Decision Trees

Inductive Bias: prefer the smallest tree consistent w/ the training data (i.e.) 0 error rate

Entropy: $H(Y) = -\sum_{y \in \text{values}(Y)} P(Y=y)\log_2 P(Y=y)$

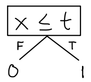
0 = pure , 1 = perfectly mixed

Mutual Information: $I(X;Y) = H(Y) - \sum_{x \in \text{values}(x)} P(X=x)H(Y|X=x)$

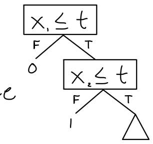
1 = 100% correlation between attribute & label

Memorizer: learns labels given attributes, MV otherwise

Decision Stump:



Decision Tree



K-Nearest Neighbor

Inductive Bias: Similar/Nearby points should have sim. labels
All label dimensions are created equal

Euclidian Distance: $d(p,q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$

* feature scale could dramatically influence class results

Model Selection

hyperparameters - tunable aspects of the model, that the learning algorithm does not select

K-fold cross-validation: create k partitions in D
do k runs: train using k-1 partitions and calc validation error on remaining partion (rotating validation partition on each run)
report average validation error

leave-one-out validation: K=N partitions
train on N-1 samples; validate only one sample per run

* model selection/hyperparameter optimization is just another form of learning

Derivative Rules

$\frac{d}{dx}(\text{constant}) = 0$ $\frac{d}{dx}(x) = 1$ $\frac{d}{dx}(x^2) = 2x$ $\frac{d}{dx}(\sqrt{x}) = \frac{1}{2}\cdot x^{-\frac{1}{2}} = \frac{1}{2\sqrt{x}}$

$\frac{d}{dx}(e^x) = e^x$ $\frac{d}{dx}(a^x) = \ln(a)a^x$ $\frac{d}{dx}(\ln(x)) = \frac{1}{x}$ $\frac{d}{dx}(\log_a(x)) = \frac{1}{x\cdot\ln(a)}$

$\frac{d}{dx}(\sin(x)) = \cos(x)$ $\frac{d}{dx}(\cos(x)) = -\sin(x)$ $\frac{\sin(x)}{\cos(x)} = \tan(x)$ $\frac{d}{dx}(\tan(x)) = \sec^2(x)$

Multiplication by constant: $\frac{d}{dx}(c\cdot f(x)) = c\cdot\frac{d}{dx}(f(x))$

Power Rule: $\frac{d}{dx}(x^n) = nx^{n-1}$

Reciprocal Rule: $\frac{d}{dx}(1/f) = \frac{-\frac{d}{dx}(f)}{f^2}$

Sum Rule: $\frac{d}{dx}(f+g) = \frac{d}{dx}(f) + \frac{d}{dx}(g)$

Difference Rule: $\frac{d}{dx}(f-g) = \frac{d}{dx}(f) - \frac{d}{dx}(g)$

Product Rule: $\frac{d}{dx}(f\cdot g) = \frac{d}{dx}(f)\cdot g + f\cdot\frac{d}{dx}(g)$

Quotient Rule: $\frac{d}{dx}(f/g) = \frac{\frac{d}{dx}(f)\cdot g - f\cdot\frac{d}{dx}(g)}{g^2}$

Chain Rule:

- as composition of functions: $\frac{d}{dx}(f\circ g) = (f'\circ g)\cdot g'$
- using ': $\frac{d}{dx}(f(g(x))) = \boxed{f'(g(x))\cdot g'(x)}$
- using $\frac{d}{dx}$: $\frac{dy}{dx} = \frac{dy}{du}\cdot\frac{du}{dx}$

$$f'(x) = \lim_{h\to 0}\frac{f(x+h) - f(x)}{h}$$

Perceptron

Inductive Bias:
1) decision boundary should be linear
2) most recent mistakes are most important (and should be corrected)

* try to learn hyperplane directly

$h(x) = \text{sign}(\theta^T x)$ for $y \in \{-1, +1\}$

First Perceptron: Linear Classifier

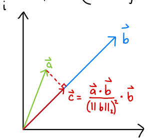
$\hat{y} = h(\vec{x}) = \text{sign}(\vec{w}^T x + b)$
$= \text{sign}(w_1x_1 + w_2x_2 + \ldots + w_Mx_M + b)$

orthogonal: $\vec{a}$ is orthogonal to $\vec{b}$ iff $\vec{a}^T\vec{b} = 0$
(right angle to each other)

dot product: $\vec{a}\cdot\vec{b} = \vec{a}^T\vec{b} = \sum_i a_i b_i$

l_2 norm: $\|\vec{u}\|_2 = \sqrt{\sum_i (u_i)^2}$ (length of vector/Euclidian Distance)

vector projection:



Ex: $\vec{a}$ projected onto $\vec{b}$

hyperplanes: 2D - line $S = \{\vec{x} : \vec{w}^T\vec{x} + b = 0\}$
3D - plane
4+D - hyperplane

halfspaces: all the points on one side of the hyperplane

$S_+ = \{\vec{x} : \vec{w}^T\vec{x} + b > 0\}$

$S_- = \{\vec{x} : \vec{w}^T\vec{x} + b < 0\}$

batch learning: learn all examples at once

online learning: gradually learn as example is recieved

↳ initialize parameters $\vec{w} = [w_1, w_2, \ldots, w_n]^T = [0, 0, \ldots, 0]^T$
$b = 0$ ← * start at zero vector

for i in range(M):

1) recieve instance $\vec{x}^{(i)}$
2) predict $\hat{y} = h(\vec{x}) = \text{sign}(\vec{w}^T\vec{x} + b)$ $\text{sign}(a) = \begin{cases} +1 & \text{if } a \ge 0 \\ -1 & \text{otherwise} \end{cases}$
3) recieve label $y^{(i)}$
4) update parameters: if $y^{(i)} = +1$ and $y^{(i)} \ne \hat{y}$:
$\vec{w} \leftarrow \vec{w} + \vec{x}^{(i)}$
$b \leftarrow b + 1$ * positive mistake

if $y^{(i)} = -1$ and $y^{(i)} \ne \hat{y}$:
$\vec{w} \leftarrow \vec{w} - \vec{x}^{(i)}$
$b \leftarrow b - 1$ * negative mistake

hyperplane w. θ: $\mathcal{H} = \{x : w^T x + b = 0\} = \{x : \theta^T x' = 0$ and $x'_0 = 1\}$

$\theta = [b, w_1, w_2, \ldots, w_M]^T$

$x' = [1, x_1, x_2, \ldots, x_3]^T$

halfspaces w. θ: $\mathcal{H}_+ = \{x : \theta^T x > 0$ and $x'_0 = 1\}$

$\mathcal{H}_- = \{x : \theta^T x < 0$ and $x'_0 = 1\}$

mistake bound: if some data has margin γ and all points lie inside a ball of radius R rooted at the origin, then the online Perceptron algorithm makes $\le (R/\gamma)^2$ mistakes

$d = \frac{ax + by + c}{\sqrt{a^2 + b^2 + c^2}}$ where $a = w_1$, $b = w_2$, $c = b$, $x = x_1$, $y = x_2$

Summary:
- linear classifier
- simple: mistake = update params
- converge if linearly seperable
- can bound # of mistakes

Linear Regression

assume $y = w^T x$

parameters: $w = [w_0, w_1, w_2, \ldots, w_D]^T$ ($w_0 \to 1$)

objective function: minimize the squared error

$$\ell_D(w) = \sum_{n=1}^{N} (w^T x^{(n)} - y^{(n)})^2$$

optimize objective: solve in closed form

- take ~~partial derivatives~~ → gradient ∇
- set them equal to 0

suppose $y \in \mathbb{R}$ and D-dimensional inputs $x = [1, x_1, x_2, \ldots, x_D]^T \in \mathbb{R}^{D+1}$

training data $D = \{(x^{(n)}, y^{(n)})\}_{n=1}^{N}$

$$X = \begin{bmatrix} 1 & x^{(1)T} \\ 1 & x^{(2)T} \\ \vdots & \\ 1 & x^{(N)T} \end{bmatrix} = \begin{bmatrix} 1 & x_1^{(1)} & \ldots & x_D^{(1)} \\ 1 & x_1^{(2)} & \ldots & x_D^{(2)} \\ \vdots & & & \\ 1 & x_1^{(N)} & \ldots & x_D^{(N)} \end{bmatrix} \in \mathbb{R}^{N \times (D+1)}$$

is the design matrix

$y = [y^{(1)}, y^{(2)}, \ldots, y^{(N)}]^T \in \mathbb{R}^N$ is the target vector

Algorithm:

Input: $D = \{X, y\} = \{(x^{(n)}, y^{(n)})\}_{i=1}^{N}$

1) Compute the pseudoinverse of X either
   a) Directly as $X^{+} = (X^T X)^{-1} X^T$ if $X^T X$ is invertable or
   b) Through singular value decomposition (SVD): if $X = U \Sigma V$, then $X^{+} = V \Sigma^{+} U^T$ where Σ^{+} inverts all non-zero elements of Σ
2) Compute $\hat{w} = X^{+} y$

Output: $\hat{w}$ weight vector

Inductive Bias: true relationship between inputs & outputs is linear

Gradient Descent

current weight vector is $\hat{w}$ (slope m)

move some distance η in most downhill direction, $\hat{v}$:

$$w^{(t+1)} = w^{(t)} + \eta \hat{v}$$

gradient points in direction of steepest increase, $\hat{v}$ is opposite:

$$\hat{v}^{(t)} = -\frac{\nabla_w \ell_D(w^{(t)})}{\|\nabla_w \ell_D(w^{(t)})\|_2}$$

$$\underbrace{\eta^{(t)}}_{\text{step size}} = \underbrace{\eta^{(0)}}_{\text{init step size}} \cdot \underbrace{\|\nabla_w \ell_D(w^{(t)})\|}_{\text{magnitud of grad}}$$

each iteration:

$$\underbrace{w^{(t+1)}}_{\text{next weight vector}} = \underbrace{w^{(t)}}_{\text{curr weight vec}} + \underbrace{\eta^{(t)}}_{\text{step size}} \underbrace{\hat{v}}_{\text{vector pointing in opposite dir from gradient}}$$

