec{x}}}$

then $\frac{d\sigma(z)}{dz} = \sigma(z)\cdot(1-\sigma(z))$

Then $\Delta w_i = -\eta\cdot\frac{\partial E}{\partial w_i} = \eta(t_d-o_d)\cdot o_d\cdot(1-o_d)\cdot x_i$.

Neural Network.
$$\begin{tikzcd} \underset{i}{O} \arrow[r, "w_{ji}"] & \underset{j/h}{O} \arrow[r, "w_{kh}"] & \underset{k}{O} \end{tikzcd}$$

```
Backpropagation(D, η, n_in, n_out, n_hidden)
1. Create a feed-forward network
2. initialize all network weights to small random number
3. Until the termination condition is met, do
      for $d\in D$ do
         i) compute output $O_u$ of every unit.
         ii). for each output unit k, its error term
               $\delta_k = \overset{f'}{\underline{O_k(1-O_k)}}(t_k-O_k)$
         iii). for each hidden unit h,
               $\delta_h = \overset{f'}{\underline{O_h(1-O_h)}}\sum_{k\in outputs} w_{k,h}\cdot\delta_k$
         iii)
               $w_{ji} = w_{ji} + \eta\cdot\delta_j\cdot x_{ji}$
```

Instance Based Learning.

N points in D dimensions, to apply k-newest neighbor algorithm. the distance must go is: $\left(\frac{k}{N}\right)^{\frac{1}{D}}$

SVM:

Lagrangian function.

i). Primal form: minimize

$$L_P(\vec{w},b,\alpha_i)=\frac{1}{2}\|w\|^2-\sum_{i=1}^{n}\alpha_i\cdot(y_i\cdot(\vec{w_i}\cdot x_i+b)-1)$$

such that $\alpha_i \geq 0$.

KKT conditions:

① $\frac{\partial L_P}{\partial w}=0 \Rightarrow \vec{w}=\sum_{i=1}^{n}\alpha_i\cdot y_i\cdot \vec{x_i}$

② $\frac{\partial L_P}{\partial b}=0 \Rightarrow \sum_{i=1}^{n}\alpha_i\cdot y_i=0$

③ $\alpha_i\cdot(y_i\cdot(\vec{w}\cdot\vec{x_i}+b)-1)=0$.

④ $\alpha_i \geq 0$.

only support vectors have $\alpha_i \neq 0$, then get b from ③

ii). Lagrangian Dual Problem.

maximize $\sum_{i=1}^{n}\alpha_i-\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\cdot\alpha_j\cdot y_i\cdot y_j\cdot\vec{x_i}^T\cdot\vec{x_j}$

such that $\alpha_i \geq 0$, $\sum_{i=1}^{n}\alpha_i\cdot y_i=0$.

Kernel function $K(x_i,x_j)=$?

After solve α_i, given a new test data x,

the prediction $g(x)=\sum_{i\in SV}\alpha_i\cdot K(x_i,x)+b=\begin{cases}1, & +\\ -1, & -\end{cases}$.

Ensemble methods (Bagging and boosting).

AdaBoost:

1. initialize the data $\{w_n\}$: $w_n^{(1)}=\frac{1}{N}$.
2. For $m=1,\cdots,M$.

i). minimize the weighted error function.

$$J_m=\sum_{n=1}^{N}w_n^{(m)}\cdot I(y_m(x_n)\neq t_n)$$

ii).

$$\varepsilon_m=\frac{J_m}{\sum_{n=1}^{N}w_n^{(m)}},\quad \alpha_m=\ln\left\{\frac{1-\varepsilon_m}{\varepsilon_m}\right\}$$

iii). $w_n^{(m+1)}=w_n^{(m)}\cdot e^{\{\alpha_m\cdot I(y_m(x_n)\neq t_n)\}}$

3. $Y_M(x)=\text{sign}\left(\sum_{m}^{M}\alpha_m\cdot y_m(x)\right)$

Bayesian Network.

Variable elimination: Process nodes in order, then eliminate = connect all children of a node to each other.

