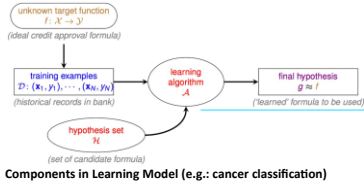
Definitions of Learning Types and AI/ML Terms

AI: compute something that shows intelligent behavior.
Machine learning: *(need rapid decision and not manually programmable)*
- Improve performance with experience computed from data.
- use data to compute hypothesis g that approximate target function f.
Data Mining: use huge data to find property (that is interesting).
If interest = find h that approx f, then they are the same
Statistics: use data to make inference about an unknown process, provable maths > computation, useful tools for ML
When use machine learning:
(1) exist "underlying pattern" to be learned *can't give a math def*
(2) no programmable (easy) definition (analytic solution)
(3) somehow there is data about the pattern
Types of machine learning:
(1) binary/multiclass classification
(2) regression: predict a continuous outcome variable
(3) unsupervised learning: categorizing data points (cluster)
(4) structured learning: predict structured output (e.g. sequence, trees, graphs, images) instead of simple scalar values
(5) active learning: ML algo selects which data require labelling (those that help learning most, e.g. uncertain ones) → for cases when labelling is expensive
(6) reinforcement: rewards for agents to learn

Formulate Learning Problem / Model



Components in Learning Model (e.g.: cancer classification)
input space X - patient's medical history, symptoms, personal health information, etc.
output space Y – he/she has cancer or not
target function (f:X→Y) - ideal formula to identify a patient's cancer situation
data set - All available patients' information and their corresponding correct cancer diagnosis
hypothesis (g:X→Y): Learned formula to be used
Binary Classification Learning Model
$y = \sum_{i=1}^{d} w_i x_i \qquad input\ x = [x_1, \dots, x_d]^T$
prediction: $\alpha_n \geq 0$
if y > threshold = positive; if y < threshold = negative

Perceptron Learning Algorithm (PLA)
perceptron function h(x)
$h(x) = sign(\sum_{i=1}^{d} w_i x_i - threshold) = sign(w^T x)$
each **w** represents a hypothesis **h**
Perceptron/ Linear separator: hypothesis set [set of h(x)]
Linearly separable: exist perfect w_f that $y = sign(w_f^T x)$
Perceptron Learning algorithm:
w(1) = 0
for iteration t = 1, 2, 3, …
  current weight vector is w(t)
  from (x1, y1), …, (xN, yN) pick any misclassified example
  call the misclassified example (x_*, y_*), $sign(w(t) \cdot x_*) \neq y_*$
  update the weight: $w(t+1) = w(t) + y_* x_*$
  t = t + 1
Theorem: if the data can be fit by a linear separator, then after some finite number of steps, the algorithm will find one (if linear separable and corrected by mistake)
if not linear sep. → will halt eventually (dk when, shd be fast)
✓ simple to implement, fast, works on any dimension of d
Why perceptron learning works
(1) w_{t+1} is always more aligned with w_f than w_t
$w_{t+1} = w_t + y_* x_* \rightarrow w_f^T w_{t+1} = w_f^T (w_t + y_* x_*)$
$\min_n y_n w_f^T x_n > 0$ (because linearly separable)
$y_* w_f^T x_* \geq \min_n y_n w_f^T x_n$ (selected y_*, x_* alw better than worse)
$\therefore w_f^T w_{t+1} = w_f^T (w_t + y_* x_*) \geq w_f^T w_t + \min_n y_n w_f^T x_n > w_f^T w_t$
** higher dot product = more aligned*
** y_*, x_* are selected misclassified example at t*
(2)
$$\|w_{t+1}\|^2 = \|w_t + y_* x_*\|^2 = \|w_t\|^2 + 2 y_* w_t^T x_* + \|y_* x_*\|^2$$
$$\leq \|w_t\|^2 + 0 + \|y_* x_*\|^2 \leq \|w_t\|^2 + \max_n \|y_n x_n\|^2$$
start from $w_0 = 0$, after T mistake corrections,
$$\frac{w_f^T}{\|w_f\|} \cdot \frac{w_T}{\|w_T\|} \geq \sqrt{T} \cdot constant$$
Inner product of Wf and Wt grows fast, len of Wt grows slowly.
$$\|w_t\|^2 \leq t \left(\max_{1 \leq n \leq N} \|x_n\| \right)^2$$

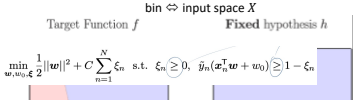
The BIN Model
- bin with red and green marbles
- pick a sample of N marbles independently
- μ = probability to pick a red marble *(population distribution)*
  ν = fraction of red marbles in the sample *(sample distribution)*
  sample → dataset → ν | BIN → outside the data → μ

Hoeffding's Inequality
the statement "ν = μ" is probably approximately correct (PAC)
$$P[|\nu - \mu| > \epsilon] \leq 2e^{-2\epsilon^2 N} \qquad for\ any\ \epsilon > 0$$
$$P[|\nu - \mu| \leq \epsilon] \geq 1 - 2e^{-2\epsilon^2 N} \qquad for\ any\ \epsilon > 0$$
ν = in-sample probability | μ = out-sample probability
N = sample size | ϵ = gap
Properties *N很大或者eps很小的时候的时候，ν和μ大概差不多（相差很多的概率很小）*
(1) do not depend on μ → no need to know μ
(2) Valid for all N and ϵ, $\uparrow N\ or \downarrow \epsilon \rightarrow \uparrow P(\nu \approx \mu)$
(3) sufficiently large N → likely can infer unknown μ by known ν.
Why the inequality is useful for learning
(1) samples must be independent *and the training and testing sets follow the same distribution*
(2) the bound does not depend on μ or population size (bin size)
(3) key player in bound is size N of samples
Find min N to achieve $\epsilon \leq k$ and $P[|\nu - \mu| > \epsilon] = \delta$
$$N \geq \frac{1}{2k^2} ln \frac{2}{\delta}$$
Problem 1.7 – Flip Coins Probability
head probability = μ, #trials = N
$P[k|N,\mu] = \binom{N}{k} \mu^k (1-\mu)^{N-k}$, training error v = $\frac{k}{N}$
given N=6, 2 coins w/ μ = 0.5, plot $P[\max_i |v_i - \mu_i| > \epsilon]$, on same plot show the bound that would be obtained using H. equality

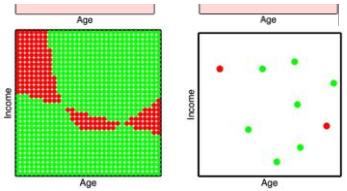
Soft margin
当数据非线性可分时，没有可行解for ynfn>=1 for all n -> slack variable. C>=0 is a hyper para to 控制允许violate margin constraint的强度

$$\mathbb{P}[\max_i |v_i - \mu_i| > \epsilon] = 1 - \mathbb{P}[\max_i |v_i - \mu_i| \leq \epsilon]$$
$$= 1 - (\mathbb{P}[|v - \mu| \leq \epsilon])^2$$
$$= 1 - (\mathbb{P}[|\frac{k}{N} - 0.5| \leq \epsilon])^2$$
$$= 1 - (\mathbb{P}[|k - 3| \leq 6\epsilon])^2$$
$$= 1 - (\mathbb{P}[3 - 6\epsilon \leq k \leq 3 + 6\epsilon])^2$$
$$= 1 - \Big(\sum_{3-6\epsilon \leq k \leq 3+6\epsilon} \binom{N}{k} \mu^k (1-\mu)^{N-k} \Big)^2$$

Link BIN Model to learning
unknown value μ in bin ⇔ unknown function f in learning bin ⇔ input space X



$$\mathcal{L}(w, w_0, \alpha, \xi, \mu) = \frac{1}{2} w^T w + C \sum_{n=1}^{N} \xi_n - \sum_{n=1}^{N} \alpha_n (\hat{y}_n (w^T x_n + w_0) - 1 + \xi_n) - \sum_{n=1}^{N} \mu_n \xi_n$$
$\mu_n \geq 0$ Same pred method

