ML (Weak AI)

- Supervised
  - Eg: Face Detection, Identification of numbers, animals etc.
- Unsupervised
  - Eg: classify photos w+w/o focus. without explicit labels
  - E-commerce - classify customers in groups
- Reinforcement
  - Learning to perform actions in an environment
  - eg: Self-driving cars, Computer Games, Robotics.

Measuring AI - Turing Test

ML initially started as a quest for AI & mimicking brain dynamics, but has now evolved to solving practical problems using algos which have little or no resemblance to brains.

Debate: AI ⊂ ML or ML ⊂ AI or AI ∦ ML just have an overlap.

Goal of ML → Generalisation of learning to new & unknown datasets & scenarios (AGI).
↳ pre-trained + last layer

often requires domain knowledge

Bias-Variance Tradeoff / Decomposition - best way to estimate generalisation error.

Bias: wrong assumption about training data (underfitting)
Variance: High sensitivity to small fluctuation & random noise. (this overfitting)

$y = f(x) + \epsilon$ $\hat{f}$: Learned function
↳ Actual function.

ϵ: zero mean σ^2 variance.

$$\text{Bias} = E_D[\hat{f}] - f$$

$$\text{Variance} = E_D\left[\left(E_D[\hat{f}] - \hat{f}\right)^2\right]$$

Expected Error on unseen sample:

$$\text{Bias}^2 + \text{Var} + \sigma^2$$
↳ Var of y or ϵ

Random Variables & Probability Distributions

Variable · whose value can change.

- Deterministic [governed by deterministic equations]
- Random [Value is not pre-determined] can change in unpredictable ways but has a probability distribution.
  - Discrete
  - Continuous

Discrete R.V: Coin Toss $X = \begin{cases} 1 & \text{Heads} \\ 0 & \text{Tails} \end{cases}$

Bernoulli Distribution $\leftarrow$ $P(X=1) = p$, $P(X=0) = q = 1-p$, $0 \le p \le 1$

Dice Roll.

Binomial Distribution : N coin tosses

$$P_r(X=k) = f(k,n,p) = {}^nC_k\, p^k (1-p)^{n-k}$$

WRITE CODE to plot this for large 'n'.

Poisson Distribution : $f(k;\lambda) = \dfrac{\lambda^k e^{-\lambda}}{k!} = P_r(X=k)$ [Number of events in a (fixed) time interval]

$$E[X] = \lambda = \text{Var}[X]$$

$$\Omega = \{x_1, x_2, x_3, \cdots, x_n\}$$

$$0 \le P(X = x_i) \le 1$$

$$\sum_{i=1}^{n} P(X = x_i) = 1.$$

Continuous R.V.

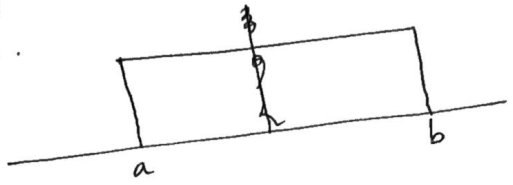
Any measuring device has finite precision & range

when outcome can be any real number [uncountable].

probability of X taking any one value is ZERO.

Practical : when no. of possibilities is very large

Uniform Distribution :



Distribution $f(x=x) = \frac{1}{b-a}$

$$P(X=x) = 0$$

$$P(x \le X \le x+\Delta x) = f(X=x)\,\Delta x, \text{ if } \Delta x \approx 0$$

$$\int_{-\infty}^{\infty} f(X=x)\,dx = 1$$

$$f(X=x) \ge 0$$

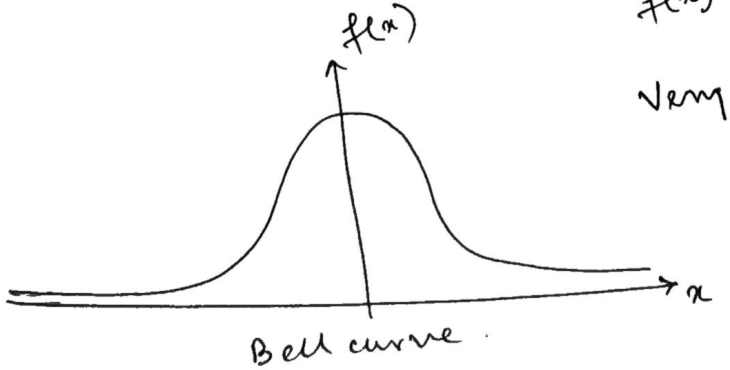
~~$0 \le f(X=x) \le 1$~~ $f(X=x)$ can be greater than 1 over a finite range of x values.

Gaussian :

Velocity of gas molecules
Noise in communication channels
Height of a large population.

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right]$$

Very crucial in ML too!!



Bell curve.

Expectation, Moments & CLT

Expectation: $\mathbb{E}[g(x)] = \int_{-\infty}^{\infty} g(x) f(x)\, dx$

Mean: $\mathbb{E}[x] = \int_{-\infty}^{\infty} x f(x)\, dx = \mu$

Variance: $\sigma^2 = \mathbb{E}[(x-\mu)]^2 = \mathbb{E}[x^2] - \mathbb{E}[x]^2$

Moments: $m_n = \mathbb{E}[x^n] = \int_{-\infty}^{\infty} x^n f(x)\, dx$

Characteristic function: $\Phi_x(\omega) = \int_{-\infty}^{\infty} f(x)\, e^{j\omega x}\, dx = \mathbb{E}[e^{j\omega x}]$

Moment Generating function (MGF): $M_x(s) = \int_{-\infty}^{\infty} f(x) e^{sx}\, dx = \mathbb{E}[e^{sx}]$

$M_x^{(n)}(0) = \mathbb{E}[x^n] = m_n$

If all moments are finite & the following series converges absolutely, then

$M_x(s) = \sum_{n=0}^{\infty} \frac{m_n}{n!} s^n$ [does not work for Log-normal distribution]

If X & Y are independent random variables,

$\Phi_{X+Y}(\omega) = \mathbb{E}[e^{j\omega(x+y)}]$

$= \mathbb{E}[e^{j\omega x}]\,\mathbb{E}[e^{j\omega y}] = \Phi_x(\omega)\,\Phi_y(\omega)$

MGF of Normal Distribution

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2} \qquad \int_{-\infty}^{\infty} f(x)\,dx = 1$$

$$M(s) = \mathbb{E}[e^{sx}] = \int_{-\infty}^{\infty} e^{sx} f(x)\,dx$$

$$= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{sx} e^{-x^2/2}\,dx$$

$$= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-(x-s)^2/2} e^{s^2/2}\,dx$$

$$= e^{s^2/2} \left[\frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-(x-s)^2/2}\,dx \right] = e^{s^2/2}$$

Central Limit Theorem

$$\bar{X}_n = \frac{X_1 + X_2 + \cdots + X_n}{n} \qquad \{X_i\} \text{ i.i.d. rv}$$

Let $Y_i = X_i - \mu$ & $Z_n = \dfrac{Y_1 + Y_2 + \cdots + Y_n}{\sqrt{n\sigma^2}}$

$$M_Y(s) = 1 + m_1 s + \frac{m_2}{2!} s^2 + \cdots = 1 + \frac{\sigma^2}{2} s^2 + \cdots$$

$$M_{Z_n}(s) = \left[M_Y\left(\frac{s}{\sqrt{n\sigma^2}} \right) \right]^n = \left[1 + \frac{\sigma^2}{2} \left(\frac{s}{\sqrt{n\sigma^2}} \right)^2 + \cdots \right]^n$$

$$= \left[1 + \frac{s^2}{2n} + \cdots \right]^n \sim e^{s^2/2} \quad \text{for large } n$$