Learning theory:

$$m \geq \frac{1}{\varepsilon}\left(\ln\left(\frac{1}{\delta}\right)+\ln|H|\right)$$

$$m \geq \frac{1}{2\varepsilon^2}\left(\ln\left(\frac{1}{\delta}\right)+\ln|H|\right)$$

$$m \geq \frac{1}{\varepsilon}\left(4\log_2\left(\frac{2}{\delta}\right)+8\,VC(H)\cdot\log_2\left(\frac{13}{\varepsilon}\right)\right)$$

The VC-dimension is always less than the size of hypothesis since $|H| \geq 2^m$, to shatter m instances, $VC(H) \leq \log_2(H)$.

At each iteration, EM improves its cost function and is guaranteed to converge to a local minima.

Guassian Naive Bayes is a linear classifier only when the variances are assumed to be independent of the class.

No free lunch theorem.

Gaussian naive Bayes classifier has more expressive power than a linear classifier if we don't restrict the variance parameters to be class independent.

Add the constraint that $0 \leq \alpha_i \leq C$, SVM.

Slower than decision tree: Bagging, boosting.

Bagging can be parallelized easily, but hurt the performance of stable clas

Bagging and boosting attempt to reduce variance.

F. if the error $\varepsilon > 0.5$, then the weight assigned to the misclassifi points will be smaller than the weight assigned to the points that are classified correctly, then the subsequent iterations will not try to classify these points correctly and thus the algorithm is likely to exhibit poor performance.

Adaboost is susceptible to outliers, a possible heuristic is to put a threshold on the weight and remove all points that have very large weights.

KNN: as we increase k, the algorithm underfits because the decision surface becomes simpler.

Perceptron training rule: (also for linear regression,

$$\begin{cases} \Delta w_i = \frac{\partial E}{\partial w_i} & \text{, where } E(w) = \frac{1}{2}\sum_{d\in D}(t_d - o_d)^2 \\ w_i \leftarrow w_i + \eta \cdot \Delta w_i \end{cases}$$

① Batch gradient descent algorithm:

```
initialize $\vec{w} = \langle w_0, w_1, \cdots, w_n \rangle$ ← small random value
Until convergence, do
    initialize $\Delta\vec{w} = \langle \Delta w_0, \Delta w_1, \cdots, \Delta w_n \rangle \leftarrow 0$
    for $d \in D$, do
        $o_d = \vec{w}^T \cdot \vec{x}$
        for $i=0$ to $n$, do
            $\Delta w_i \leftarrow \Delta w_i + \frac{\partial E}{\partial w_i}$
    for $i=0$ to $n$, do
        $w_i \leftarrow w_i + \eta \cdot \Delta w_i$
return $\vec{w}$
```

Decision Tree:

$H(V) = \sum - p(v) \cdot \log p(v)$

$\text{Gain}(S,A) = H(S) - \sum_{v \in \text{Value}(A)} p(v) \cdot H(S_v)$

Given $X = (x_1, \cdots, x_d)$, the bound on the number of leaf nodes is $O(2^d)$.

$$\frac{d\sigma(x)}{dx} = \sigma(x) \cdot (1 - \sigma(x))$$

Support Vector Machine

$L_p(\vec{w}, b, \alpha_i) = \frac{1}{2}||w||^2 - \sum_{i=1}^{n} \alpha_i \cdot (y_i(\vec{w}^T \cdot \vec{x}_i + b) - 1), \alpha_i \geq 0$

$g(x) = \vec{w}^T \cdot x + b = \sum_{i \in SV} \alpha_i \cdot x_i^T \cdot x + b$

Lagrangian dual problem: maximize $\sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \cdot \alpha_j \cdot y_i \cdot y_j \cdot x_i^T \cdot x_j$

with slack penalty. such that $C \geq \alpha_i \geq 0$, and $\sum_{i=1}^{n} \alpha_i \cdot y_i = 0$

all α_i should be bounded, if the optimal α are greater than C, we may end up with a sub-optimal decision boundary.