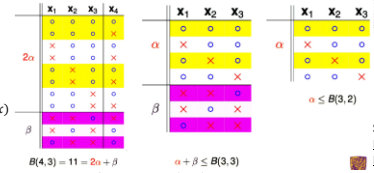


red = area where $h(x) \neq f(x)$ | green = area where $h(x) = f(x)$
Out-of-sample error **= μ *in bin***
$$E_{out}(h) = P_x[h(x) \neq f(x)]$$
fraction of population where f and h disagree
(area of red region in left pic)
In-sample error **= ν**
$$E_{in}(h) = \frac{1}{N} \sum_{n=1}^{N} [h(x) \neq f(x)]$$
fraction of $\mathcal{D}$ where f and h disagree
(proportion of red marbes in right pic)
Rewrite Hoeffding's inequality with in/out error ***(verification)***
Formal Guarantee for any fixed h, when the sample size N is large, E_{in} is probably close to E_{out} (within ϵ)
$$P[|E_{in}(h) - E_{out}(h)| > \epsilon] \leq 2e^{-2\epsilon^2 N} \qquad for\ any\ \epsilon > 0$$
$$P[|E_{in}(h) - E_{out}(h)| \leq \epsilon] \geq 1 - 2e^{-2\epsilon^2 N} \qquad for\ any\ \epsilon > 0$$
Properties
(1) do not depend on E_{out} → no need to know E_{out}
(2) Valid for all N and ϵ
(3) sufficiently large N → $E_{in}(h) \approx E_{out}(h)$
** to claim g ≈ f , $E_{in}(h)$ has to be small despite large N
Verification vs Learning

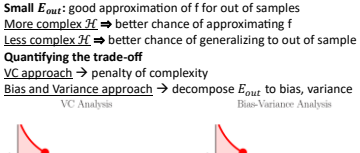
Verification	Real Learning
Fixed single hypothesis h	Fixed *hypothesis set* $\mathcal{H}$
h to be certified	g to be certified
h does not depend on $\mathcal{D}$	g results after searching $\mathcal{H}$ to fit $\mathcal{D}$
No control over E_{in}	Pick best E_{in}

** g = best hypothesis out of hypothesis set H*
Verification evaluate a particular hypothesis *(the equation above)*
Learning evaluate a set of hypothesis *(simulate real learning)*
Rewrite Hoeffding's inequality for hypothesis set ***(learning)***
$$P[|E_{in}(g) - E_{out}(g)| > \epsilon] \leq 2|\mathcal{H}|e^{-2\epsilon^2 N} \qquad for\ any\ \epsilon > 0$$
$$P[|E_{in}(g) - E_{out}(g)| \leq \epsilon] \geq 1 - 2|\mathcal{H}|e^{-2\epsilon^2 N} \qquad for\ any\ \epsilon > 0$$
- finite |H| & larget N → any g picked by learning algo satisfies
$E_{in}(g) \approx E_{out}(g)$ **->any g that makes Ein = 0, PAC ensures Eout=0**
How did |H| come in?

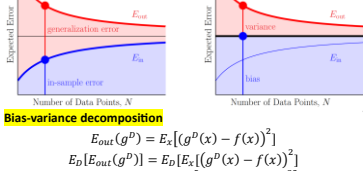
Bad events B:
$$B_g = \{|E_{out}(g) - E_{in}(g)| > \epsilon\}$$
$$B_m = \{|E_{in}(h_m) - E_{in}(h_m)| > \epsilon\}$$
dk which g selected → use worst case union bound
(这种做法假设各个h之间没有交集)
$$P[B_g] \leq P[any\ B_m] \leq \sum_{m=1}^{|H|} P[B_m]$$
Trade-off of |ℋ| - small: few choices, can't find g that makes Ein small enough, but Ein ≈ Eout; Large相反
|ℋ| fails to account for similarity between hypotheses. If all the "bad" datasets overlap, maybe we can handle much bigger than the union bound suggests. **Why PLA has inf. |ℋ| but can learn?**
Overlapping for similar hypotheses h1 ≈ h2. The union bound over-estimating. Many lines but only one dichotomy
Growth Function and Dichotomy
Hypothesis $h: X \rightarrow \{-1, +1\}$
e.g. all lines in R^2 → possibly infinite size
Dichotomy $h: \{x_1, x_2, \dots, x_N\} \rightarrow \{-1, +1\}$
e.g. {oooo, ooox, ooxx, …} → upper bounded by 2^N
Growth function
largest set of dichotomies induced by H *(measures diversity of H)*
$m_H(N) = \max_{x_1, \dots, x_N} |H(x_1, \dots, x_N)|$ *e.g.* $m_H(3) = 8, m_H(4) = 14$
Problem 2.3c **max num of dichotomies**
Two concentric spheres: $d_{vc} = 2, m_H = \binom{N+1}{2} + 1 = \frac{N^2+N}{2} + 1$
Break point (k)
if H cannot shatter all set of k points (if k* is bp, all k ≥ k* are bp) → any k that $m_H(k) < 2^k$ → upper bound growth function
Special cases of H and their growth function
Positive ray (1d plane, +ve after ray)
$m_H(N) = N + 1, \quad k = 2$
Positive interval (1d, +ve in intv)
$m_H(N) = \binom{N+1}{2} + 1 = \frac{1}{2}N^2 + \frac{1}{2}N + 1, \quad k = 3$
Convex set $m_H(N) = 2^N, \quad k = \infty$
Positive Rectangle (Problem 2.2)
Show that for the learning model of positive rectangles (aligned horizontally or vertically), $m_h(4) = 2^4$ and $m_h(5) < 2^5$. Hence, give a bound for $m_h(N)$.
找到一个case能被shatter就行
$N = 4$: pick (1,3), (2, 4), (3,1), (4,2) → shattered by +ve rectangles in practice: $N = 10 d_{vc}$ | VC bound still loose | smaller dvc better
idea: any two points ,draw a rect w/ them as diag pts → rect shd not contain any other point, otherwise, whenever the two diagnoal points have values 1, the middle point will have value 1 as well, which excludes the possibility of having -1.
Trade-off of d_{vc}
↑ d_{vc}: better chance of approximating f ($E_{in} \approx 0$) **but Ω ↑,Ein ↓ but Ω ↓, Ein ↑**
↓ d_{vc}: better chance of generalizing to out of sample ($E_{in} \approx E_{out}$)
Goal of using VC dimension: understanding which learning models are more or less powerful under which circumstances
Bounding function B(N,k): max possible $m_H(N)$ at break point k
$N = 5$: draw hor. and vert. lines thru each pt above → 9-grid plane. 5th point must out of 9 grid, else exist rect. containing
- if 5th pt outside 9-grid area → must lie below/above at least points (in either x or y) → these three points construct a rec which contains a point in it → $m_H(5) < 2^5$
$w = \sum_{n=1}^{N} \hat{\alpha}_n \hat{y}_n x_n$ uctural risk minimization to choose best learning machine
$0 = \sum_{n=1}^{N} \hat{\alpha}_n \hat{y}_n$ imation and Generalization



- the transition from smaller B(N, k) to larger ones
If k is any break point for H, so $m_H(k) < 2^k$, then

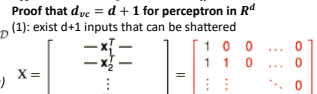


Small E_{out}: good approximation of f for out of samples
More complex ℋ ⇒ better chance of approximating f
Less complex ℋ ⇒ better chance of generalizing to out of sample
Quantifying the trade-off
VC approach → penalty of complexity
Bias and Variance approach → decompose E_{out} to bias, variance



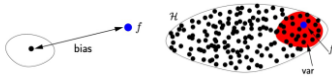
Bias-variance decomposition
$$E_{out}(g^D) = E_x[(g^D(x) - f(x))^2]$$
$$E_D[E_{out}(g^D)] = E_D[E_x[(g^D(x) - f(x))^2]]$$
$$= E_x[E_D[(g^D(x) - f(x))^2]]$$
Evaluate $E_D[(g^D(x) - f(x))^2]$:
(1) define average hypothesis $\bar{g}(x) = E_D[g^D(x)]$
(2) let there be K datasets $D_1, D_2, \dots, D_K$, $\bar{g}(x) \approx \frac{1}{K} \sum_{k=1}^{K} g^{D_k}(x)$
$$E_D\left[(g^D(x) - f(x))^2\right] = E_D\left[(g^D(x) - \bar{g}(x) + \bar{g}(x) - f(x))^2\right]$$
$$= E_D[(g^D(x) - \bar{g}(x))^2 + (\bar{g}(x) - f(x))^2$$
$$+ 2(g^D(x) - \bar{g}(x))(\bar{g}(x) - f(x))]$$
$$= E_D\left[(g^D(x) - \bar{g}(x))^2\right] \quad (var(x))$$
$$+ (\bar{g}(x) - f(x))^2 \quad (bias(x))$$
$$E_D[E_{out}(g^D)] = E_x\left[E_D\left[(g^D(x) - f(x))^2\right]\right]$$