BAYES' THEOREM

$U = [A_1, A_2, \cdots, A_n]$ is a partition of S, meaning A_i & A_j are mutually exclusive & $\sum_i P(A_i) = 1$

Let B be an arbitrary event:

$$\therefore B = B \cap S = B \cap [A_1 \cup A_2 \cup A_3 \cdots \cup A_n]$$

$$= (B \cap A_1) \cup (B \cap A_2) \cdots \cup (B \cap A_n)$$

BA_i & BA_j are mutually exclusive $\therefore P(B) = P(B \cap A_1) + P(B \cap A_2) + \cdots + P(B \cap A_n)$

Total probability theorem: $P(B \cap A_i) = P(B/A_i)\, P(A_i)$

$$= P(A_i/B_i)\, P(B_i)$$

$$\Rightarrow P(A_i/B) = \frac{P(B/A_i)\, P(A_i)}{P(B)}$$

$$= \frac{P(B/A_i)\, P(A_i)}{\sum_i P(B/A_i)\, P(A_i)}$$

BAYES' THEOREM

Example: A test for cancer is known to be 95% accurate. A person submits to test & is found positive. Let the person belong to a population of 1L people out of which 2000 actually have the disease. What's the prob. of the person being actually infected?

$P(T_+/C) = 0.95$ $\quad P(T_-/H) = 0.95$

$P(H) = 0.98$ $\quad P(C) = 0.02$

$$P(C/T_+) = \frac{P(T_+/C)\, P(C)}{P(T_+)} = \underline{0.278}.$$

$$P(T_+) = P(T_+/C) P(C) + P(T_+/H) P(H)$$

~~307~~

Example: Papoulis page 34

$$P(D/B_1) = \frac{100}{2000} = 0.05 \qquad P(D/B_2) = \frac{200}{500} = 0.4$$

$$P(D/B_3) = \frac{100}{1000} = 0.1 \qquad P(D/B_4) = \frac{100}{1000} = 0.1$$

$$P(B_1) = P(B_2) = P(B_3) = P(B_4) = 1/4$$

$$P(D) = 0.1625$$

$$P(B_2/D) = \frac{0.4 \times 0.25}{0.1625} = 0.615$$

Example Papoulis page 104

n coin tosses k heads

Prob of head in $(n+1)$th toss?

Ans: $\frac{k+1}{n+2}$

Naive Bayes & Gaussian Naive Bayes

Bayes' Theorem

$U = [A_1, A_2, \ldots, A_n]$ is a partition of S
& B is an arbitrary event.

$$P(A_i/B) = \frac{P(B/A_i)\,P(A_i)}{\sum_i P(B/A_i)\,P(A_i)} = \frac{P(B/A_i)P(A_i)}{P(B)}$$

Dataset of 1L emails with 70% Normal & 30% Spam.
Find total count of each occuring words in each category

BAG of WORDS model

w_1 = work
w_2 = deadline
w_3 = hand
w_4 = bonus
w_5 = luxury
w_6 = Money
(SPAM TRIGGER WORDS)
Total 100 words in each category

#words / #docs — Two ways to calculate

$p(w_1/N) = \frac{22}{100} = 0.22$
$p(w_2/N) = \frac{26}{100} = 0.26$
$p(w_3/N) = \frac{19}{100} = 0.19$
$p(w_4/N) = \frac{13}{100} = 0.13$
$p(w_5/N) = \frac{8}{100} = 0.08$
$p(w_6/N) = \frac{12}{100} = 0.12$

$p(w_1/S) = \frac{11}{100} = 0.11$
$p(w_2/S) = \frac{4}{100} = 0.04$
$p(w_3/S) = \frac{14}{100} = 0.14$
$p(w_4/S) = \frac{31}{100} = 0.31$
$p(w_5/S) = \frac{21}{100} = 0.21$
$p(w_6/S) = \frac{19}{100} = 0.19$

If $p(w_i/N) = 0$ & $p(w_i/S) \neq 0$ or reverse, add 1 to all word counts

New email received with words w_3, w_5 & w_6 [in any order]

Naive ≡ [we also assume word occurrence to be independent events]

$$P(N/w_3 \cap w_5 \cap w_6) = \frac{P(w_3 \cap w_5 \cap w_6/N)\,P(N)}{P(w_3 \cap w_5 \cap w_6)} \quad \leftarrow \text{Prior probability}$$

High Bias – ignores word order
Low variance ↳ works well

$$= \frac{P(w_3/N)\,P(w_5/N)\,P(w_6/N)\,P(N)}{P(w_3 \cap w_5 \cap w_6)}$$

$\propto 0.19 \times 0.08 \times 0.12 \times \frac{0.7}{0.5} = 0.0012768$ ~~0.000912~~

$$P(S/w_3 \cap w_5 \cap w_6) = \frac{P(w_3/S)\,P(w_5/S)\,P(w_6/S)\,P(S)}{P(w_3 \cap w_5 \cap w_6)}$$

$\propto 0.14 \times 0.21 \times 0.19 \times \frac{0.3}{0.5} = 0.0016758$ ~~0.002793~~

[Take log since numbers get very small ↳ underflow]

Gaussian Naive Bayes

Estimating $P(x_i/c_j)$ using Gaussian Distribution

used when 'x_i' values can be any real number in a range.

Eg: Boy vs. Girl classification using height, weight & feet size.

fit a Gaussian using mean & SD.

Use mean & variance of available data to fit Gaussian.

Bernoulli Naive Bayes : use Bernoulli distribution

$x_i \in \{0, 1\}$

word present or absent

Categorical Naive Bayes

Multinomial Naive Bayes : $x_i \in \{0, 1, 2, \ldots, K\}$

word count taken into account.

Bayes optimal classifier

Model parameter dependencies leading to the most general classifier. Abstract notion & hard to implement in practice.

Supervised Learning : Regression vs. Classification

ML is another form of ~~cost~~ curve fitting, or function approximation, but where the functional form of the curve is not explicitly obvious.
[not binary but gray scale]

Linear Regression ⟶ Linear classification Models
[Logistic, Linear SVM]
↓
~~Probabilistic~~ Non-linear classification models
(SVM with kernels, Decision Trees)
↓
ANN ⟶ Deep Learning.

Model encompassing a broad family of functions & changing parameter helps in navigating this set of functions.

Learning = Training + Testing
↑
Generalisation

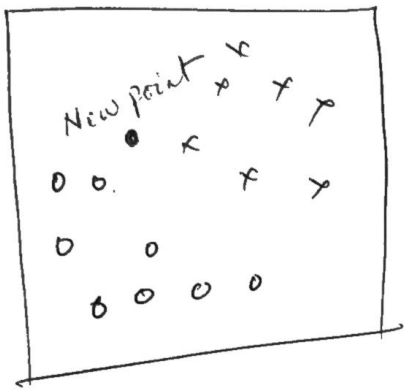
Underfitting : High Bias & Low variance
Overfitting : Low Bias & High Variance
↳ reduce model complexity
use regularisation

	Testing Error High	Testing Error Low
Training Error High	Underfitting	Fluke
Training Error Low	overfitting	Ideal Good fit

- ~~Naive Bayes'~~ & ~~Bayes'~~ optimal classifier
- Maximum Likelihood Estimator & ~~Maximum Aposteriori Probability~~
- Linear Regression
- Logistic Regression
- Support Vector Machines
- Decision Trees

~~Boosting~~
- K-nearest Neighbours
- ANNs.

K- Nearest Neighbours



Measure distance from K- nearest neighbours & classify based on majority rule.

Curse of Dimensionality

Consider a D-dimensional unit cube.