As data points $N \to \infty$, prior $p(\theta)$ is forgotten, and MLE = MAP.

Point Estimator:

i). MLE: $P(D|\theta)$, $\hat{\theta} = \arg\max_\theta \ln P(D|\theta) = \frac{d}{d\theta} \ln P(D|\theta) = 0$.

ii). MAP: $P(\theta|D)$, $\hat{\theta} = \arg\max_\theta P(\theta|D) = \frac{\alpha_H + \beta_H - 1}{\alpha_H + \beta_H + \alpha_T + \beta_T - 2}$

We use MAP in naive Bayes.

$\text{Beta}(\beta_H, \beta_T)$

Naive Bayes: $P(Y = y_i | X = x_k) = \frac{P(X = x_k | Y = y_i) \cdot P(Y = y_i)}{\sum_j P(X = x_k | Y = y_j) \cdot P(Y = y_j)}$

if we assume the variance is independent of the class, then naive Bayes is a linear classifier.

$$= \frac{P(X_1 \cdots X_n | Y = y_i) \cdot P(Y = y_i)}{\sum_j \prod_i P(X_i | Y = y_i) \cdot P(Y = y_i)}$$

① Gaussian naive Bayes: estimate the parameters: $P(Y = y_k)$, $u_{ik} = E(X_i | Y = y_k)$, $\hat{\sigma}_{ik}^2 = E[(X_i - u_{ik})^2 | y_i]$

② Discrete-valued Input: $P(X_i | Y = y_k)$. $P(Y = y_k)$.

Neural Network · ↑ the error surface may contain many different local minima, therefore, only guaranteed to converge toward some local minima, not global minima.

Backpropagation:

$$E_d(\vec{w}) = \frac{1}{2}(t_d - O_d)^2$$

```
1. initialize all network weights to small random number.
2. Until convergence do
       for each <x, y> in training-example do.
           i). input the instance $\bar{x}$ to the network, and compute the output $O_u$ of every
               unit u in the network.
           ii). for each network $\boxed{\text{output}}$ unit k, calculate its error term $\delta_k$.
                    $\delta_k \leftarrow O_k(1-O_k)(t_k - O_k)$        // $\frac{\partial E_d}{\partial net_j} = \frac{\partial E_d}{\partial O_j}\cdot\frac{\partial O_j}{\partial net_j}$
           iii). for each hidden unit h, calculate its error term $\delta_h$.
                    $\delta_h \leftarrow O_h(1-O_h)\underbrace{\sum_{k\in outputs} W_{k,h}\delta_k}$   output of h.
           iv). Update each network weight $w_i$
                    $w_i \leftarrow w_i + \eta\cdot\underset{\downarrow\ \text{target}}{\delta_j}\cdot\underset{\text{input}}{\underline{x_{ji}}}$
```

$$\delta_k = -\frac{\partial E_d}{\partial net_k}$$

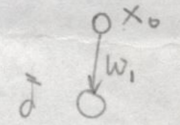
$$\uparrow$$

$$\frac{\partial E_d}{\partial net_j} = \sum_{k\in outputs} \frac{\partial E_d}{\partial net_k}\cdot\frac{\partial net_k}{\partial net_j}$$

$$= \sum_{k\in outputs} -\delta_k\cdot\frac{\partial net_k}{\partial O_j}\cdot\frac{\partial O_j}{\partial net_j}$$

$$= \sum_{k\in outputs} -O_j(1-O_j)\cdot W_{kj}\cdot\delta_k$$

$$\Delta w_{ji} = -\eta\cdot\frac{\partial E_d}{\partial w_{ji}}$$



$$w_1 \leftarrow w_1 + \eta\cdot\delta\cdot x_0$$

Logistic Regression MCAP. L2 algorithm: the objective function is always concave, and therefore the algorithm is always converge under mild assumption.

```
initialize $w = \langle w_0, w_1, \cdots, w_n\rangle$.
until convergence do
    i) for each data $d \in D$.
           prob[d] = sigmoid($W^T\cdot x$)
    ii). intialize $dw = \langle dw_0, dw_1, \cdots, dw_n\rangle$ to 0.
    iii). for i=0 to n
              for j=0 to m
                  $dw[i] = dw[i] + \underbrace{x_{ji}\cdot(Y_j - Prob[j])} = \frac{\partial Loss(w)}{\partial w_i}$
    iv). for i=0 to n
              $w[i] = w[i] + \eta(dw[i] - \lambda\cdot w[i])$
```

The time complexity of k-means is $O(nkdi)$, n is the number of data points, d is the number of features, k is the number of clusters, i is iterations.