(2) cannot shatter any set of d+2 points
consider a d-dimensional general case:
more rows than columns:
linear dependence (some a_i non-zero)
$\mathbf{x}_{d+2} = a_1 \mathbf{x}_1 + a_2 \mathbf{x}_2 + \dots + a_{d+1} \mathbf{x}_{d+1}$
can you generate (sign(a_1), sign(a_2), …, sign(a_{d+1}), ×)? if so, what **w**?
$$\mathbf{w}^T \mathbf{x}_{d+2} = a_1 \mathbf{w}^T \mathbf{x}_1 + a_2 \mathbf{w}^T \mathbf{x}_2 + \dots + a_{d+1} \mathbf{w}^T \mathbf{x}_{d+1}$$
$$> 0 \text{(contradiction!)}$$
For $N \leq d_{vc}$: H could shatter the data (some N points)
For $N > d_{vc}$: N is break point for H; H cannot shatter the data.
Vapnik-Chervonenkis Bound (VC Bound)
$$P[|E_{in}(g) - E_{out}(g)| > \epsilon] \leq 4 m_H(2N) e^{\frac{-\epsilon^2 N}{8}} \text{ for any } \epsilon > 0$$
$$P[|E_{in}(g) - E_{out}(g)| \leq \epsilon] \geq 1 - 4 m_H(2N) e^{\frac{-\epsilon^2 N}{8}} \text{ for any } \epsilon > 0$$
$$E_{out}(g) \leq E_{in}(g) + \sqrt{\frac{8}{N} log \frac{4 m_H(2N)}{\delta}} \text{ w.p. at least } 1 - \delta$$
$m_H(N) \leq \sum_{i=1}^{k-1} \binom{N}{i} \leq N^{k-1} + 1$, k is a break point
given finite VC dimension → g will generalize ($E_{out}(g) \approx E_{in}(g)$)
$$E_{out}(g) \leq E_{in}(g) + \sqrt{\frac{1}{2N} log \frac{2|H|}{\delta}}$$
$$E_{out}(g) \leq E_{in}(g) + \sqrt{\frac{8}{N} log \frac{4((2N)^{d_{vc}} + 1)}{\delta}}$$
$E_{out}(g) \leq E_{in}(g) + \Omega(d_{vc})$, Ω = penalty of model complexity
 # data points (N) required
set the error bar at $\epsilon = \sqrt{\frac{8}{N} ln \frac{4((2N)^{d_{vc}} + 1)}{\delta}}$
$$N = \frac{8}{\epsilon^2} ln \frac{4((2N)^{d_{vc}} + 1)}{\delta} = O(d_{vc} ln N)$$
Compute m_h - Exercise 3.12
Φ(x) = $(1, x_1^2, x_2^2)$, (a) $m_h(3) = 8$ (b) $m_h(4) < 16$
(a) draw pic (b)注意不包括偏置项为 2 维 2 维感知机最多 shatter 3 个点，所以$m_h(4) < 16$
VC Dimension d_{vc} **约等于#free parameters但不一定永远对**
tightest bound k* = smallest breakpoint k
$d_{vc} = k^* - 1$ = largest N where $m_H(N) = 2^N$
Proof that $d_{vc} = d + 1$ for perceptron in R^d
(1): exist d+1 inputs that can be shattered



for any y = $\begin{bmatrix} y_1 \\ \vdots \\ y_{d+1} \end{bmatrix}$, find **w** such that
$$sign(\mathbf{Xw}) = \mathbf{y} \iff (\mathbf{Xw}) = \mathbf{y} \overset{X\ invertible!}{\iff} \mathbf{w} = \mathbf{X}^{-1}\mathbf{y}$$

$$= E_x[bias(x) + var(x)] = bias + var$$
$\text{var}(\mathbf{x}) = \mathbb{E}_D[(g^D(\mathbf{x}) - \bar{g}(\mathbf{x}))^2]$ ← how variable is your prediction?
$= \mathbb{E}_D[g^D(\mathbf{x})^2] - \bar{g}(\mathbf{x})^2$
Example of computing Eout by given target function f:
Given $f(x) = x^2$, data distributed in interval [-1,1], consist of two points $[(x_1, x_1^2), (x_2, x_2^2)]$
By $\bar{g}(x) = E_D[g^D(x)], g(x) = ax + b$, *we have:*
$ax_1 + b = x_1^2$ and $ax_2 + b = x_2^2$
By solving it, we have $g^D(x) = (x_1 + x_2)x - x_1 x_2$
$\bar{g}(x) = E_D(x_1)x + E_D(x_2)x - E_D(x_1)E_D(x_2)$
$E_D(x^k) = \frac{1}{2}\int_{-1}^{1} x^k \, dx \rightarrow E_D(x) = 0, E_D(x^2) = \frac{1}{3}, E_D(x^4) = \frac{1}{5}$
$\rightarrow \bar{g}(x) = 0$
By using eq. of variance and bias, we have:
$var = E_x\left[E_D\left[(g^D(x) - \bar{g}(x))^2\right]\right] = E_x\left[x^2\left(\frac{1}{3} + \frac{1}{3}\right) + \left(\frac{1}{3}\frac{1}{3}\right)\right]$
$bias = E_x\left[(\bar{g}(x) - f(x))^2\right] = E_x[(-x^2)^2]$
$E_D(E_{out}(g^D)) = var + bias$
Exercise 2.7
show $P[h(x) \neq f(x)]$ can be written as an exp value of a MSE when the convention used for binary is (a) 0 or 1 ; (b)-1 or 1
(a) $P[h(x) \neq f(x)] = E[|h(x) - f(x)| = 1] = E[(h(x) - f(x))^2] = 1 \times P(|h(x) - f(x)| = 1) + 0\ P(|h(x) - f(x)| = 0)$
(b) $P[h(x) \neq f(x)] = E[|h(x) - f(x)| = 2] = \frac{1}{4}E[(h(x) - f(x))^2]$; $E[(h(x) - f(x))^2 = 4 \times P(|h(x) - f(x)| = 2) + 0\ P(|h(x) - f(x)| = 0)$
Exercise 2.8 – if H is closed under linear combination
(a) if H is closed under linear combination (any linear combination of hypotheses in H is also a hypothesis in H), then $\bar{g} \in H$
$\bar{g}(x) = \frac{1}{K}\sum_{k=1}^{K} g_k(x)$, if H is closed ..., then $\bar{g} \in H$
(b) Give a model for which the average function $\bar{g}$ is not in the model's hypothesis set
$H = \{g_1, g_2\}, g_1(x) = 0, g_2(x) = 1$
(c) expect $\bar{g}$ to be a binary function in binary classification?
no, use example above
Bias and Variance Analysis



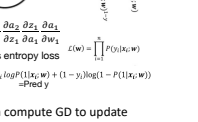
Small model(left): high bias small variance
Huge model (right): small bias high variance
High variance
- Training error is much lower than test error.
- Training error is lower than acceptable test error ε
- Test error is above ε
Solution to high variance
- Add more training data (increase N)
- Reduce model complexity
- Other technique (e.g. bagging)
High bias
- Training error is higher than acceptable test error ε
Solution to high bias: more complex model, or Boosting
Linear regression Error
in-sample error $E_{in}(h) = \frac{1}{N}\sum_{n=1}^{N}(h(x_n) - y_n)^2$
out-of-sample error $E_{out}(h) = E_x[(h(x) - y)^2]$
where $h(x) = w^T x$
$$E_{in}(w) = \frac{1}{N}\sum_{n}^{N}(\hat{y}_n - y_n)^2 = \frac{1}{N}||\hat{y} - y||_2^2$$
$$= \frac{1}{N}||Xw - y||_2^2 = \frac{1}{N}(w^T X^T X w - 2w^T X^T y + y^T y)$$
w need to be continuous, differentiable, convex
Linear Regression Algorithm
1. compute matrix X and vector y from dataset $(x_1, y_1) \dots (x_N, y_N)$, where each x includes $x_0 = 1$
2. Compute pseudo inverse X^+ of matrix X, if $X^T X$ is invertible, $X^+ = (X^T X)^{-1} X^T$
3. return $w_{lin} = X^+ y$
$$\nabla_w E_{in}(w) = \frac{2}{N}(X^T X w - X^T y)$$
Target Distribution:
Instead of y=f(x), we use P(y|x).
(x,y) is now generated by the joint distribution, P(x) P(y|x)
Noise target = deterministic target (**w*^TTx) + Noise
In-sample error vector = Xw − y =ŷ−y
Problem 3.12
In linear regression , the in sample predictions are given by $\hat{y} = Hy$, where $H = X(X^T X)^{-1} X^T$. (a) Show that H is a projection matrix, i.e. $H^2 = H$. (prove using some T and symmetric prop)
(b) $\hat{y}$ is projection of y to what space
$\hat{y} = \mathbf{H}y = X(X^T X)^{-1} X^T y = Xw_{lin}$,
$\hat{y}^T(y - \hat{y}) = y^T H^T (I - H) y = y^T (H^T - H^T H) y = y^T (H - H^2) y = 0$, so $(y - \hat{y})$ is perpendicular to y
Expected in-sample error: Ea(h-I)≥ Ein(wlin) = 1/N*E^TE = 1/N*(e^T(H-I)^T(H-I)e) = 1/N*(e^T(I-H)e) = 1/N*e^T(I-H)^2e = 1/N*e^T(I-H)e; ED[Ein(wlin)]=1/N*(ED[e^Ee]-ED[e^TPe]) ; ED[e^Ee] = Nrow^2 ; ED[sum(1toN)Hiie^2]+ED[sum(I!=j toN)Hijeiej] = trace(H)row^2+0 = (d+1)row^2; expected in-sp err = row^2(1-(d+1)/N)