If we want to do our KNN estimate based on a fraction, f, of data points, we need to take a cube around our test point of edge size,

$$e_D(f) = f^{1/D}$$

If $f = 1\% = 0.01$,

$D = 3 \Rightarrow e_D(f) = 0.215$

$D = 10 \Rightarrow e_D(f) = 0.63 \rightarrow$ very large cube!!

$D = 100 \Rightarrow e_D(f) = 0.96!!!$

Blessing of Dimensionality

If Signal to Noise ratio is high.

High dimensional data can have very interesting & geometric properties which could be exploited for improving accuracy.

KNN - Algo - pseudo - code

Actual \ Predicted	0	1
0	TN	FP
1	FN	TP

Accuracy -

$$\frac{TN + TP}{Total}$$

$$Precision = \frac{TP}{TP + FP}$$

$$F1 = \frac{2}{1/Prec + 1/Recall}$$

$$Recall = \frac{TP}{TP + FN}$$ (Sensitivity) : True Positive Rate

$$Specificity = \frac{TN}{TN + FP}$$: True Negative Rate

~~Sensitivity~~

BAYESIAN PARAMETER ESTIMATION

Event A : k Heads in n tosses

p : probability of getting Head in one toss

Event A' : Head in $(n+1)$th toss

Since we do not know the value of 'p' & are trying to estimate it, we need to consider it as a random variable.

Before event A, lets say we have no information about 'p'

So, assume $f(p) = \begin{cases} 1, & 0 \le p \le 1 \\ 0, & \text{otherwise} \end{cases}$

$$P(A/p) = p^k (1-p)^{n-k}$$

$$f(p/A) = \frac{P(A/p)\, f(p)}{P(A)} = \frac{P(A/p)\, f(p)}{\int_0^1 P(A/p) f(p)\, dp}$$

$$= p^k (1-p)^{n-k} \cdot \frac{(n+1)!}{(n-k)!\, k!} \qquad 0 \le p \le 1$$

$\sim \beta(\alpha, \beta)$ Beta-distributed $\alpha = n$, $\beta = k$ here.

$$P(A'/A) = \int_0^1 \underbrace{P(A'/p)}_{p} f(p/A)\, dp$$

$$= \frac{k+1}{n+2}$$

This is generally very complicated to estimate in most problems. Hence, we instead estimate the value of 'ρ' that best explains the observed data.

There are two ways of doing it.

MLE & MAP

MLE: Assume that you have no prior info about the parameters [uniform distribution]

MAP: Assume availability of prior info based on reasonable considerations. also leads to "regularisation" which helps in preventing overfitting.

(x, y) Training Data

$\tilde{x}$ new input data

$\tilde{y}$ predicted output data.

$$p(\tilde{y}/x, y, \tilde{x}) = \int \underbrace{p(\tilde{y}/\rho, \tilde{x})}_{\text{model}} p(\rho/x, y)\, d\rho$$

$$p(\rho/x, y) = \frac{p(y/x, \rho)\, p(\rho)}{p(y/x)}$$

$$= \frac{p(y/x, \rho)\, p(\rho)}{\int p(y/x, \rho)\, p(\rho)\, d\rho}$$

BAYESIAN PARAMETER ESTIMATION.

MLE & MAP : $p(\tilde{y}/x, y, \tilde{x}) = p(\tilde{y}/\rho_E, \tilde{x})$

$$\rho_E = \arg\max_{\rho} p(\rho/x, y)$$

MLE & Regression

Let $y_i = g(x_i, \theta) + \epsilon_i$ → curve fitting

$$\theta_{MLE} = \underset{\theta}{\text{argmax}} \sum_i \log p(y_i | x_i, \theta)$$

$$= \underset{\theta}{\text{argmax}} \sum_i \log p(\epsilon_i)$$

$$= \underset{\theta}{\text{argmax}} \sum_i \log \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\epsilon_i^2/2\sigma^2}$$: Assuming Gaussian Error/Noise

$$= \underset{\theta}{\text{argmax}} \left[\sum_i \underbrace{\log \frac{1}{\sqrt{2\pi\sigma^2}}}_{\text{constant}} - \frac{1}{2\sigma^2} \sum_i \epsilon_i^2 \right]$$

$$= \underset{\theta}{\text{argmax}}\ \underset{\theta}{\text{argmin}} \sum_i \epsilon_i^2$$

$$= \underset{\theta}{\text{argmin}} \sum_i \left[y_i - g(x_i, \theta) \right]^2$$

Least Squares Error Cost function.

MLE & Classification

Predict output $y \in \{0, 1\}$ → Disease / Healthy; Bank default / pay; Fake news / factual

using a probabilistic function

$g(x, \theta) = P[y(x) = 1]$ ↳ could be Logistic Regression or ANN, etc.

$$\theta_{MLE} = \underset{\theta}{\text{argmax}}\ p(y_i | x_i, \theta)$$

$$= \underset{\theta}{\text{argmax}} \prod_i g_i^{y_i} (1 - g_i)^{1 - y_i}$$

$$= \underset{\theta}{\text{argmax}} \sum_i \left[y_i \log g_i + (1 - y_i) \log (1 - g_i) \right]$$

used in Logistic Regression

$$= \underset{\theta}{\text{argmin}} \left(- \right) \sum_i y_i \log g_i + (1 - y_i) \log (1 - g_i)$$

Binary Cross-Entropy

MLE ~~& MAP~~ Maximum Likelihood Estimate

Naive Bayes assumes independence of parameter which is generally not true in real world.
So, need to build models & estimate parameter.

$$f_{\phi}(\theta|x,y) = \frac{p(y|x,\theta) f_{\phi}(\theta)}{p(y|x)}$$

$p(y|x,\theta)$ → Likelihood; $f_{\phi}(\theta)$ → prior, $p(\theta|x) = p(\theta)$

$f_{\phi}(\theta|x,y)$ — Posterior estimate

MLE

Assume $f_{\phi}(\theta)$ to have uniform distribution
i.e. no prior knowledge of parameters available.

$$\therefore \theta_{MLE} = \underset{\theta}{\text{argmax}}\, f_{\phi}(\theta|x,y)$$

$$= \underset{\theta}{\text{argmax}}\, p(y|x,\theta) \xrightarrow{} L$$

$$= \underset{\theta}{\text{argmax}} \prod_i p(y_i|x_i,\theta)$$

$$= \underset{\theta}{\text{argmax}} \sum_i \log p(y_i|x_i,\theta) \rightarrow \text{Log likelihood}$$

Coin toss experiment

ρ = prob. of heads (1) tails = 0

$$p(Y_i = y_i|\rho) = \rho^{y_i}(1-\rho)^{1-y_i} \qquad y_i \in \{1,0\}$$

↳ Bernoulli dist.

$$LL = \log p(y|\rho)$$

$$= \sum_{i=1}^{N} \log p(y_i|\rho) = \sum_{i=1}^{N} \log \rho^{y_i}(1-\rho)^{1-y_i}$$

$$= \log\rho \sum y_i + \log(1-\rho) \sum (1-y_i)$$

$$= n^{(1)} \log\rho + n^{(0)} \log(1-\rho)$$

$$\frac{\partial LL}{\partial \rho} = 0 \Rightarrow \frac{n^{(1)}}{\rho} - \frac{n^{(0)}}{1-\rho} = 0 \Rightarrow \frac{1}{\rho} - 1 = \frac{n^{(0)}}{n^{(1)}} \Rightarrow \rho_{MLE} = \frac{n^{(1)}}{n^{(1)} + n^{(0)}}$$

[Intuitive Estimate is MLE]

MAP Estimation - Maximum a posteriori

$$f_p(\theta/y,x) = \frac{p(y/x,\theta)\, f_p(\theta/x)}{p(y/x)} \qquad \leftarrow p(\theta)$$

$$\theta_{MAP} = \text{argmax}\ f_p(\theta/y,x)$$

$$= \text{argmax}\ p(y/x,\theta)\, f_p(\theta)$$

$$= \text{argmax}\left[\sum_i \log p(y_i/x_i,\theta) + f_p(\theta)\right]$$

Usually clear info of $p(\theta)$ is not available.
So use $f_p(\theta)$ to enforce certain constraints.
[regularisation]

Eg: choose $f_p(\theta)$ such that there is low probability for θ taking high values.

