A **training set** D is a set of n data points used to train a machine learning algorithm.

Each **data point** $\vec{x_i}$ where $i \in \{1, 2, \ldots, n\}$ has k features so

$$\vec{x_i} = \langle x_{i,1},\ x_{i,2},\ \ldots,\ x_{i,k} \rangle$$

In **supervised classification**, each data point has a label y that notes which class it belongs to.

A **hypothesis** h is the output of a machine learning algorithm. It maps a feature vector $\vec{x}$ to an unknown label y. The **target function** is a hypothesis that correctly labels all unseen data points.

A **parameter vector** $\vec{\theta}$ is a k long vector (k is number of features) that gives the likelihood each feature value belongs to a certain label.

Decision Tree

A **decision tree** classifies a data point by analyzing each feature individually.

A data set D can be partitioned by selecting a feature

The **entropy** $H(D)$ of a data set is

$$H(D) = -p_+ \cdot \log_2 p_+ - p_- \cdot \log_2 p_-$$

where p_+ is the number of positively labeled examples and p_- is the number of negatively labeled examples

Information gain $Gain(D, f)$ for selecting a feature f from the data set is the difference in original entropy $H(D)$ of the data set and after the data set is partitioned by that feature

$$Gain(D, f) = H(D) - \sum_{\text{all values of } f} \frac{|D_f|}{|D|} \cdot H(D_f)$$

To **build a decision tree**, find a feature with highest information gain. Split the data set using that feature and repeat.

Maximum Likelihood Estimate

The **likelihood function** $L(D|\vec{\theta})$ of a data set $D = \{\vec{x_1}, \vec{x_2}, \ldots, \vec{x_n}\}$ given parameters $\vec{\theta}$ is

$$L(D|\vec{\theta}) = \prod_{i=1}^{n} \mathbf{P}\left(\vec{x_i}|\vec{\theta}\right)$$

The **log likelihood** $\ell\left(D|\vec{\theta}\right)$ of the same data set is

$$\ell\left(D|\vec{\theta}\right) = \sum_{i=1}^{n} \log \mathbf{P}\left(\vec{x_i}|\vec{\theta}\right)$$

The **maximum likelihood estimate (MLE)** $\hat{\theta}_{MLE}$ finds the parameter vector $\vec{\theta}$ that maximizes the likelihood of the data set D

$$\hat{\theta}_{MLE} = \arg\max_{\vec{\theta}} L\left(D|\vec{\theta}\right) = \arg\max_{\vec{\theta}} \ell\left(D|\vec{\theta}\right) = \arg\max_{\vec{\theta}} \sum_{i=1}^{n} \log \mathbf{P}\left(\vec{x_i}|\vec{\theta}\right)$$

Maximum a Posteriori

The **posterior** $P(\vec{\theta}|D)$ is the probability of getting a parameter vector $\vec{\theta}$ given the data D

$$\mathbf{P}\left(\vec{\theta}|D\right) = \frac{\mathbf{P}\left(D|\vec{\theta}\right) \cdot \mathbf{P}\left(\vec{\theta}\right)}{\mathbf{P}\left(D\right)} \propto \mathbf{P}\left(D|\vec{\theta}\right) \cdot \mathbf{P}\left(\vec{\theta}\right)$$

The **prior** $P(\vec{\theta})$ is the probability of getting an exact vector of parameters $\vec{\theta}$

The **maximum a posteriori (MAP) estimate** $\hat{\theta}_{MAP}$ finds the parameter vector that is most likely for the data set $D = \{\vec{x_1}, \vec{x_2}, \ldots, \vec{x_n}\}$

$$\begin{aligned}\hat{\theta}_{MAP} &= \arg\max_{\vec{\theta}} \prod_{i=1}^{n} \mathbf{P}\left(\vec{x_i}|\vec{\theta}\right) \cdot \mathbf{P}\left(\vec{\theta}\right) \\ &= \arg\max_{\vec{\theta}} \sum_{i=1}^{n} \left(\log \mathbf{P}\left(\vec{x_i}|\vec{\theta}\right) + \log \mathbf{P}\left(\vec{\theta}\right)\right)\end{aligned}$$

To **evaluate argmax**, take the derivative, set the value to zero, and solve for $\vec{\theta}$.