E in out-sample error Problem 3.8
For $E_{out}(h) = E_{x,y}[(h(x) - y)^2]$, take derivative w.r.t. h and let it to be zero, we have
$$\frac{\partial E_{out}(h)}{\partial h} = E_{x,y}[2(h(x) - y)]$$
$$= 2E_{x,y}[h(x) - y]$$
$$= 2E_x E_{y|x}[h(x) - y]$$
To make this derivative equal to zero, it's suffice to have $E_{y|x}[h(x) - y]$ equal to 0. That is $E_{y|x}[h(x) - y] = E_{y|x}[h(x)] - E_{y|x}[y] = h(x) - E[y|x] = 0$.
Thus we obtain an optimal h, i.e. $h^*(x) = E[y|x]$.
If we treat h^* as a deterministic target function, we can write $y = h^*(x) + \epsilon(x)$ where $\epsilon(x)$ is a noise variable. Take expectation on both sides of the equation, we have
$$E[y] = E[h^*(x)] + E[\epsilon(x)]$$
$$= E[E[y|x]] + E[\epsilon(x)]$$
$$= E[y] + E[\epsilon(x)]$$
From which we derive that $E[\epsilon(x)] = 0$.

Logistic Regression
Logistic Function: Interpret as probability ($\theta = sigmoid$)
Likelihood: Plausible error that if h = f, how likely to get y from x
$$P(y \mid x) = \begin{cases} h(x) & for\ y = +1 \\ 1 - h(x) & for\ y = -1 \end{cases}$$
substitute $h(x) = \theta(w^T x)$, noting $\theta(-s) = 1 - \theta(s)$
$P(y \mid x) = \theta(y\, w^T x)$
Likelihood of $D = (x_1, y_1), \dots, (x_N, y_N)$ is
$\Pi_{n=1}^{N} P(y_n|x_n) = \Pi_{n=1}^{N} \theta(y_n w^T x_n)$
In-sample error
$$E_{in}(w) = \frac{1}{N}\sum_{n=1}^{N} \ln(1 + e^{-y_n w^T x_n})$$
Logistic regression Algorithm **(iterative solution vs IR closed form)**
1. Initialize the weights at t=0 to w(0)
2. for t = 0, 1, 2, … do **step size: too small-slow; too large-unstable; so use variable η_t**
3. compute the gradient
$$\nabla E_{in} = -\frac{1}{N}\sum_{n=1}^{N} \frac{y_n x_n}{1 + e^{-y_n w^T x_n}}$$
4. update the weights $w(t+1) = w(t) - \eta \nabla E_{in}$
5. iterate to the next step until it is time to stop
6. return the final weights w
Problem 3.10b Logistic reg and PLA with very large w
When $y_n w^T x_n > 0, \nabla e_n \approx 0, \nabla e_n \approx -y_n x_n$
ex3.5 tanh(s) = 2θ(2s) − 1
Maximize Likelihood **max ln(likelihood),加个负号变min，再把负号放进ln里**
$$\text{minimize} -\frac{1}{N} \ln\left(\Pi_{n=1}^{N} \theta(y_n w^T x_n)\right) = \frac{1}{N}\sum_{n=1}^{N} \ln\left(\frac{1}{\theta(y_n w^T x_n)}\right)$$

Neural Network – Activation Functions
a function that runs on the neurons and responsible for mapping the input of the neuron to the output
Purpose Introduces a non-linear factor to the neuron, so that the neural network can approach any non-linear function.
Sigmoid $f(x) = \frac{1}{1+e^{-x}}$ Derivative $f(x)(1 - f(x))$
Tanh $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ Derivative $1 - f(x)^2$
✓ zero-centered
✗ when approach 1, -1, rate of change becomes flat
✗ exp() is computational expensive
Relu $f(x) = \max(0, x)$ Derivative $\begin{cases} 1 & x > 0 \\ 0 & x < 0 \\ undef. & x = 0 \end{cases}$
✓ easy to compute
✓ avoid gradient vanishing/explosion

Neutral Network - Computation
Jacobian matrix
Given a function with m outputs and n inputs:
$$\frac{\partial f}{\partial x} = A = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \dots & \frac{\partial f_1}{\partial x_n} \\ \dots & \dots & \dots \\ \frac{\partial f_m}{\partial x_1} & \dots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$
Backpropagation



$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial a_2}\frac{\partial a_2}{\partial w_2} \qquad \frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial a_2}\frac{\partial a_2}{\partial z_1}\frac{\partial z_1}{\partial a_1}\frac{\partial a_1}{\partial w_1}$$
$$\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial a_2}\frac{\partial a_2}{\partial z_1}\frac{\partial z_1}{\partial a_1}\frac{\partial a_1}{\partial b_1}$$
Binary cross entropy loss $L(w) = \frac{1}{N}\sum_{i=1}^{N} y_i \log P(y_i|x_i, w)$
$J(w) = -\frac{1}{N}\sum_{i=1}^{N} y_i \log P(1|x_i, w) + (1 - y_i)\log(1 - P(1|x_i, w))$ =Pred y

Using the above derivative, we can compute GD to update
Softmax: before computing losses
$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$
σ = softmax
$\vec{z}$ = input vector
e^{z_i} = standard exponential function for input vector
K = number of classes in the multi-class classifier
e^{z_j} = standard exponential function for output vector
e^{z_j} = standard exponential function for output vector
Turn numerical output to probability distribution that add up to 1
Mean-Squared-Error (MSE) Suited for regression task of real num
$$MSE = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}$$

Gradient Descent

$$w_{t+1} = w_t - \gamma \frac{1}{n}\sum_{i\in B} \nabla_w Q(z_i, w_t)$$

For Logistic regression: $\Delta E_{in} = \eta \nabla E_{in}(w(t))^T v + O(\eta^2)$

Drawback of GD can hardly escape from local min and saddle pts

(1) hard to choose learning rate (overshoot → X converge)

(2) may converge to local minimum instead of global minimum

(3) infeasible to iterate overa ll samples in a single descent

Stochastic Gradient Descent

$$w_{t+1} = w_t - \gamma \frac{1}{|B|}\sum_{i\in B} \nabla_w Q(z_i, w_t)$$

where $B \subseteq \{1,\dots,n\}$ is a random subset, $|B|$ = batch size

Algorithm does not necessarily decrease the value of the loss at each step

while not converge

select random batch of data points

compute gradient with the batch *(formula above)*

update parameter

** this is mini-batch SGD

** batch size = 1: online SGD (immediate computation)

Why it works

(1) treating gradients of random samples as unbiased estimation of global gradient, but *(each indv step not likely aligned w/ global gradient, but avg dir of all stpes is fine)*

(2) each update much faster than GD → converge faster & better

Performance Analysis of SGD v.s. GD

convex problem (efficiency-accuracy tradeoff)

✓ less gradient evaluation per iteration

✗ need more iterations

✗ noise is a by-product

Typical strategy of **learning rate** γt : starting with some step size and gradually decrease

non-convex problem

- SGD for training deep neural networks,
- find solution faster + finds better local minima

Large Margin Classifier

Margin: the distance of the closest point to the decision boundary

Large margin ⟹ less sensitive to perturbations of the data

The location of this boundary is determined by support vectors.