Now Lets say we wish to make a prediction for new data point

$$p(\tilde{y}/\tilde{x},y,x) = p(\tilde{y}/\tilde{x},\theta) \rightarrow \theta_{MAP} \text{ or } \theta_{ML}$$

Usually,

$$p(\tilde{y}/\tilde{x},y,x) = \int p(\tilde{y}/\tilde{x},\theta)\, f_p(\theta/y,x)\, d\theta$$

But actually, $p(\tilde{y}/\tilde{x},y,x)$

→ usually we estimate this just using θ_{ML} or θ_{MAP} which is not same as RHS.
But estimating RHS is very hard since it requires $p(x)$ too.

MAP for Coin Toss

$$\theta_{MAP} = \text{argmax} \sum_{i=1}^{N} \log p$$

MAP for Coin Toss

$$\theta_{MAP} = \arg\max \sum_{i=1}^{N} \{\log p(y_i|\theta) + \log f_\theta(\theta)\}$$

$$= \arg\max \left[n^{(1)} \log \theta + n^{(0)} \log(1-\theta) + \log \frac{\theta^{\alpha-1}(1-\theta)^{\beta-1}}{B(\alpha,\beta)} \right]$$

Beta-distribution
$\alpha, \beta > 1$

$\theta = p(y_i = 1)$

$n^{(1)}$ = no. of 1s

$n^{(0)}$ = no. of 0s

$n^{(1)} + n^{(0)} = n$

$$= \arg\max \left[n^{(1)} \log \theta + n^{(0)} \log(1-\theta) + (\alpha-1)\log\theta + (\beta-1)\log(1-\theta) \right]$$

$$= \arg\max \left[(n^{(1)} + \alpha - 1)\log\theta + (n^{(0)} + \beta - 1)\log(1-\theta) \right]$$

$\alpha = 1, \beta = 1$

$\Rightarrow$

$\theta_{MAP} = \theta_{MLE}$

$\theta = \frac{n^{(1)}}{n}$

$$\frac{\partial L}{\partial \theta} = 0 \Rightarrow \frac{n^{(1)} + \alpha - 1}{\theta} - \frac{n^{(0)} + \beta - 1}{1-\theta} = 0$$

$$\Rightarrow \frac{1}{\theta} - 1 = \frac{n^{(0)} + \beta - 1}{n^{(1)} + \alpha - 1}$$

$$\Rightarrow \theta_{MAP} = \frac{n^{(1)} + \alpha - 1}{n + \alpha + \beta - 2}$$

Bayesian Estimation

$$p(\tilde{y}|y) = \int_0^1 p(\tilde{y}|\theta)\, p(\theta|y)\, d\theta$$

$$p(\theta|y) = \frac{p(y|\theta)\,p(\theta)}{p(y)} = \frac{p(y|\theta)\,p(\theta)}{\int_0^1 p(y|\theta)\,p(\theta)\,d\theta}$$

$$= \frac{\left[\prod_i p(y_i|\theta)\right] p(\theta)}{\int_0^1 \left[\prod_i p(y_i|\theta)\right] p(\theta)\, d\theta}$$

$$= \frac{\theta^{n^{(1)}} (1-\theta)^{n^{(0)}} \theta^{\alpha-1} (1-\theta)^{\beta-1}}{B(n^{(1)}+\alpha,\, n^{(0)}+\beta)}$$

↳ Assume Beta distribution

Supervised Learning: Math Details [Gradient Descent + Regularisation]

Data : (x_i, y_i)

$y_i = h(x_i, \theta) + \epsilon_i$ → Gaussian Noise

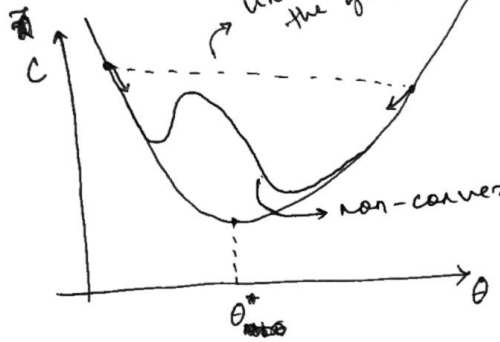
h: parameters; x_i: features

$\theta_{MLE} = \text{argmin} \sum_i [y_i - h(x_i, \theta)]^2$ → Least Squares Error fn.

C : cost function

$\frac{\partial C}{\partial \theta} = 0$ May not be analytically tractable.

$C(\theta)$ has to be convex.



Gradient Descent:

1. Pick a starting value of θ
2. Move on the graph in direction opposite to the gradient.

$\theta \rightarrow \theta - \eta \frac{\partial C}{\partial \theta}$

η : Learning rate

η should neither be too high → no convergence nor too low. → very slow convergence

why not $\theta \rightarrow \theta - \eta \, \text{sgn}\left\{\frac{\partial C}{\partial \theta}\right\}$?

As we approach θ^*, we want change in θ to become smaller & smaller.

If needed $\frac{\partial C}{\partial \theta}$ can be estimated using Finite Difference or Automatic Differentiation (i.e. backpropagation in ANN)

If y_i, x_i, θ are vectors.

$$C = \sum_{i=1}^{N} [y_i - h(x_i, \theta)]^2 + \frac{\lambda}{2} \|\theta\|^2 \; ; \quad \|\cdot\| : \text{Norm of vector}$$

Regularisation: prior information prevents parameters from becoming too large.

[Ridge Regression]

Lasso : L^1 norm

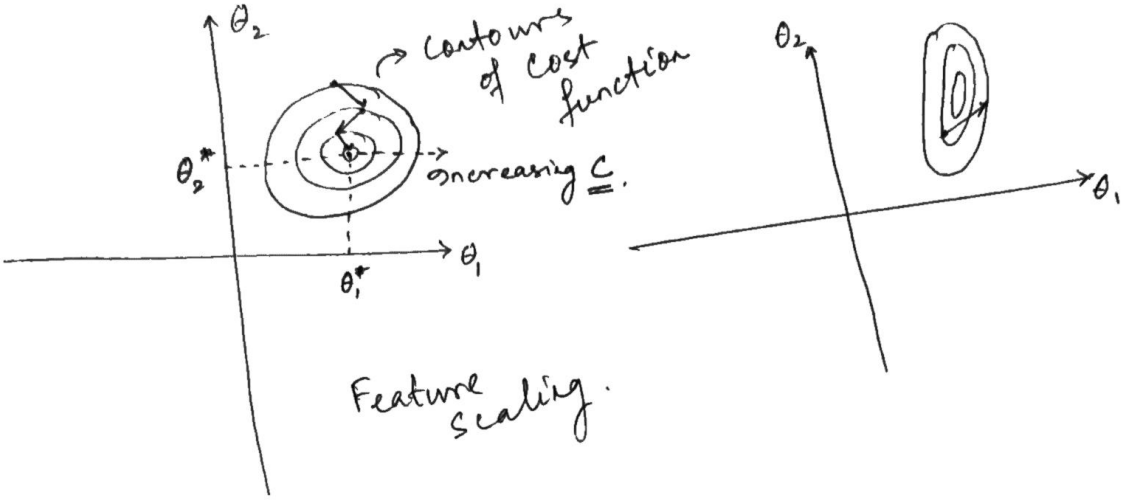
$\lambda = 0 \Rightarrow$ overfitting

$\lambda \approx \infty \Rightarrow$ underfitting

$\theta \rightarrow \theta - \eta \nabla_\theta C$

[Example in scikit-learn website]

(see next page)



Feature scaling.