Naive Bayes

The **naive bayes assumption** is that all k features are conditionally independent of one another given the label y. For a single data point $\vec{x}$

$$\mathbf{P}\left(\vec{x}, y\right) = \mathbf{P}\left(x_1, x_2, \ldots, x_k|y\right) \cdot \mathbf{P}\left(y\right) = \prod_{j=1}^{k} \mathbf{P}\left(x_j|y\right) \cdot \mathbf{P}\left(y\right)$$

To **train** a naive bayes classifier with data points that have feature values $X_j \sim Bernoulli(\theta_{j,Y})$ for features $j \in \{1, 2, \ldots, k\}$ and label values $Y \sim Bernoulli(\phi)$, we find the parameters for each of these distributions that maximize the probability of seeing a label y, given the data $\vec{x}$. Find **MAP** estimator $\theta_{j,y}$ for each $j \in \{1, 2, \ldots, k\}$ and $y_i \in \{0, 1\}$

$$\begin{aligned}\theta_{j,y} &= \mathbf{P}\left(X_j = 1|Y = y\right) \text{ where } y \in \{0, 1\} \\ 1 - \theta_{j,y} &= \mathbf{P}\left(X_j = 0|Y = y\right) \\ \vec{\theta}_y &= \langle \mathbf{P}\left(X_1 = 1|Y = y\right), \mathbf{P}\left(X_2 = 1|Y = y\right), \ldots, \mathbf{P}\left(X_k = 1|Y = y\right)\rangle\end{aligned}$$

where for each training data point $D = \{\vec{x_1}, \vec{x_2}, \ldots, \vec{x_n}\}$

$$\mathbf{P}\left(X_j = 1 | Y = y\right) = \frac{\sum_{i=1}^{n} x_{i,j} \cdot y_i}{\sum_{i=1}^{n} x_{i,j}} \text{ where } x_{i,j} \in \{0, 1\} \text{ and } y_i \in \{0, 1\}$$

or with using a β prior to do **smoothing**

$$\mathbf{P}\left(X_j = 1 | Y = y\right) = \frac{(\alpha - 1) + \sum_{i=1}^{n} x_{i,j} \cdot y_i}{(\alpha - 1) + (\beta - 1) + \sum_{i=1}^{n} x_{i,j}} \text{ where } \alpha, \beta \geq 1$$

To **classify** find the probability a new data point $\vec{x}$ has label $y = 1$ and the probability it has label $y = 0$. Take the maximum of these to label $\hat{y}$ a new data point

$$\begin{aligned}
\hat{y} &= \arg\max_{y \in \{0,1\}} \mathbf{P}\left(Y = y | X = \vec{x}\right) \\
&= \arg\max_{y \in \{0,1\}} \mathbf{P}\left(X = \vec{x} | Y = y\right) \cdot \mathbf{P}\left(Y = y\right) \\
&= \arg\max_{y \in \{0,1\}} \mathbf{P}\left(Y = y\right) \cdot \prod_{j=1}^{k} \mathbf{P}\left(X_j = x_j | Y = y\right) \text{ make Naive Bayes assumption} \\
&= \arg\max_{y \in \{0,1\}} \log \mathbf{P}\left(Y = y\right) \cdot \sum_{j=1}^{k} \log \mathbf{P}\left(X_j = x_j | Y = y\right) \\
&= \arg\max_{y \in \{0,1\}} \log \mathbf{P}\left(Y = y\right) \cdot \sum_{j=1}^{k} \left(\log \theta_{j,y} \cdot x_j + \log\left(1 - \theta_{j,y}\right) \cdot (1 - x_j)\right)
\end{aligned}$$