Best separating hyperplane: with the max margin

$$f(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + w_0 = (\boldsymbol{w}^T\boldsymbol{x}_\perp + w_0) + r\frac{\boldsymbol{w}^T\boldsymbol{w}}{||\boldsymbol{w}||} = (\boldsymbol{w}^T\boldsymbol{x}_\perp + w_0) + r||\boldsymbol{w}||$$

Since, $f(\boldsymbol{x}_\perp) = \boldsymbol{w}^T\boldsymbol{x}_\perp + w_0$ primal $\min_{\boldsymbol{w},w_0} \frac{1}{2}||\boldsymbol{w}||^2$ s.t. $\tilde{y}_n(\boldsymbol{w}^T\boldsymbol{x}_n + w_0) \geq 1, n = 1:N$

Therefore, we have the Lagrange $\mathcal{L}(\boldsymbol{w}, w_0, \boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} - \sum_{n=1}^{N}\alpha_n(\tilde{y}_n(\boldsymbol{w}^T\boldsymbol{x}_n + w_0) - 1)$

$$r = \frac{f(\boldsymbol{x})}{||\boldsymbol{w}||}$$

$(\hat{\boldsymbol{w}}, \hat{w}_0, \hat{\boldsymbol{\alpha}}) = \min_{\boldsymbol{w},w_0}\max_{\boldsymbol{\alpha}} \mathcal{L}(\boldsymbol{w}, w_0, \boldsymbol{\alpha})$ → $\max_{\boldsymbol{\alpha}}\min_{\boldsymbol{w},w_0} \mathcal{L}(\boldsymbol{w}, w_0, \boldsymbol{\alpha})$

Lagrange dual problem

$$\nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w}, w_0, \boldsymbol{\alpha}) = \boldsymbol{w} - \sum_{n=1}^{N}\alpha_n\tilde{y}_n\boldsymbol{x}_n \quad \hat{\boldsymbol{w}} = \sum_{n=1}^{N}\hat{\alpha}_n\tilde{y}_n\boldsymbol{x}_n$$

$\sum_{n=1}^{N}\alpha_n\tilde{y}_n = 0$ and $0 \leq \alpha_n$ for $n = 1:N$ Constraints

$$\frac{\partial}{\partial w_0}\mathcal{L}(\boldsymbol{w}, w_0, \boldsymbol{\alpha}) = -\sum_{n=1}^{N}\alpha_n\tilde{y}_n$$

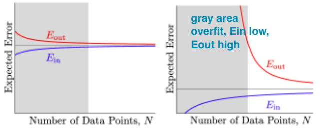
$$\mathcal{L}(\hat{\boldsymbol{w}}, \hat{w}_0, \boldsymbol{\alpha}) = \frac{1}{2}\hat{\boldsymbol{w}}^T\hat{\boldsymbol{w}} - \sum_{n=1}^{N}\alpha_n\tilde{y}_n\hat{\boldsymbol{w}}^T\boldsymbol{x}_n - \sum_{n=1}^{N}\alpha_n\tilde{y}_n w_0 + \sum_{n=1}^{N}\alpha_n = \frac{1}{2}\hat{\boldsymbol{w}}^T\hat{\boldsymbol{w}} - \hat{\boldsymbol{w}}^T\hat{\boldsymbol{w}} - 0 + \sum_{n=1}^{N}\alpha_n = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j\tilde{y}_i\tilde{y}_j\boldsymbol{x}_i^T\boldsymbol{x}_j + \sum_{n=1}^{N}\alpha_n$$

$$0 = \sum_{n=1}^{N}\hat{\alpha}_n\tilde{y}_n$$

Overfitting

d_{vc} or $E_{out} - E_{in}$ too large, bad generalization

Cause: Excessive d_{vc}, noise, limited data size N



When N is small: complex model >> gen. error → simple may win

Even know f is in H10, g10 may not win g2

Deterministic Noise and Stochastic Noise (When f is a noisy target)

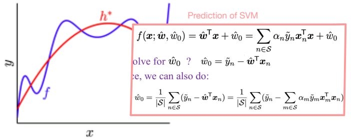
$$E_{\mathcal{D},x}\left[\left(g^{\mathcal{D}}(x) - \bar{g}(x)\right)^2\right] + E_x\left[\left(\bar{g}(x) - f(x)\right)^2\right] + E_{\epsilon,x}\left[\left(\epsilon(x)\right)^2\right]$$

var — bias (deterministic) — σ^2 (stochastic)

both cause overfit

Deterministic Noise

part of f that H cannot capture, $f(x) - h^*(x)$



diff. with stochastic noise: depends on H | fixed for given x

Solution of overfitting

(1) Early stopping when E_{out} increase

(2) regularization

(3) model selection and validation

(4) polynomial transformation

map from X space to Z space, Z = transform x to higher degree $(z = \Phi(x)$ e.g. $x \to x, x^2, \dots)$

min: $E_{in}(w) = \frac{1}{N}(Zw - y)^T(Zw - y)$, $w_{lin} = (Z^TZ)^{-1}Z^Ty$

- tradeoff: higher degree → lower Ein → worse generalization
- dimensionality of feature space increases rapidly

Regularization

Soft constraint

$$H_C = \begin{cases} h(x) = w_0 + w_1\Phi_1(x) + \dots + w_{10}\Phi_{10}(x) \\ s.t.\ \sum_{q=0}^{10} w_q^2 \leq C \end{cases}$$

How would wreg = wlin? λ=0/C=inf/C>=llwlinll^2

Regularized regression problem

min: $E_{in}(w) = \frac{1}{N}(Zw - y)^T(Zw - y)$, subject to $w^Tw \leq C$

Solving for Wreg $-\nabla E(W_{reg})$ parallel to W_{reg}; $\nabla E(W_{reg}) + (2\lambda/N)W_{reg} = 0$

$$\nabla E_{in}(W_{reg}) + \frac{2\lambda}{N}W_{reg} = 0, W_{reg} \leftarrow (Z^TZ + \lambda I)^{-1}Z^Ty$$

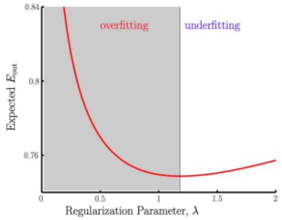
If $\lambda > 0$, then solving it will equivalent to minimizing:

$E_{aug}(w) = E_{in}(w) + \frac{\lambda}{N}w^Tw$ *(regularizer)* (ridge reg / weight decay)

Problem 4.6 Soft vs hard constraint

for the hard-order constraint and the soft-order constraint. Which more useful for binary classification using the perceptron model? If we use the hard-order constraint, with less parameters, the perceptron's VC dimension decreases, and it's less likely to classify the same amount of points with more parameters. If we use the soft-order constraint, it won't change the signs of W^Tx even when w is small. So we'll still be able to classify the points.

Effect of λ



higher λ = stronger regularization = simpler model = underfit

E_{aug} can be a good approximation of E_{out} eff dVC small if regularized

$$E_{out}(w) \leq E_{in}(w) + \Omega(H(C))$$

Problem 4.5 Augmented error and soft constraint:

If λ<0, since w^Tw is positive, to minimize Eaug, the algo will increase |w| as muchas possible, corresponding to a soft constraint of w^Tw = infinity

Model Selection L1: lwil, convex,非到处可导, sparse sol(more w=0) L2: llwill^2, convex, 可导，easy to opt)

Select by best E_{in} → prone to overfitting :(

Cross Validation

size N data → select size K as validation, size N-K as train

Leave-one-out cross validation (K=1) 用 1 pt 作为Dval

$$E_{loocv}(H, A) = \frac{1}{N}\sum_{n=1}^{N} e_n = \frac{1}{N}\sum_{n=1}^{N} err(g_n^-(x_n), y_n)$$

Model by D train

$$\mathbb{E}_{\mathcal{D}}[E_{loocv}] = \frac{1}{N}\sum_{n=1}^{N}\mathbb{E}_{\mathcal{D}_n}\mathbb{E}_{(x_n,y_n)}err(g_n^-(x_n), y_n) = \frac{1}{N}\sum_{n=1}^{N}\mathbb{E}_{\mathcal{D}_n}E_{out}(g_n^-)$$

$$= \frac{1}{N}\sum_{n=1}^{N}\overline{E_{out}}(N-1) = \overline{E_{out}}(N-1)$$

Disadv of loocv: computation cost is high → V-fold CV [K=N/V]