..... math

$$\boxed{= \underbrace{w^{(t)}}_{\text{curr weight vec}} - \underbrace{\eta^{(0)}}_{\text{learning rate}} \underbrace{\nabla_w \ell_D(w^{(t)})}_{\text{gradient (vect of derivatives) of loss func wrt } w^{(t)}}}$$

Linear Regression via Gradient Descent

Input: $D = \{\underbrace{X}_{\text{design matrix}}, \underbrace{y}_{\text{target vector}}\} = \{(\underbrace{x^{(n)}}_{\text{attributes}}, \underbrace{y^{(n)}}_{\text{labels}})\}_{n=1}^{N}, \underbrace{\eta^{(0)}}_{\text{learning rate}}$ ← dataset size (N)

1) initialize $w^{(0)}$ to all zeros, $w = [\overset{w_0}{0}, \overset{w_1}{0}, \ldots, \overset{w_N}{0}]^T$ { init $t=0$
2) while termination criteria is not satisfied (loop):
   a) compute gradient $\nabla_w \ell_D(w^{(t)})$ ← gradient (vector of derivaties of loss function) wrt $w^{(t)}$
   b) update w: $w^{(t+1)} = w^{(t)} - \eta^{(0)} \nabla_w \ell_D(w^{(t)})$
   c) increment t

Output: $w^{(t)}$ ← regression line

Minimizing the Squared Error

$$\ell_{\mathcal{D}}(\boldsymbol{w}) = \sum_{n=1}^{N} \left(\boldsymbol{w}^T \boldsymbol{x}^{(n)} - y^{(n)}\right)^2 = \sum_{n=1}^{N} \left(\boldsymbol{x}^{(n)T} \boldsymbol{w} - y^{(n)}\right)^2$$

$$= \|X\boldsymbol{w} - \boldsymbol{y}\|_2^2 \text{ where } \|\boldsymbol{z}\|_2 = \sqrt{\sum_{d=1}^{D} z_d^2} = \sqrt{\boldsymbol{z}^T \boldsymbol{z}}$$

$$= (X\boldsymbol{w} - \boldsymbol{y})^T (X\boldsymbol{w} - \boldsymbol{y})$$

$$= \left(\boldsymbol{w}^T X^T X \boldsymbol{w} - 2\boldsymbol{w}^T X^T \boldsymbol{y} + \boldsymbol{y}^T \boldsymbol{y}\right)$$

$$\nabla_{\boldsymbol{w}} \ell_{\mathcal{D}}(\widehat{\boldsymbol{w}}) = \left(2X^T X \widehat{\boldsymbol{w}} - 2X^T \boldsymbol{y}\right) = 0$$

$$\to X^T X \widehat{\boldsymbol{w}} = X^T \boldsymbol{y}$$

$$\to \widehat{\boldsymbol{w}} = \left(X^T X\right)^{-1} X^T \boldsymbol{y}$$

$$\ell_{\mathcal{D}}(\boldsymbol{w}) = \sum_{n=1}^{N} \left(\boldsymbol{w}^T \boldsymbol{x}^{(n)} - y^{(n)}\right)^2 = \sum_{n=1}^{N} \left(\boldsymbol{x}^{(n)T} \boldsymbol{w} - y^{(n)}\right)^2$$

$$= \|X\boldsymbol{w} - \boldsymbol{y}\|_2^2 \text{ where } \|\boldsymbol{z}\|_2 = \sqrt{\sum_{d=1}^{D} z_d^2} = \sqrt{\boldsymbol{z}^T \boldsymbol{z}}$$

$$= (X\boldsymbol{w} - \boldsymbol{y})^T (X\boldsymbol{w} - \boldsymbol{y})$$

$$= \left(\boldsymbol{w}^T X^T X \boldsymbol{w} - 2\boldsymbol{w}^T X^T \boldsymbol{y} + \boldsymbol{y}^T \boldsymbol{y}\right)$$

$$\nabla_{\boldsymbol{w}} \ell_{\mathcal{D}}(\boldsymbol{w}) = \left(2X^T X \boldsymbol{w} - 2X^T \boldsymbol{y}\right)$$

$$H_{\boldsymbol{w}} \ell_{\mathcal{D}}(\boldsymbol{w}) = 2X^T X$$

$H_{\boldsymbol{w}} \ell_{\mathcal{D}}(\boldsymbol{w})$ is positive semi-definite

Matrix

* for dot product, $w_1 == h_2$

$$\begin{bmatrix} A & B \\ C & D \\ E & F \end{bmatrix} \times \begin{bmatrix} G \\ H \end{bmatrix} = \begin{bmatrix} AG + BH \\ CG + DH \\ EG + FH \end{bmatrix}$$

$$(AB)^T = B^T A^T$$