Logistic Regression (is a classifier)

The **logistic function** is

$$logistic(x) = \frac{1}{1 + e^{-x}}$$

The **loss function** is the probability of a label y given the data $\vec{x}$

$$\begin{aligned}
\mathbf{P}\left(Y = 1 | X = \vec{x}\right) &= \frac{1}{1 + e^{-\vec{w} \cdot \vec{x}}} \\
\mathbf{P}\left(Y = 0 | X = \vec{x}\right) &= 1 - \frac{1}{1 + e^{-\vec{w} \cdot \vec{x}}}
\end{aligned}$$

The goal of logistic regression is to find parameters $\vec{w}$ that maximize the conditional log likelihood of the data.

The **likelihood** $L(D|\vec{w})$ of data $D = (\vec{x_1}, y_1), (\vec{x_2}, y_2), \ldots, (\vec{x_n}, y_n)$

$$L(D|\vec{w}) = \prod_{i=1}^{n} \mathbf{P}\left(y_i | \vec{x_i}, \vec{w}\right) = \prod_{i=1}^{n} \left(\frac{e^{-\vec{w} \cdot \vec{x_i}}}{1 + e^{-\vec{w} \cdot \vec{x_i}}}\right)^{y_i} \cdot \left(\frac{1}{1 + e^{-\vec{w} \cdot \vec{x_i}}}\right)^{1 - y_i} = \prod_{i=1}^{n} \frac{e^{-y_i \cdot \vec{w} \cdot \vec{x_i}}}{1 + e^{-\vec{w} \cdot \vec{x_i}}}$$

The **conditional log likelihood** $\ell(D|\vec{w})$ of the data is

$$\ell(D|\vec{w}) = \log L(D|\vec{w}) = \sum_{i=1}^{n} \left(-y_i \cdot \vec{w} \cdot \vec{x_i} - \log\left(1 + e^{-\vec{w} \cdot \vec{x_i}}\right)\right)$$

To **maximize likelihood**, take the derivative of the log likelihood with respect to $\vec{w}$, set that equal to $\vec{0}$, and solve for $\vec{w}$

$$\frac{d}{d\vec{w}} \ell(D|\vec{w}) = \vec{0}$$

If no closed form solution, use **stochastic gradient descent** to find $\arg\min_{\vec{w}} \ell(D|\vec{w})$

(a) Define log conditional likelihood $\ell(D|\vec{w}) = \sum_{i=1}^{b} \log \mathbf{P}\left(y_i|\vec{w} \cdot \vec{x_i}\right)$

(b) Start with $\vec{w} = \vec{w_0}$

(c) update $\vec{w_{t+1}} = \vec{w_t} + \lambda \cdot \frac{d}{d\vec{w}} \ell(D|\vec{w})$ for $t \in \{1, 2, \ldots, \infty\}$ until convergence

where λ is very small

Linear Regression

To solve the equation with input data $\vec{x}$ and label y

$$y = \vec{w} \cdot \vec{x} + \epsilon$$

we minimize the **least mean square** to minimize the distance between the data points and predicted line.

$$\hat{\theta} = \arg\min_{\vec{\theta}} J(\vec{w}) = \arg\min_{\vec{\theta}} \frac{1}{2} \cdot \sum_{i=1}^{n} \left(\vec{\theta} \cdot \vec{x_i} - y_i\right)^2$$

The **closed form solution** using data $\mathbf{X}$ is

$$\hat{w} = \left(\mathbf{X}^T \cdot \mathbf{X}\right)^{-1} \cdot \mathbf{X}^T \cdot \vec{y}$$

The iterative **gradient descent** involves updating $\vec{w}$ until convergence

$$\vec{w}_{t+1} = \vec{w}_t + \lambda \cdot \vec{\bigtriangledown}_{\vec{w}} \cdot J(\vec{w})$$