Basic Steps of Supervised Learning:

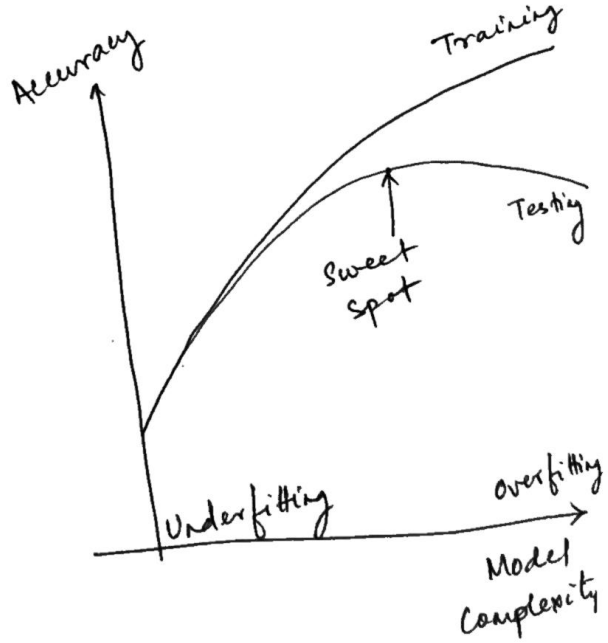
1. Choose appropriate model, i.e. $f(x_i, \theta)$ & $C(\theta)$
2. Divide data into training & testing set.
3. Train model parameters using gradient descent or some other optimisation method.
4. Stopping criteria for g.d.
5. Find training & testing accuracy.
6. If accuracy not adequate, go to step (1).

- Fix a desired precision

$$\| \theta^{k+1} - \theta^k \| < \epsilon$$

$$\| C^{k+1} - C^k \| < \epsilon$$

$$\| \nabla_\theta C \| < \epsilon$$



Linear Regression with Single ~~Variable~~ feature

$$y_i = mx_i + c + \epsilon_i$$

MLE.

$$S(m,c) = \sum_{i=1}^{N} \epsilon_i^2 \quad \text{LSE: Least Squares Error}$$

[Gives most optimal solution if ϵ_i is Gaussian. If there are outliers, non-Gaussian dist. may be required.]

$$= \sum_{i=1}^{N} (y_i - mx_i - c)^2$$

~~We~~ find m & c which minimise S.

Ex: House Price vs. Size

$$\frac{\partial S}{\partial m} = 0 \Rightarrow -2\sum_{i=1}^{N} x_i(y_i - mx_i - c) = 0$$

$$\Rightarrow m\sum_{i=1}^{N} x_i^2 + c\sum_{i=1}^{N} x_i = \sum_{i=1}^{N} x_i y_i$$

$$\frac{\partial S}{\partial c} = 0 \Rightarrow -2\sum_{i=1}^{N} (y_i - mx_i - c) = 0$$

$$\Rightarrow m\sum_{i=1}^{N} x_i + cN = \sum_{i=1}^{N} y_i$$

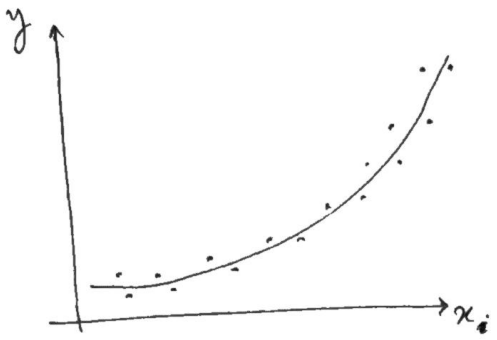
Solving the simultaneous equation, we get

$$m = \frac{N\sum_i x_i y_i - \sum_i x_i \sum_i y_i}{N\sum_i x_i^2 - \left(\sum_i x_i\right)^2}$$

$$c = \frac{\sum_i y_i - m\sum_i x_i}{N}$$

Linear Reg is one of the most popular & effective methods used in science & engineering.

"Work Horse"



$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2$$

Can still be solved using Linear Regression methods.

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2}$$

where $x_{i1} = x_i$

$x_{i2} = x_i^2$

In general,

$$f(x) = \beta_0 + \sum_{j=1}^{m} x_j \beta_j \; ; \quad X = [x_1, x_2, \cdots, x_m]$$

$$= \sum_{j=0}^{m} x_j \beta_j \quad \text{where } x_0 = 1.$$

Multiple Linear Regression

~~or Multivariate~~

Multivariate (General)

↳ multiple output.

$$y = \omega_0 + \omega_1 x_1 + \omega_2 x_2 + \epsilon$$

$$y_i = \omega_0 + \omega_1 x_{1i} + \omega_2 x_{2i} + \epsilon_i \; ; \; 1 \le i \le 5$$

Five data points :

$$X = \begin{bmatrix} 1 & x_{11} & x_{21} \\ 1 & x_{12} & x_{22} \\ 1 & x_{13} & x_{23} \\ 1 & x_{14} & x_{24} \\ 1 & x_{15} & x_{25} \end{bmatrix}$$

$$\omega = \begin{bmatrix} \omega_0 \\ \omega_1 \\ \omega_2 \end{bmatrix}$$

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_5 \end{bmatrix}$$

MULTIPLE ~~MULTIVARIATE~~ LINEAR REGRESSION

Linear Regression with single feature: MLE with Gaussian error.

$$y_i = mx_i + c + \epsilon_i$$

m, c obtained by minimizing $S = \sum_i (y_i - mx_i - c)^2$ — LSE.

Multiple LinReg:

$$y_i = w_0 + w_1 x_{1i} + x_{2i} + x_{3i} + \cdots + w_m x_{mi} + \epsilon_i$$

$$= w^T x_i + \epsilon_i$$

↳ w & x_i are vectors instead of scalars. but y is scalar.

~~y can also be a vector. (in which case y_i is a tensor)~~

$$S = \sum_i (y_i - w^T x_i)^2$$

~~vector vector matrix~~

$$= (y - Xw)^T (y - Xw)$$

(y: vector of all data; Xw: vector; X: Matrix)

$$= (y^T - w^T X^T)(y - Xw)$$

$$= y^T y - 2y^T Xw + w^T X^T X w$$

Each x_i can be a non-linear function of other variables. So, this linear reg. can actually model polynomial & other functions as well.

$$\nabla_w S = 2X^T X w - 2X^T y = 0$$

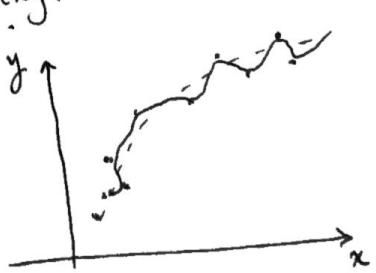
$$\Rightarrow \boxed{X^T X w = X^T y} \equiv \text{NORMAL EQUATION}$$

$$\Rightarrow w_{OLS} = (X^T X)^{-1} X^T y$$ — ORDINARY LEAST SQUARES

↳ very hard to compute

If model complexity is large, parameter values can be large leading to unstable solution, i.e. w_{OLS} can change a lot for small changes in data points. Also leads to over-fitting.

USE REGULARISATION.