Training vs Validation vs Testing

Training: select among hypotheses

Validation: select among finalists

Testing: just to evaluate

Regularization vs validation

$$E_{out}(h) = E_{in}(h) + overfit\ penalty$$

Regularization: estimate quantity of overfit penalty

Validation: estimate quantity of E_{out}

Good enough (simple that fits) is Better Better Eout

Occam Razor: simplest model that fits data is the most plausible

more complex hypothesis → more likely to choose complex hypothesis → more prone to overfit

Bayesian learning

Bayes Rule $p(y|x) = \frac{p(x|y)p(y)}{p(x)}$

$$p(x) = p(x|y=1)p(y=1) + p(x|y=0)p(y=0)$$

Maximum Likelihood Estimation (MLE) w^* = argmax(w) {p(D|w)}

pick parameters that assign highest probability to training data

Example of linear regression: Wlin = arg min (y-Xw)^T(y-Xw), Equivalent to the ML solution when $y \sim N(Xw, \sigma^2 I)$

Likelihood Model

$$\text{NLL}(\boldsymbol{w}) = -\sum_{n=1}^{N}\log\left[\left(\frac{1}{2\pi\sigma^2}\right)^{\frac{1}{2}}\exp\left(-\frac{1}{2\sigma^2}(y_n - \boldsymbol{w}^T\boldsymbol{x}_n)^2\right)\right]$$

$$\sum_{n=1}^{N}(y_n - \boldsymbol{w}^T\boldsymbol{x}_n)^2 = ||\mathbf{X}\boldsymbol{w} - \boldsymbol{y}||_2^2 = (\mathbf{X}\boldsymbol{w} - \boldsymbol{y})^T(\mathbf{X}\boldsymbol{w} - \boldsymbol{y})$$

$$\hat{\boldsymbol{w}}_{mle} \triangleq \underset{\boldsymbol{w}}{\text{argmin}}\,\text{RSS}(\boldsymbol{w}) = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\boldsymbol{y}$$

For categorical distribution: Suppose we roll a K-sided dice N times, We want to estimate the probabilities from the dataset $\mathcal{D}$. $NLL(\theta) = -\sum_k N_k log\theta_k$, Nk is the number of times Y = k is observed. We have this constraint too: $\sum_{k=1}^{K}\theta_k = 1$

Prior model: assume w is Gaussian, i.e. $w \sim N(0, \lambda^{-1}I)$

Maximum A Posteriori estimation (MAP): seeks the most probable value w according to its posterior distribution:

$$\begin{aligned} w_{MAP} &= \arg\max_w \ln p(w|y, X) \\ &= \arg\max_w \ln\frac{p(y|w, X)p(w)}{p(y|X)} \\ &= \arg\max_w \ln p(y|w, X) + \ln p(w) - \ln p(y|X) \end{aligned}$$

w^* = argmax(w) {p(w|D)} = argmax(w){likelihood*prior p(w)}

The normalizing constant term ln p(y|X) doesn't involve w. Therefore, we can maximize the first two terms alone.

$$\begin{aligned} w_{MAP} &= \arg\max_w \ln p(y|w, X) + \ln p(w) \\ &= \arg\max_w -\frac{1}{2\sigma^2}(y - Xw)^T(y - Xw) - \frac{\lambda}{2}w^Tw + \text{const.} \end{aligned}$$

This is similar formula as regularization L2

Generative Learning Algorithm

generative $P(x|y)$ – model probability distribution of input

discriminative $P(y|x)$ – model probability of output given input

Multivariate normal / Multivariate Gaussian distribution

$$p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}}\exp\left(-\frac{1}{2}(x - \mu)^T\Sigma^{-1}(x - \mu)\right)$$

dimension = d, mean vector $\mu \in R^d$, covariance matrix $\Sigma \in R^{d\times d}$

Gaussian Discriminant Analysis Model (GDA):

generative classifier, class conditional densities are multivariate Gaussians:

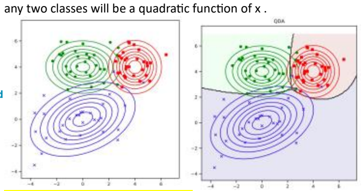
$$p(x|y = c, \theta) = N(x|\mu_c, \Sigma_c)$$

the corresponding class posterior has the form of:

$$p(y = c|x, \theta) \propto \pi_c N(x|\mu_c, \Sigma_c), \quad \text{where } \pi_c = p(y = c)$$

$$\log p(y = c|\boldsymbol{x}, \boldsymbol{\theta}) = \log\pi_c - \frac{1}{2}\log|2\pi\Sigma_c| - \frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_c)^T\Sigma_c^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_c) + \text{const}$$

this is discriminant function, e.g., the decision boundary between any two classes will be a quadratic function of x .



Linear Discriminant Analysis (LDA)

the covariance matrices are tied or shared across classes: Σ_c=Σ:

$$\begin{aligned} \log p(y = c|\boldsymbol{x}, \boldsymbol{\theta}) &= \log\pi_c - \frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_c)^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_c) + \text{const} \\ &= \log\pi_c - \frac{1}{2}\boldsymbol{\mu}_c^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_c + \boldsymbol{x}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_c + \text{const} - \frac{1}{2}\boldsymbol{x}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{x} \\ &= \gamma_c + \boldsymbol{x}^T\boldsymbol{\beta}_c + \kappa \end{aligned}$$

decision boundary: log p(y=c1|x,theta)-log p(y=c2|x,theta)=0

cancel out in db

How to fit GDA model using MLE:

$$\log p(\mathcal{D}|\boldsymbol{\theta}) = \left[\sum_{n=1}^{N}\sum_{c=1}^{C}\mathbb{I}(y_n = c)\log\pi_c\right] + \sum_{c=1}^{C}\left[\sum_{n:y_n=c}\log\mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)\right]$$

$$\hat{\boldsymbol{\mu}}_c = \frac{1}{N_c}\sum_{n:y_n=c}\boldsymbol{x}_n$$

$$\hat{\boldsymbol{\Sigma}}_c = \frac{1}{N_c}\sum_{n:y_n=c}(\boldsymbol{x}_n - \hat{\boldsymbol{\mu}}_c)(\boldsymbol{x}_n - \hat{\boldsymbol{\mu}}_c)^T$$

LDA and logistic regression:

$$p(y = c|\boldsymbol{x}, \boldsymbol{\theta}) = \frac{e^{\boldsymbol{\beta}_c^T\boldsymbol{x} + \gamma_c}}{\sum_{c'}e^{\boldsymbol{\beta}_{c'}^T\boldsymbol{x} + \gamma_{c'}}} = \frac{e^{\boldsymbol{w}_c^T[1, \boldsymbol{x}]}}{\sum_{c'}e^{\boldsymbol{w}_{c'}^T[1, \boldsymbol{x}]}}$$

$\boldsymbol{w}_c = [\gamma_c, \boldsymbol{\beta}_c]$ **multinomial logistic regression (from LDA)**

Softmax function: return probability of every class:

$$\mathcal{S}(\boldsymbol{a}) \triangleq \left[\frac{e^{a_1}}{\sum_{c'=1}^{C}e^{a_{c'}}}, \dots, \frac{e^{a_C}}{\sum_{c'=1}^{C}e^{a_{c'}}}\right]$$

$0 \leq \mathcal{S}(\boldsymbol{a})_c \leq 1$ and $\sum_{c=1}^{C}\mathcal{S}(\boldsymbol{a})_c = 1$

C is class number, a is also called logits.

LDA vs Logistic Regression: LDA and logistic regression both use the underlying linear model. Both models differ in the way they estimate the parameters.

GDA makes stronger modeling assumptions (data need to follow Gaussian distribution), and is more data efficient (i.e., requires less training data to learn "well") when the modeling assumptions are correct or at least approximately correct.

Logistic regression makes weaker assumptions, and is significantly more robust to deviations from modeling assumptions.