To add **regularization** (a penalty) to reduce overfitting

(a) L1 regularization or LASSO: $J_{LASSO}(\vec{w}) = J(\vec{w}) + \gamma \cdot |\vec{w}|_1$

(b) L2 regularization or Ridge Regression: $J_{Ridge}(\vec{w}) = J(\vec{w}) + \gamma \cdot |\vec{w}|_2$

Perceptron

The **perceptron** algorithm learns a linear decision boundary to classify data points into two classes. Use the following rules

(a) $\vec{w}$ is a **weight vector** of how much each misclassified $\vec{x}$ counts toward each feature

(b) use $\vec{w} \cdot \vec{x}$ to predict label y of data point $\vec{x}$

Run the algorithm

(a) if $\vec{w} \cdot \vec{x} > 0$ misclassifies training data $\vec{x}$, update $\vec{w}_{t+1} = \vec{w}_t + \vec{x}$

(b) if $\vec{w} \cdot \vec{x} < 0$ misclassifies training data $\vec{x}$, update $\vec{w}_{t+1} = \vec{w}_t - \vec{x}$

(c) if $\vec{x}$ classified correctly, do not update $\vec{w}$

A **decision plane** is a hyperplane described by $\vec{w}$ to classify data points (can be thought of as a line in two dimensions)

A **margin** $\gamma_{\vec{w}}$ is the minimum distance from all points to the hyperplane $\vec{w}$

The **margin** γ is the maximum $\gamma_{\vec{w}}$ for all $\vec{w_1}, \vec{w_2}, \ldots, \vec{w_t}$

The **number of mistakes** is bounded by

$$\text{number of mistakes} \leq \left(\frac{R}{\gamma}\right)^2$$

where R is the radius (distance from the origin to the furthest data point)

Kernels

A **kernel** $K(\vec{x}, \vec{z})$ replaces a dot product $\vec{x} \cdot \vec{z}$ to act as if the data was in a higher dimensional space.

example: $K(\vec{x}, \vec{y}) = (\vec{x} \cdot \vec{y} + 1)^d$ maps n dimensions to n^d dimensions

the formal definition says

$$K(\vec{x}, \vec{z}) = \phi(\vec{x}) \cdot \phi(\vec{z})$$

if $\phi(\vec{x})$ is the map function that maps $\vec{x}$ to a higher dimensional space

example: $\vec{x} = \langle x_1, x_2 \rangle$ then $\phi(\vec{x}) = \left\langle x_1^2, x_2^2, \sqrt{2} \cdot x_1 \cdot x_2 \right\rangle$

Below are some kernels

linear	$K(\vec{x}, \vec{z}) = \vec{x} \cdot \vec{z}$
polynomial	$K(\vec{x}, \vec{z}) = (\vec{x} \cdot \vec{z})^d$
gaussian	$K(\vec{x}, \vec{z}) = e^{-\frac{\lvert\vec{x}-\vec{z}\rvert^2}{2 \cdot \sigma^2}}$
laplace	$K(\vec{x}, \vec{z}) = e^{-\frac{\lvert\vec{x}-\vec{z}\rvert}{2 \cdot \sigma^2}}$

Support Vector Machines (SVMs)

To use **SVMs**, maximize the margin γ to hyperplane $\vec{w} \cdot \vec{x} = 0$ to better predict test data

We want to maximize γ under constraints

(a) $|\vec{w}| = 1$

(b) $y_i \cdot \vec{w} \cdot \vec{x_i} > \gamma$ for $i \in \{1, 2, \ldots, n\}$

by changing $\vec{w} \rightarrow \vec{w} \cdot \gamma$, this becomes the equivalent to problem where we minimize $|\vec{w}|^2$ under constraints