RIDGE REGRESSION

Encourage parameter values to be small [MAP]

use zero-mean Gaussian PRIOR

$$S = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - w^T x_i\right)^2 + \lambda \underbrace{\|w\|_2^2}_{\sim} \rightarrow w^T w$$

L_2 Regularisation term or weight decay.

$$w_{RIDGE} = \left(\lambda I_D + X^T X\right)^{-1} X^T y$$

$\lambda = 0 \Rightarrow$ over-fitting

$\lambda \approx 1 \Rightarrow$ under-fitting.

$\|w\|_2 = \sqrt{w_0^2 + w_1^2 + w_2^2 + \dots}$

↳ usually omitted from regularisation

LASSO

Use L_1 regularisation

$$S = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - w^T x_i\right)^2 + \lambda \underbrace{\|w\|_1}_{\sim} \rightarrow L_1 \text{ norm}$$

useful when we have outliers (non-Gaussian errors)

$\|w\|_1 = |w_0| + |w_1| + |w_2| + \dots$

SUBSET SELECTION

Pick K best features using some thresholds.

BAYESIAN Linear Regression

Estimate errors in estimation of weights.

MULTIVARIATE (GENERAL) LINEAR REGRESSION

~~Multiple~~ Vector output

Same expression for Normal Eqn. works

but y is now a matrix instead of a vector.

LOGISTIC REGRESSION - I

Classification : Spam vs. Normal Email
Malignant vs. Healthy Tissue
Fake vs. Real News.
Human vs. Animal image.

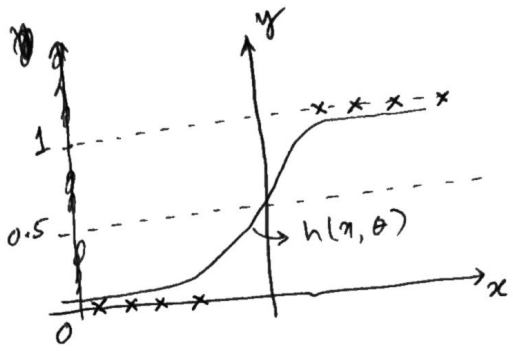
One of the most popular ML algos.

$y_i \in \{0, 1\}$ → True/Yes, False/No

No classifier is perfect.
So, we are looking for a probability estimate for

$p(y_i | x_i, \theta)$

$\hat{y}_i = h(x_i, \theta)$ → hypothesis
& $0 \le h \le 1$ $\forall$ x_i, θ



Clearly y vs. x is NOT a straight line.
So, makes no sense to ~~fit a st~~ use linear regression

Decision boundary: (Linear) or HYPERPLANE

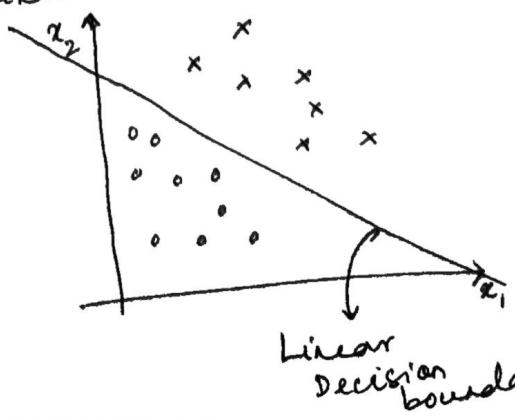
$\theta_0 + \theta^T x_i > 0 \Rightarrow h(x_i, \theta) > 0.5$
$\theta_0 + \theta^T x_i \le 0 \Rightarrow h(x_i, \theta) \le 0.5$
$\Rightarrow h(x_i, \theta) = h(\theta^T x_i + \theta_0)$

Sigmoid / Logistic function : or Logit

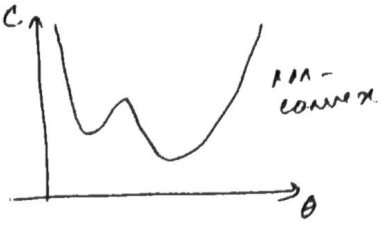
$$h(x_i, \theta) = \frac{1}{1 + e^{-(\theta^T x_i + \theta_0)}}$$

Many other options available but this one is easier to handle & interpret.

$$\log \underbrace{\frac{h(x_i, \theta)}{1 - h(x_i, \theta)}}_{\text{odds}} = \theta_0 + \theta^T x_i$$: Linear Model.



LSE : $C_{LSE}(\theta) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$



But this not convex.

▼

for a function, $f(\theta)$, to be convex.

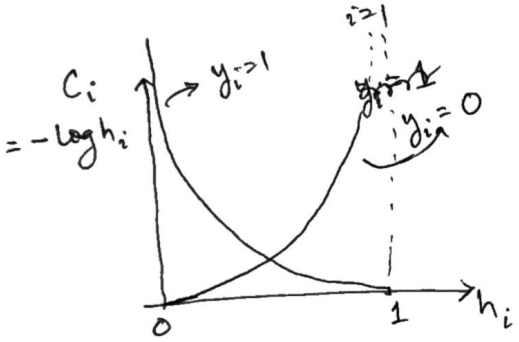
we must have $\frac{\partial^2 f}{\partial \theta^2} > 0 \quad \forall \theta$

for binary classification,

$\theta_{MLE} = \text{argmin} \frac{1}{N}\sum_{i=1}^{N}[-y_i \log h_i(\theta) - (1-y_i)\log(1-h_i)]$

$\therefore C(\theta) = \frac{1}{N}\sum_{i=1}^{N}[-y_i \log h_i - (1-y_i)\log(1-h_i)] \quad ; y_i \in \{0,1\}$

↳ Binary Cross Entropy.



If $y_i = 1$, $C_i(\theta) = -\log h_i$

If $y_i = 0$, $C_i(\theta) = -\log(1-h_i)$

Gradient Descent:

$\theta_j \longrightarrow \theta_j - \eta \frac{\partial C}{\partial \theta_j}$

↳ Learning Rate

LOGISTIC REGRESSION - II

MULTI-CLASS CLASSIFICATION

$S = \{\text{Spam}, \text{Important}, \text{Not Important}\}$

$$y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ y^{(3)} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \quad \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

ONE-HOT ENCODING

$$\hat{y} = \begin{bmatrix} \hat{y}^{(1)} \\ \hat{y}^{(2)} \\ \hat{y}^{(3)} \end{bmatrix}$$

$$\hat{y}^{(1)} = p[y^{(1)} = 1]$$
$$\hat{y}^{(2)} = p[y^{(2)} = 1]$$
$$\hat{y}^{(3)} = p[y^{(3)} = 1]$$

$$0 \le \hat{y}^{(k)} \le 1 \qquad \sum_k \hat{y}^{(k)} = 1$$

$$z^{(k)} = e^{+x^T \theta^{(k)} + \theta_0^{(k)}}$$

SOFTMAX function:

$$\hat{y}^{(1)} = \frac{z^{(1)}}{z^{(1)} + z^{(2)} + z^{(3)}}$$

$$\hat{y}^{(2)} = \frac{z^{(2)}}{z^{(1)} + z^{(2)} + z^{(3)}}$$

$$\hat{y}^{(3)} = \frac{z^{(3)}}{z^{(1)} + z^{(2)} + z^{(3)}}$$

COST / LOSS FUNCTION

for 2-classes:

$$C = \frac{1}{N} \sum_{i=1}^{N} \left[-y_i \log \hat{y}_i - (1 - y_i) \log (1 - \hat{y}_i) \right]$$

$$= \frac{1}{N} \sum_{i=1}^{N} \left[-y_i^{(1)} \log \hat{y}_i^{(1)} - y_i^{(2)} \log \hat{y}_i^{(2)} \right]$$

for K-classes:

$$C = \frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} - y_i^{(k)} \log \hat{y}_i^{(k)}$$

CATEGORICAL CROSS ENTROPY

LOGISTIC REGRESSION - I (contd.)