Nearest centroid classifier: Under additional assumption with a uniform prior over classes, we can compute the most probable class label as follows:

$$\hat{y}(\boldsymbol{x}) = \underset{c}{\text{argmax}}\log p(y = c|\boldsymbol{x}, \boldsymbol{\theta}) = \underset{c}{\text{argmin}}(\boldsymbol{x} - \boldsymbol{\mu}_c)^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_c)$$

Naïve Bayes Classifiers

Naïve Bayes Assumption: features are conditionally independent given the class label.

$$p(y = c|\boldsymbol{x}, \boldsymbol{\theta}) = \frac{p(y = c|\boldsymbol{\pi})\prod_{d=1}^{D}p(x_d|y = c, \boldsymbol{\theta}_{dc})}{\sum_{c'}p(y = c'|\boldsymbol{\pi})\prod_{d=1}^{D}p(x_d|y = c', \boldsymbol{\theta}_{dc'})}$$

Binary feature: use Bernoulli distribution and fit by MLE

Real-valued feature: use univariate Gaussian

Example of using Naïve Bayes Classifiers:

$$p(\mathcal{D}|\boldsymbol{\theta}) = \prod_{n=1}^{N}\text{Cat}(y_n|\boldsymbol{\pi})\prod_{d=1}^{D}p(x_{nd}|y_n, \boldsymbol{\theta}_d) = \prod_{n=1}^{N}\text{Cat}(y_n|\boldsymbol{\pi})\prod_{d=1}^{D}\prod_{c=1}^{C}p(x_{nd}|\boldsymbol{\theta}_{dc})^{\mathbb{I}(y_n=c)}$$

$$\hat{\mu}_{dc} = \frac{1}{N_c}\sum_{n:y_n=c}x_{nd}$$ cth class, dth feature nth training example is added if yn=c

$$\hat{\sigma}_{dc}^2 = \frac{1}{N_c}\sum_{n:y_n=c}(x_{nd} - \hat{\mu}_{dc})^2$$

so the log-likelihood is given by

$$\log p(\mathcal{D}|\boldsymbol{\theta}) = \left[\sum_{n=1}^{N}\sum_{c=1}^{C}\mathbb{I}(y_n = c)\log\pi_c\right] + \sum_{c=1}^{C}\sum_{d=1}^{D}\left[\sum_{n:y_n=c}\log p(x_{nd}|\theta_{dc})\right]$$

this decomposes into a term for π, and CD terms for each θ_{dc}

$$\log p(D|\theta) = \log p(D_y|\pi) + \sum_c\sum_d \log p(D_{dc}|\theta_{dc})$$

where $D_y = \{y_n: n = 1: N\}$ are all the labels, and $D_{dc} = \{x_{nd}: y_n = c\}$ are all the values of feature d for examples for class c. Hence we can estimate these parameters separately.

Fisher's linear discriminant analysis: To find a projection that can map the data from high-dim, e.g., xd ∈ ℝ (dim D),to a lower dim, such that the lowdimensional data can be classified as well as possible using a Gaussian class-conditional density model.

Two class case:

Derive the optimal direction w for the two-class case and project it down to one dimension. Thus, the classification can be done by placing the threshold on the line. Class- conditional mean:

$$\mu_1 = \frac{1}{N_1}\sum_{n:y_n=1}x_n, \quad \mu_2 = \frac{1}{N_2}\sum_{n:y_n=2}x_n$$

Let $m_k = \boldsymbol{w}^T\boldsymbol{\mu}_k$be the projection of each mean onto the line. also let $z_n = \boldsymbol{w}^T\boldsymbol{x}_n$be the projection of the data onto the line.

Var of projected points proportion to: $s_k^2 = \sum_{n:y_n=k}(z_n - m_k)^2$

Goal: The goal is to find w such that we maximize the distance between the means, m2-m1, while also ensuring the projected clusters are "tight", i.e., minimizing their variance. This suggests:

$$J(\boldsymbol{w}) = \frac{(m_2 - m_1)^2}{s_1^2 + s_2^2} = \frac{\boldsymbol{w}^T S_B\boldsymbol{w}}{\boldsymbol{w}^T S_W\boldsymbol{w}}$$ Between class scatter matrix $S_B = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T$

$S_W = \sum_{n:y_n=1}(X_n - \mu_1)(X_n - \mu_1)^T$ Within class scatter matrix

Max of J(w) when $S_B w = \lambda S_W w, \lambda w = S_W^{-1}(\mu_2 - \mu_1)(m_2 - m_1)$

Example:

Class 1 X1=(x1,x2)={(4,2),(2,4),(2,3),(3,6),(4,4)} 这个方向上的

Class 2 X2=(x1,x2)={(9,10),(6,8),(9,5),(8,7),(10,8)} 等于所有x1*x2的和

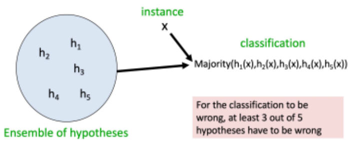
$S_1 = \sum_{x\in c_1}(x - \mu_1)(x - \mu_1)^T \quad S_2$ same

$S_w = S_1 + S_2$ LDA sol: $w^* \propto S_w^{-1}(\mu_2 - \mu_1) = xxx$

Ensemble learning

Definition: method to select and combine an ensemble of hypotheses into a (hopefully) better hypothesis. Perceptrons: linear separators, Ensemble of perceptrons: polytope

Bagging: Majority voting



Bagging assumption: Each hi makes error with probability p; The hypotheses are independent

Majority voting of n hypotheses:

k hypotheses make an error: $\binom{n}{k}p^k(1-p)^{n-k}$

Majority makes an error: $\sum_{k>n/2}\binom{n}{k}p^k(1-p)^{n-k}$

Example: n = 5, p = 0.1, the error of majority is than 0.01!

Weighted Majority: In practice hypotheses are rarely independent, some of them have less errors too

Intuition of weighted majority: Decrease weights of correlated hypotheses, increase weights of good hypotheses

Boosting: Computes a weighted majority, Can boost a weak learner, Operates on a weighted training set 每轮加一个新的h分类器

Weighted training set: minimize training error in supervised learning, bias algorithm to learn correct instances of high weights

Idea: when instance is misclassified by a hypothesis, increase its weight so the next hypothesis is more likely to classify it correctly.

Boosting framework: Set all instance weights w_x to be the same→

Repeat: h_i⟵learn(dataset, weights), Increase w_x of misclassified instances x, **until** the sufficient number of hypotheses

Ensemble hypothesis is the weighted majority of h_i's with weights w_x proportional to the accuracy of h_i

What can we boost?: weak learner that produces hypotheses at least as good as random classifier, e.g. Perceptrons, Naïve Bayes

Advantage of boosting: No need to learn a perfect hypothesis, Can boost any weak learning algorithm, Boosting is very simple to program, Good generalization

Unsupervised learning have data, to infer the latent structure underlying the data.

Clustering: Grouping data points into clusters, with no labels

K-mean clustering: Assume the data x lives in a Euclidean space, the data belongs to K clusters, Data points from the same cluster are similar to each other, i.e., close in the Euclidean distance.

Initialization: randomly initialize cluster centers

The algorithm iteratively alternates between two steps:

Assignment step: Assign each data point to the closest cluster

Refitting step: Move each cluster center to the center of gravity of the data assigned to it

K-means Objective:

Find cluster centers **m** and assignments **r** to minimize the sum of squared distances of data points $\{\mathbf{x}^{(n)}\}$ to their assigned cluster centers

$$\min_{\{\mathbf{m}\},\{\mathbf{r}\}} J(\{\mathbf{m}\},\{\mathbf{r}\}) = \min_{\{\mathbf{m}\},\{\mathbf{r}\}}\sum_{n=1}^{N}\sum_{k=1}^{K}r_k^{(n)}||\mathbf{m}_k - \mathbf{x}^{(n)}||^2$$

$$\text{s.t.} \sum_k r_k^{(n)} = 1, \forall n, \quad \text{where } r_k^{(n)} \in \{0, 1\}, \forall k, n$$

where $r_k^{(n)} = 1$ means that $\mathbf{x}^{(n)}$ is assigned to cluster k (with center $\mathbf{m}_k$)

Repeat until convergence (until assignments do not change):

- Assignment: Each data point $\mathbf{x}^{(n)}$ assigned to nearest mean

$$\hat{k}^n = \arg\min_k d(\mathbf{m}_k, \mathbf{x}^{(n)})$$

(with, for example, L2 norm: $\hat{k}^n = \arg\min_k ||\mathbf{m}_k - \mathbf{x}^{(n)}||^2$)

and Responsibilities (1-hot encoding)

$$r_k^{(n)} = 1 \longleftrightarrow \hat{k}^{(n)} = k$$

- Refitting: Model parameters, means are adjusted to match sample means of data points they are responsible for:

$$\mathbf{m}_k = \frac{\sum_n r_k^{(n)}\mathbf{x}^{(n)}}{\sum_n r_k^{(n)}}$$

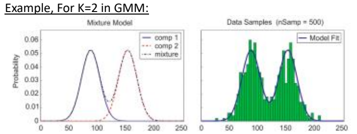
Local minimum => run 多几次

Generative view of clustering: The data is produced by a generative model, then adjust the model parameter to maximize the probability that it produces exactly the data we observed