(a) $y_i \cdot \vec{w} \cdot \vec{x_i} > 1$ for $i \in \{1, 2, \ldots, n\}$

If data is not linearly separable, we **account for mistakes** by doing

$$\hat{w} = \underset{\vec{w},\xi_1,\xi_2,\ldots,\xi_n}{\arg\min} |\vec{w}|^2 + C \cdot \sum_{i=1}^{n} \xi_i$$
$$\text{when } y_i \cdot \vec{w} \cdot \vec{x} \geq 1 - \xi_i \text{ for } i \in \{1, 2, \ldots, n\} \text{ and } \xi \geq 0$$

which is equivalent to the **lagrangian dual**

$$\hat{w} = \arg\min_{\vec{\alpha}} \frac{1}{2} \cdot \sum_{i=1}^{n} \left(\sum_{j=1}^{n} y_i \cdot y_j \cdot \alpha_i \cdot \alpha_j \cdot x_i \cdot x_j \right) - \sum_{i=1}^{n} \alpha_i$$
$$\text{when } 0 \leq \alpha_i \leq C_i \;\; i \in \{1, 2, \ldots, n\} \text{ and } \sum_{i=1}^{n} y_i \cdot \alpha_i = 0$$

Learning Theory

The **true error** of a hypothesis h on data x that is drawn from distribution D where c^* correctly labels D is

$$err_D(h) = P_{x \sim D}\left(h(x) \neq c^*(x)\right)$$

This can be thought of the probability we make a mistake on future examples drawn from D.

The **sample error** of sample S for n examples is

$$err_S(h) = \frac{1}{n} \cdot \sum_{i}^{n} \mathbf{I}\left(h(x_i) \neq c^*(x_i)\right) \text{ where } \mathbf{I} \text{ is the indicator function (returns 0 or 1)}$$

The **probability approximately correct (PAC)** bounds the true error in terms of sample error. There are two cases:

Realizable where $c^* \in H$ then number of samples m

$$m \geq \frac{1}{\epsilon} \cdot \left(\log(|H|) + \log\left(\frac{1}{\delta}\right) \right)$$

Agnostic where c^* close to H then number of samples m

$$m \geq \frac{1}{2 \cdot \epsilon^2} \cdot \left(\log(|H|) + \log\left(\frac{2}{\delta}\right) \right)$$

meaning we need m examples to have **gernaralized error** $\leq \epsilon$ with probability $1 - \delta$

A hypothesis h **shatters** data $D = \{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_n\}$ if there is a parameter $\vec{\theta}$ for h that makes no mistakes on classifying D.

The **VC Dimension** of hypothesis space H is the number of elements in the largest set S that shatters H.

To define **bounds on generalization error**

(a) Realizable: $m = O\left(\frac{1}{\epsilon} \cdot \left(VCdim(H) \cdot \log \frac{1}{\epsilon} + \log \frac{1}{\delta}\right)\right)$

(b) Realizable: $m = \frac{C}{\epsilon^2} \cdot \left(VCdim(H) + \log \frac{1}{\delta}\right)$

K-means Clustering

The **k-means** algorithm for clustering n data points $\vec{x} = \vec{x}_1, \vec{x}_2, \ldots, \vec{x}_n$ into k clusters $C = C_1, C_2, \ldots, C_k$ finds centers $\vec{c} = \vec{c}_1, \vec{c}_2, \ldots, \vec{c}_k$ that minimize

$$\sum_{i=1}^{n} \min_{j=1,2,\ldots,k} dist(\vec{x}_i, \vec{c}_j)$$

where in **euclidean k-means clustering** the distance function $dist(\vec{a}, \vec{b})$ is

$$dist(\vec{a}, \vec{b}) = |\vec{a} - \vec{b}|^2 = (a_1 - b_1)^2 + (a_2 - b_2)^2 + \ldots + (a_n - b_n)^2$$

To **initialize** do the following algorithm

1. choose $\vec{c}_1$ at random
2. for $j = 2, 3, \ldots, k$
3. choose $\vec{c}_j$ from $\vec{x}_1, \vec{x}_2 \ldots, \vec{x}_n$ according to distribution $P\left(\vec{c}_j = \vec{x}_i\right) = D^\alpha\left(\vec{x}_i\right)$

where the distance $D^\alpha\left(\vec{x}_i\right)$ between $\vec{x}_i$ and its nearest center is

$$D^\alpha\left(\vec{x}_i\right) = \min_{p=1,2,\ldots,j} |\vec{x}_i - \vec{c}_p|^\alpha$$

For **random sampling** $\alpha = 0$, for **k-median** $\alpha = 1$, for **k-means++** $\alpha = 2$, and for **furthest point** $\alpha = \infty$.