<u>Agglomerative clustering</u>: 1). Maintain a set of clusters, initially, each instance in its own cluster.
2). Repeat: i) pick the two closest clusters, merge them into a new cluster,
ii) stop when there is only one cluster left.

produce not one clustering, but a dendrogram tree.

VC dimension of hyperplane in d-dimensional space is $d+1$.

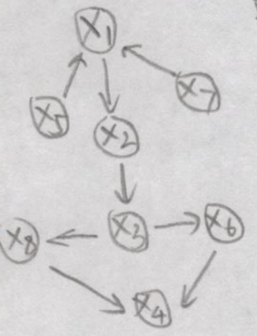
The VC dimension of all axis parallel rectangles is ≥ 4, no 5 instances can be shattered since there can be at most 4 distinct extreme points and these 4 points cannot be included without including any possible 5th point.

no point
1 point
2 point
3 point
4 point

$2^4 = 16$

$VC(H) = 2n$, n is dimension of axis-parallel hyper-rectangle.

Hypothesis space $|H| = n^m$, n is the possible range of variable, m is the number of variables.



Elimination: $x_1, x_2, x_3, x_5, x_6, x_7, x_8$. to compute $P(X_4 = x_4)$.

x_1, $\phi(x_7, x_5, x_2) = \sum_{x_1} P(x_1 \mid x_7, x_5) \cdot P(x_2 \mid x_1)$ exp(4)

x_2, $\phi(x_5, x_7, x_3) = \sum_{x_2} \phi(x_7, x_5, x_2) \cdot P(x_3 \mid x_2)$, exp(4).

x_3, $\phi(x_5, x_7, x_6, x_8) = \sum_{x_3} \phi(x_5, x_7, x_3) \cdot P(x_6 \mid x_3) \cdot P(x_8 \mid x_3)$ exp(5)

x_5, $\phi(x_7, x_6, x_8) = \sum_{x_5} \cdot \phi(x_5, x_7, x_6, x_8) \cdot P(x_5)$ exp(4).

x_6, $\phi(x_7, x_8) = \sum_{x_6} \phi(x_7, x_6, x_8) \cdot P(X_4 = x_4 \mid x_6, x_8)$ exp(3)

x_7, $\phi(x_8) = \sum_{x_7} \cdot \phi(x_7, x_8) P(x_7)$ exp(2)

x_8, $P(X_4 = x_4) = \sum_{x_8} \phi(x_8)$ exp(1).

Time complexity $= O(7 \cdot \exp(5))$, Space complexity: $O(7 \cdot \exp(4))$.
size of the largest function created.

X_7, $\phi(X_8) = \sum_{X_7} \cdot \phi(X_7, X_8)\, p(X_7)$ exp(2)

X_8, $p(X_4 = x_4) = \sum_{X_8} \phi(X_8)$ exp(1).

Time complexity = $O(7 \cdot \exp(5))$, $\underbrace{\text{Space}}_{\text{size of the largest function created.}}$ complexity : $O(7 \cdot \exp(4))$.

BN 1 : have no-edge, BN2 : have an edge,

x_1 and x_2 independent $\searrow$ x_1 and x_2 not independent.

Assume no noisy, then 1-NN is 100% true because the class of each example is the class of its nearest neighbor, that is itself.

Some functions, e.g. the parity function requires exponential number of hidden nodes, if only one hidden layer is used.

Each node in the decision tree divides the examples into two non-empty sets, n training examples; then the number of nodes from the leaf to the root is bounded by "n".