Classification : $y_i \in \{0, 1\}$

$$\hat{y}_i = h(x_i, \theta) = \frac{1}{1 + e^{-(\theta^T x_i + \theta_0)}} \rightarrow \text{SIGMOID, LOGISTIC, LOGIT fn.}$$

$$C(\theta) = \frac{1}{N} \sum_{i=1}^{N} \left[-y_i \log h_i - (1 - y_i) \log (1 - h_i) \right] \rightarrow \text{BINARY CROSS ENTROPY.}$$

Gradient Descent : $\theta_j \rightarrow \theta_j - \eta \frac{\partial C}{\partial \theta_j}$ (η : Learning Rate)

$$\frac{\partial C}{\partial \theta_j} = \frac{1}{N} \sum_{i=1}^{N} \left[-\frac{y_i}{h_i} \frac{\partial h_i}{\partial \theta_j} + \frac{1 - y_i}{1 - h_i} \frac{\partial h_i}{\partial \theta_j} \right]$$

$$= \frac{1}{N} \sum_{i=1}^{N} \frac{(h_i - y_i)}{h_i (1 - h_i)} \frac{\partial h_i}{\partial \theta_j}$$

$$= \frac{1}{N} \sum_{i=1}^{N} (h_i - y_i) x_{ij}$$

Form Exactly same as Linear Regression just with a different expression for h_i

$$\frac{\partial h_i}{\partial \theta_j} = \frac{-1}{(1 + e^{-\theta^T x_i - \theta_0})^2} \cdot \frac{\partial}{\partial \theta_j} e^{-\theta_T x_i}$$

$$= \frac{e^{-\theta_T x_i - \theta_0}}{(1 + e^{-\theta^T x_i - \theta_0})^2} \cdot \frac{\partial}{\partial \theta_j} \theta^T x_i$$

$$= \frac{1}{1 + e^{-\theta^T x_i - \theta_0}} \frac{e^{-\theta^T x_i - \theta_0}}{1 + e^{-\theta^T x_i - \theta_0}} \cdot x_{ij}$$

$$= h_i (1 - h_i) x_{ij}$$

The features x_{ij} used in Logit can actually be nonlinear fn. (eg. polynomial) of other features thereby allowing the decision boundary to be effectively nonlinear.

REGULARISATION

$$C(\theta) = \frac{1}{N} \sum_{i=1}^{N} \left[-y_i \log h_i - (1 - y_i) \log (1 - h_i) \right] + \frac{\lambda}{N} \|\theta\|_2^2$$

as $N \rightarrow \infty$, this term drops out.

$$\therefore \frac{\partial C}{\partial \theta_j} = \frac{1}{N} \sum_{i=1}^{N} (h_i - y_i) x_{ij} + \frac{2\lambda}{N} \theta_j$$

MULTI-CLASS LOGISTIC REGRESSION

Email: Spam, Important, Not-Important

Face identification

ONE vs. ALL [or One vs. Rest]

Let $y_i \in \{S, I, NI\} \equiv K$ - categories

Divide into three problems

$h^{(1)}(x_i, \theta) =$ 1 (S) vs. 0 ($\{I, NI\}$)

$h^{(2)}(x_i, \theta) =$ I vs. $\{S, NI\}$

$h^{(3)}(x_i, \theta) =$ NI vs. $\{S, I\}$.

K - binary classifications

$h(x_i, \theta) = \max \{h^{(1)}, h^{(2)}, h^{(3)}\}$

ONE vs. ONE

S vs. I

I vs. NI

S vs. NI

${}^{K}C_2$ binary classifications.

Pick category which has most classifications.

These methods do not give probability estimates

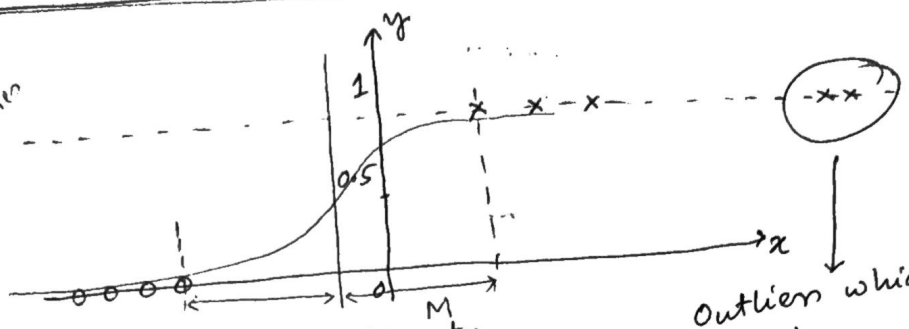
since $h^{(1)} + h^{(2)} + h^{(3)} \neq 1$

One vs. One is also computationally expensive.

Better to have a method which can learn multi-class parameter at one go.

SUPPORT VECTOR MACHINES - I

faster than Logit
extendible to nonlinear boundaries



SVM: Give up probabilistic estimates for a better decision boundary

Only worry about points closest to decision boundary.

Classification Rule : $G(x_i) = \text{sgn}[\theta^T x_i]$

$h(x_i, \theta) = \theta_0 + \theta^T x_i > 0 \Rightarrow \hat{y}_i = +1$

$h(x_i, \theta) = \theta_0 + \theta^T x_i < 0 \Rightarrow \hat{y}_i = -1$

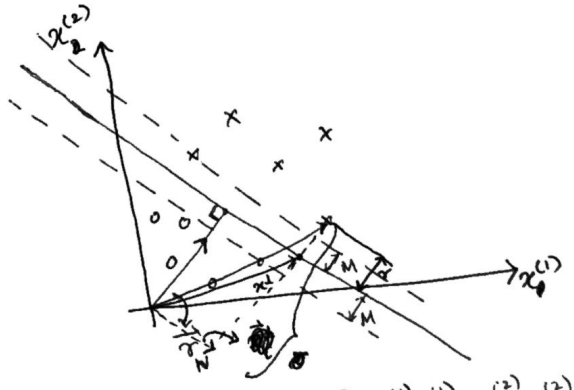
M : Margin

$y \in \{-1, +1\}$

find θ such that M is maximised for the ~~closest~~ points closest to Decision Boundary hyperplane.

$\vec{r}_N = (\theta^{(1)}, \theta^{(2)})$

(Normal to the decision boundary)



$\vec{x}_i = \vec{x}_i^{\perp} + y_i r_i \frac{\vec{\theta}}{\|\vec{\theta}\|}$

$h(x_i, \theta) = \theta_0 + \theta^T x_i = \theta_0 + \vec{\theta} \cdot \vec{x}_i$

$= \theta_0 + \vec{\theta} \cdot \left[\vec{x}_i^{\perp} + y_i r_i \frac{\vec{\theta}}{\|\theta\|} \right]$

$= \left(\theta_0 + \vec{\theta} \cdot \vec{x}_i^{\perp} \right) + y_i r_i \frac{\|\theta\|^2}{\|\theta\|} = y_i r_i \|\theta\|$

$\Rightarrow r_i = \frac{h(x_i, \theta)}{y_i \|\theta\|}$

$= \frac{\theta_0 + \theta^T x_i}{y_i \|\theta\|}$

Objective : Maximise r_i for the closest point [Support Vector]

while ensuring $y_i h(x_i, \theta) > 0$

$\theta^{(0)} + \theta^{(1)} x^{(1)} + \theta^{(2)} x^{(2)} = 0$

$\Rightarrow x^{(2)} = -\frac{\theta^{(1)}}{\theta^{(2)}} x^{(1)} - \frac{\theta^{(0)}}{\theta^{(2)}}$

OBJECTIVE

$$\max_{\theta, \theta_0} \min_{x_i} \frac{y_i(\theta_0 + \theta^T x_i)}{\|\theta\|}$$

Note: Scaling θ, θ_0 does not change this function.