Hierarchical Clustering

The **bottom-up** approach uses a distance $d(P, Q)$ between two clusters. We take the minimum of these and merge the two clusters.

single linkage defines the distance $d(P, Q)$ between two clusters P, Q as

$$d(P, Q) = \min_{x \epsilon P, y \epsilon Q} dist(x, y)$$

complete linkage uses maximum distance between point x from P and point y from Q

$$d(P, Q) = \max_{x \epsilon P, y \epsilon Q} dist(x, y)$$

Principal Component Analysis (PCA)

The **principal components** are orthogonal vectors along the direction of highest variance (the data is most spread out).

$$\max \frac{1}{n} \cdot \sum_{i=1}^{n} \left(\vec{v}^T \cdot \vec{x}_i\right)^2 = \vec{v}^T \cdot \mathbf{X} \cdot \mathbf{X^T} \cdot \vec{v} = \vec{v}^T \cdot \vec{v} \cdot \lambda = \lambda$$

To **find principal components** $\vec{v}$ of data $\mathbf{X} = \vec{x}_1, \vec{x}_2, \ldots, \vec{x}_n$, we use the sample co variance matrix $\left(\mathbf{X} \cdot \mathbf{X}^T\right)$ to solve

$$\left(\mathbf{X} \cdot \mathbf{X}^T\right) \cdot \vec{v} = \lambda \cdot \vec{v}$$

for variance λ. Note there are multiple solutions to λ so $\lambda_1 \geq \lambda_2 \geq \ldots$ are variances of principle components $\vec{v}_1, \vec{v}_2, \ldots$ respectively.

Neural Networks

A neural network is split into **layers** of nodes. Each node is connected to nodes in the previous layer and the next layer but not any nodes in the same layer.

The **input layer** has k nodes each representing one feature.

The **hidden layers** are any layers in between the input and **output** layers. Add more hidden layers to make decision boundary less linear and decrease overfitting.

The **parameters** are represented by the weights of edges between nodes. These parameters are trained to fit the data.

Use **batches** of a constant number of random training data points to train.

The **decision** function $\hat{y}$ for parameters $\vec{\theta}$ and data $\vec{x}$ is usually the logistic function

$$\hat{y} = \sigma\left(\vec{\theta} \cdot \vec{x}\right) = \frac{1}{1 + e^{-\vec{\theta} \cdot \vec{x}}}$$

We minimize the **loss** function ℓ to find the best parameters $\vec{\theta}^*$ using n training batches

$$\vec{\theta}^* = \arg\max_{\vec{\theta}} \sum_{i=1}^{n} \ell\left(\hat{y}\left(\vec{x_i}\right), \vec{y_i}\right)$$

Euclidean distance is a standard choice for loss function $\ell(a, b)$

$$\ell(a, b) = \frac{1}{2} \cdot |a - b|^2$$

To **train**, use stochastic gradient descent with learning rate η to update parameters at time $t + 1$

$$\vec{\theta}^*_{t+1} = \vec{\theta}^*_t - \eta \cdot \vec{\bigtriangledown} \ell\left(\hat{y}\left(\vec{x_i}\right), \vec{y_i}\right)$$

Use **back propagation** to calculate the gradient of the loss function $\vec{\bigtriangledown}\ell$.