Gaussian Mixture Model (GMM): represents a distribution

$p(x) = \sum_{k=1}^{K}\pi_k N(x|\mu_k, \Sigma_k)$, where $\sum_{k=1}^{K}\pi_k = 1$, $\pi_k \geq 0\ \forall k$

Example, For K=2 in GMM:



We can interpret mixing coefficients in GMM as prior probability:

$p(x) = \sum_{k=1}^{K}p(k)p(x|k)$

MLE of Mixture parameter: Joint distribution(for all N data points)

$$p(X|\pi, \mu, \Sigma) = \prod_{n=1}^{N}\sum_{k=1}^{K}\pi_k N(x_n|\mu_k, \Sigma_k)$$

$$ln\ p(X|\pi, \mu, \Sigma) = \sum_{n=1}^{N}\ln\left\{\sum_{k=1}^{K}\pi_k N(x_n|\mu_k, \Sigma_k)\right\}$$

→ log of sum → very messy

$$\mathbb{E}[z_{nk}] = \frac{\sum_{z_{nk}} z_{nk}[\pi_k\mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)]^{z_{nk}}}{\sum_{z_{nj}}[\pi_j\mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)]^{z_{nj}}} = \frac{\pi_k\mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K}\pi_j\mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} = \gamma(z_{nk})$$

Expectation Maximization Algorithm (EM): suppose some oracle told us which point comes from which Gaussian, by providing a "latent" variable z_{nk} which is 1 if point n comes from the k-th component Gaussian, and 0 otherwise

An alternative view of EM: The goal of EM is to find th maximum likelihood sol for models having latent variables.We can consider instead its expected value under posterior distribution of latent variable in E step and max the expectation in M step.

$\mathbb{E}_{\mathbf{Z}}[\ln p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi})] = \sum_{n=1}^{N}\sum_{k=1}^{K}\gamma(z_{nk})\{\ln\pi_k + \ln\mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)\}$ The expected value of the complete-data log likelihood function

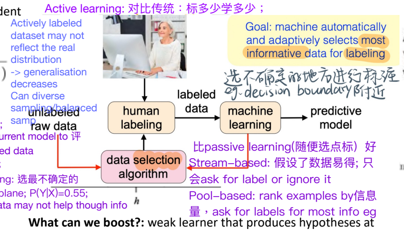
$$\ln p(\mathbf{X}|\boldsymbol{\theta}) = \ln\left\{\sum_{\mathbf{Z}}p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta})\right\}$$

In the E step, we use the current parameter values θold to find the posterior distribution of the latent variables given by p(Z|X, θold). then use this posterior distribution to find the expectation of the complete-data log likelihood evaluated for some general parameter value θ. This expectation, denoted Q(θ, θold)

• In the M step, we determine the revised parameter estimate θnew by maximizing this function

$$\boldsymbol{\theta}^{\text{new}} = \arg\max_{\boldsymbol{\theta}}\mathcal{Q}(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}}) = \sum_{\mathbf{Z}}p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{\text{old}})\ln p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta})$$

Active learning: 对比传统：标多少学多少；

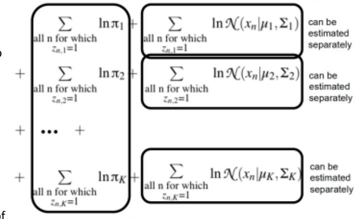


Again, if an oracle gave us the values of the latent variables (component that generated each point) we could work with the complete log likelihood

$$p(\mathbf{X}, \mathbf{Z}|\mu, \Sigma, \pi) = \prod_{n=1}^{N}\prod_{k=1}^{K}\pi_k^{z_{nk}}\mathcal{N}(\mathbf{x}_n|\mu_k, \Sigma_k)^{z_{nk}}$$

$$\ln p(\mathbf{X}, \mathbf{Z}|\mu, \Sigma, \pi) = \sum_{n=1}^{N}\sum_{k=1}^{K}z_{nk}\{\ln\pi_k + \ln\mathcal{N}(\mathbf{x}_n|\mu_k, \Sigma_k)\}.$$

note: for a given n, there are k of these latent variables, and ONLY ONE of them is 1 (all the rest are 0)



these are coupled because the mixing weights all sum to 1, but it is no big deal to solve

Since we don't know the latent variables, we instead take the **expected value** of the log likelihood with respect to their posterior distribution P(z|x,θ). In the GMM case, this is equivalent to "softening" the binary latent variables to continuous ones (the expected values of the latent variables)

$$\ln p(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta}) = \sum_{n=1}^{N}\sum_{k=1}^{K}z_{nk}\{\ln\pi_k + \ln\mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)\}$$

unknown discrete value 0 or 1

$$\mathbb{E}_{\mathbf{z}}[\ln p(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta})] = \sum_{n=1}^{N}\sum_{k=1}^{K}\gamma_k(\mathbf{x}_n)\{\ln\pi_k + \ln\mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)\}$$

known continuous value between 0 and 1

Where $\gamma_j(\mathbf{x}_n)$ is $P(z_{nk} = 1)$

EM algorithm overview:

Optimization uses the Expectation Maximization algorithm, which alternates between two steps:

1. E-step: Compute the posterior probability that each Gaussian generates each datapoint (as this is unknown to us)

$$\gamma_k = p(z = k|x) = \frac{p(z = k)p(x|z = k)}{p(x)}$$

Responsibility/P(x belongs to kth gaussian)

$$= \frac{p(z = k)p(x|z = k)}{\sum_{j=1}^{K}p(z = j)\ p(x|z = j)}$$

Weighted sum of prob(x belongs to jth gaussian)

$$= \frac{\pi_k N(x|\mu_k, \Sigma_k)}{\sum_{j=1}^{K}\pi_j N(x|\mu_j, \Sigma_j)}$$

2. M-step: Assuming that the data really was generated this way, change the parameters of each Gaussian to maximize the probability that it would generate the data it is currently responsible for. At the optimum we satisfied:

$$\frac{\partial\ln p(\mathbf{X}|\pi, \mu, \Sigma)}{\partial\mu_k} = 0 = \sum_{n=1}^{N}\frac{\pi_k\mathcal{N}(\mathbf{x}^{(n)}|\mu_k, \Sigma_k)}{\sum_{j=1}^{K}\pi_j\mathcal{N}(\mathbf{x}^{(n)}|\mu_j, \Sigma_j)}\Sigma_k(\mathbf{x}^{(n)} - \mu_k)$$

- Initialize the means μ_k, covariances Σ_k and mixing coefficients π_k
- Iterate until convergence:
  - E-step: Evaluate the responsibilities given current parameters

$$\gamma_k^{(n)} = p(z^{(n)}|\mathbf{x}) = \frac{\pi_k\mathcal{N}(\mathbf{x}^{(n)}|\mu_k, \Sigma_k)}{\sum_{j=1}^{K}\pi_j\mathcal{N}(\mathbf{x}^{(n)}|\mu_j, \Sigma_j)}$$

  - M-step: Re-estimate the parameters given current responsibilities

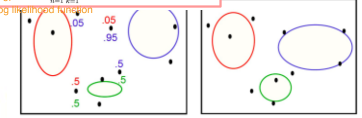
$$\mu_k = \frac{1}{N_k}\sum_{n=1}^{N}\gamma_k^{(n)}\mathbf{x}^{(n)}$$

$$\Sigma_k = \frac{1}{N_k}\sum_{n=1}^{N}\gamma_k^{(n)}(\mathbf{x}^{(n)} - \mu_k)(\mathbf{x}^{(n)} - \mu_k)^T$$

$$\pi_k = \frac{N_k}{N} \quad \text{with} \quad N_k = \sum_{n=1}^{N}\gamma_k^{(n)}$$

  - Evaluate log likelihood and check for convergence

$$\ln p(\mathbf{X}|\pi, \mu, \Sigma) = \sum_{n=1}^{N}\ln\left(\sum_{k=1}^{K}\pi_k\mathcal{N}(\mathbf{x}^{(n)}|\mu_k, \Sigma_k)\right)$$



Calculation Formulas

$$\frac{d\ w^Txw}{dx} = w^Tw\ , \quad \frac{d\ w^Tx}{dx} = w, \quad \frac{d\ w^Txw}{dw} = x + w$$

$$||x||^2 = x^Tx$$