1. create a directed acyclic graph (DAG) with input as source and output as target
2. for each layer l find:
3. the output $\vec{y}^{(l)}$ with respect to each input $\vec{x}_1^{(l)}, \vec{x}_2^{(l)}, \ldots, \vec{x}_k^{(l)}$ for k input nodes
4. the change in output y_j with respect to input x_i as $\frac{\partial y_j}{\partial x_i}$
5. in reverse topological order, contribute each node's partial derivative to partial derivative of the parent. This uses the **chain rule** of calculus to get $\vec{\bigtriangledown}\ell\left(\hat{y}\left(\vec{x_i}\right), \vec{y_i}\right)$

$$\vec{\bigtriangledown}\ell\left(\hat{y}\left(\vec{x_i}\right), \vec{y_i}\right) = \vec{\bigtriangledown}\frac{1}{2} \cdot \left|\hat{y}\left(\vec{x_i}\right) - \vec{y_i}\right|^2$$

An *example of using chain rule* to get the change in output y_j with respect to input x_i as $\frac{dy_j}{d_i}$ for all K nodes in the only hidden layer is

$$\frac{dy_j}{dx_i} = \sum_{k=1}^{K} \frac{\partial y_j}{\partial u_k} \cdot \frac{\partial u_k}{\partial x_i}$$

Deep Learning

vanishing gradients is an issue in deep networks where the final output depends more on layers close to the output than layers close to the input.

An **auto-encoder** uses unsupervised pre-training. Train each layer l of the neural network as if the output were the input layer (layer $l-1$). Fix parameters. Move to next layer.

Look up **convolutional neural networks** on the internet

The **recurrent neural network** can contain cycles making it ideal for handwriting and speech recognition.

Boosting

A **weak learner** is an algorithm that performs better than random guessing

$$error \leq \frac{1}{2} - \gamma$$

The **adaptive boosting** or *adaboost* algorithm turns a weak classifier h into a strong classifier H by running h T times

1. for $t = 1, 2, \ldots, T$ use a vector $D_t(i)$ to weight training sample i at iteration t

$$D_{t+1}(i) = \begin{cases} \frac{D_t(i)}{Z_t} \cdot e^{-\alpha_t} & if y_i = h_t(x_i) \\ \frac{D_t(i)}{Z_t} \cdot e^{\alpha_t} & if y_i \neq h_t(x_i) \end{cases}$$

where

$$Z_t = \sum_{i=1}^{n} D_t(i) \cdot e^{-\alpha_t \cdot y_i \cdot h_t(x_i)} \qquad \text{and} \qquad \alpha_t = \frac{1}{2} \cdot \log\left(\frac{1-\epsilon_t}{\epsilon}\right)$$

2. final classifier $H(x) = sign\left(\sum_{t=1}^{T} \alpha_t \cdot h_t(x)\right)$

has final error $err_S(H)$ and only needs T rounds

$$err_S(H) \leq \epsilon = e^{-2\cdot\gamma^2\cdot T} \qquad \text{and} \qquad T = O\left(\frac{1}{\gamma^2}\cdot\log\left(\frac{1}{\epsilon}\right)\right)$$

Active Learning

Active learning is a learning algorithm that requests the labels on some informative training examples.

ex. in SVM, get label for data point closest to current separator (highest uncertainty)

The **number of label requests** for passive supervised is $\Omega(1/\epsilon)$ but active learning is $O(\log 1/\epsilon)$ for getting an ϵ accurate threshold.

For **online learning**, when an examples arrives we decide to ask for label or not.

Warning: selectively viewing labels (active learning) can create sample bias.

The **current version space** H_t of hypothesis space H is the set of possible hypotheses after viewing t labeled samples.

$$H \supseteq H_1 \supseteq H_2 \supseteq \ldots \supseteq H_T$$

The **region of disagreement** contains any samples that would decrease the current version space if chosen

$$\begin{cases} H_{t+1} \subset H_t & \text{if } x_t \,\epsilon\, DIS(H_t) \\ H_{t+1} = H_t & \text{otherwise} \end{cases}$$

Semi-Supervised Learning

Semi-Supervised Learning does no label requesting. It is supervised learning with lots of additional unlabeled data. example: semi-supervised SVM

1. maximize margin over labeled points
2. use separator to give labels to unlabeled points
3. adjust labels on unlabeled points to maximize margin

Iterative co-training runs multiple classifiers each with different feature sets

1. use labeled points to find initial classifier
2. use label from most confident classifier for each unlabeled data point (and repeat)

A **graph-based** approach creates a graph with edges between similar examples. The run a graph partitioning algorithm.

Bayesian Networks

A **Bayesian network** is a directed acyclic graph (DAG) with probability of all n nodes $X_1, X_2, \ldots, X_n$ defined as

$$\mathbf{P}(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} \mathbf{P}(X_i | parents(X_i))$$

Two variables X and Y are **D-separated**, or conditionally independent, given variables $Z = \{Z_1, Z_2, \ldots\}$ if all undirected paths from X to Y are **blocked**.

A path from X to Y given $Z = \{Z_1, Z_2, \ldots\}$ is **blocked** if any node A on the path

(a) if A in Z and has two edges directed away from A on the path

(b) if A in Z and has one edge directed away and one toward A on the path

(c) if A not in Z and has two edges directed toward A on the path

Factor Graphs

A **factor graph** uses a black box ψ_1 to connect any k variables (nodes) $\vec{s_1} = x_1, x_2, \ldots, x_k$ in a graph. In this graph, variables are only connected to factors.

The joint probability of a set of variables $x_1, x_2, \ldots$ with q contributing factors is

$$\mathbf{P}(S = x_1, x_2, \ldots) = \frac{1}{Z} \cdot \prod_{i=1}^{q} \psi_i (s_i \subseteq S)$$

where s_i is a subset of nodes S that ψ_i contributes to.

A **markov random field** (MRF) represents a black box along an edge between two nodes $A = a_1, a_2, \ldots, a_n$ and $B = b_1, b2, \ldots, b_m$ as a matrix X where each element is

$$X_{i,j} = P(a_i, b_j) \qquad \forall\ i = 1, 2, \ldots, n \text{ and } j = 1, 2, \ldots, m$$

A **conditional random field** (CRF) is similar to a markov random field (MRF) but defines conditional factors for node A as *conditional probability*

$$X_i = P(a_i | B) \qquad \forall\ i = 1, 2, \ldots, n$$

ex. for labeling words in a sentance as *noun, verb, etc.* X_i is probability of label (ex. *noun*) given the word

Hidden Markov Model

The **hidden nodes** $Y_1, Y_2, \ldots, Y_k$ and **observed nodes** $X_1, X_2, \ldots, X_k$ are used to define the joint probability of observed and hidden meaning

$$\mathbf{P}(X_1, X_2, \ldots, X_k, Y_1, Y_2, \ldots, Y_k) = \prod_{i=1}^{k} \mathbf{P}(X_i|Y_i) \cdot \mathbf{P}(Y_i|Y_{i-1})$$

An **emission matrix** A can be defined as $A_{r,s} = \mathbf{P}(X_i = r|Y_i = s)$ and **transition matrix** B defined as $B_{s,t} = \mathbf{P}(Y_i = s|Y_{i-1} = t)$ to redefine

$$\mathbf{P}(X_1, X_2, \ldots, X_k, Y_1, Y_2, \ldots, Y_k) = \prod_{i=1}^{k} A_{Y_i,X_i} \cdot B_{Y_i,Y_{i-1}}$$

Expectation Maximization

Hard EM (expectation maximization) is k-means clustering.

Soft EM uses a *guess* for a centroid μ_i and covariance Σ_i to define each cluster. Do the steps:

1. create a data point for each possible value of the *latent variables*. Weight each point according to model confidence
2. set parameters to values that maximize likelihood