So, scale them so that $y_i h_i = 1$ for point closest to decision boundary.

$$\max_{\theta, \theta_0} \frac{1}{\|\theta\|} \quad \text{s.t.} \quad y_i(\theta_0 + \theta^T x_i) \geq 1 \quad \forall i$$

$$= \min_{\theta, \theta_0} \frac{1}{2}\|\theta\|^2 \quad \text{s.t.} \quad "$$

↳ small $\|\theta\|$ is regularisation. ⇒ good generalisation

QUADRATIC program with N linear constraints [convex]

Lagrange function

Convert ↺ $L_D = \frac{1}{2}\|\theta\|^2 - \sum_{i=1}^{N} \alpha_i \left[y_i(\theta_0 + \theta^T x_i) - 1 \right]$

↳ Lagrange multipliers

$\alpha_i \geq 0 \ \forall i$

$\alpha_i \neq 0$ only for support vectors.

$$\max_{\alpha_i} \min_{\theta, \theta_0} L_D$$

$$\frac{\partial L_D}{\partial \theta_0} = -\sum_{i=1}^{N} \alpha_i y_i = 0$$

$$\nabla_\theta = \theta_j - \sum_{i=1}^{N} \alpha_i y_i x_{ij} = 0 \Rightarrow \boxed{\theta_j = \sum_{i=1}^{N} \alpha_i y_i x_{ij}}$$

Support vectors.

Substituting this in L_D, we get.

$$L_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{i'=1}^{N} \alpha_i \alpha_{i'} y_i y_{i'} x_i^T x_{i'}$$

concave.

↳ lower bound on objective function.

Maximising this is a simpler convex quadratic programming problem.

LAGRANGE DUAL OBJECTIVE FUNCTION (WOLFE)

min. of original L_D is $-\infty$!!

SUPPORT VECTOR MACHINES – II

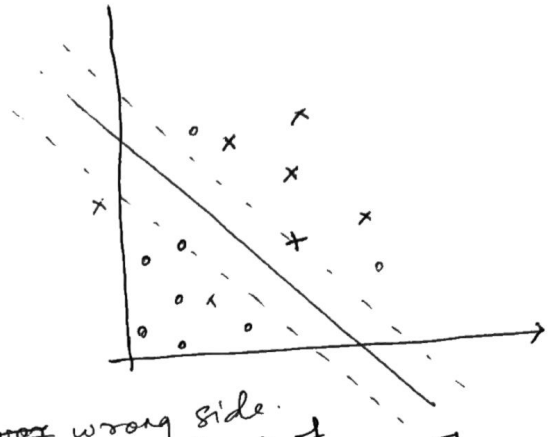
What if the classes are not linearly separable?

If overlap is small, use SLACK VARIABLES

$\xi_i \geq 0$, $i = 1, 2, \cdots, N$

↓

Margin by which points are allowed to be on wrong side.



$\xi_i \geq 0$, $\sum \xi_i \leq$ constant [sort of regularization]

$$\min_{\theta, \theta_0} \frac{1}{2}\|\theta\|^2 \quad \text{s.t.} \quad y_i(\theta_0 + \theta^T x_i) \geq 1 - \xi_i$$

$$L_D = \frac{1}{2}\|\theta\|^2 + C\sum_i \xi_i - \sum_i \alpha_i \left[y_i(\theta_0 + \theta^T x_i) - (1 - \xi_i) \right] - \sum_i \mu_i \xi_i$$

(C: trade-off between large margin & small training error)

Lagrange (primal) function [minimise wrt. θ_0, θ, ξ_i]

$$L_D = \sum_i \alpha_i - \frac{1}{2}\sum_i \sum_{i'} \alpha_i \alpha_{i'} y_i y_{i'} \boxed{x_i^T x_{i'}}$$

Lagrange (Wolfe) Dual objective fn.

Optimisation depends only on dot product of input features.

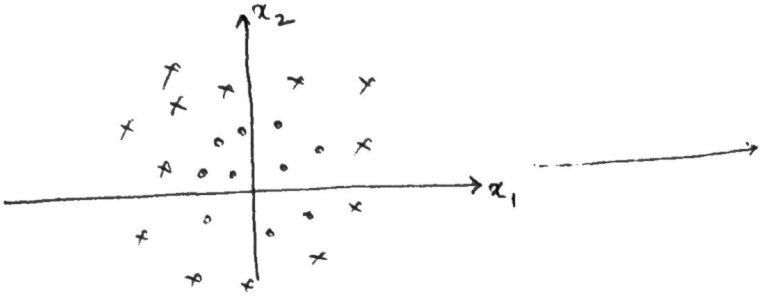
* What if overlap is large?

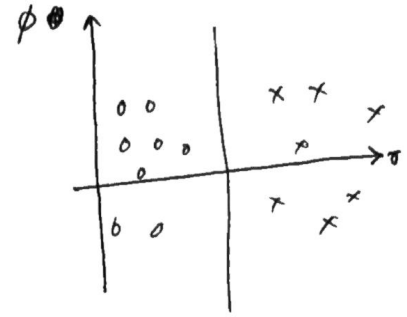
Need to transform variables, so that in the new space, the points are linearly separable.

Since dual L_D only depends on dot products, we don't need the actual transformation expression but only the new dot-product.

⇒ Kernel trick.

$$L_D = \sum_i \alpha_i - \sum_i \sum_{i'} \alpha_i \alpha_{i'} y_i y_{i'} \underbrace{K(x_i, x_{i'})}_{\text{positive semi-definite \& symmetric.}}$$





$r = \sqrt{x_1^2 + x_2^2}$

$\phi = \tan^{-1} x_2/x_1$

Popular choices for Kernel.

d-th degree polynomial:

$$K(x_i, x_{i'}) = (1 + \langle x_i, x_{i'} \rangle)^d$$

Radial Basis (RBF) or Gaussian (usually best choice):

$$K(x_i, x_{i'}) = \exp\left(-\gamma \|x_i - x_{i'}\|^2\right)$$

(γ: kernel bandwidth) — Most widely used based on Euclidean distance.

Neural Network or Sigmoidal:

$$K(x_i, x_{i'}) = \tanh(\gamma_1 \langle x_i, x_{i'} \rangle + \gamma_2)$$

In original LD, if C is large, it penalises any $\xi_i \neq 0$ leads to a very wiggly boundary (overfit).

$$h_{i'} = \theta_0 + \theta^T x_{i'}$$

$$\nabla_\theta L_D = \theta_j - \sum_i \alpha_i y_i x_{ij} = 0 \Rightarrow \theta_j = \sum_i \alpha_i y_i x_{ij}$$

$$\Rightarrow h_{i'} = \theta_0 + \sum_i \alpha_i y_i x_i^T x_{i'}$$

$$= \theta_0 + \sum_i \alpha_i y_i K(x_i, x_{i'})$$

RBF: Large λ leads to overfitting since even small distances are penalised.

Curse of Dimensionality

SVM-II (contd.)

$$L_D = \frac{1}{2}\|\theta\|^2 + C\sum_i \xi_i - \sum_i \alpha_i\left[y_i(\theta_0 + \theta^T x_i) - (1-\xi_i)\right] - \sum \mu_i \xi_i$$

minimize w.r.t. θ, θ_0, ξ_i → vector

$$\frac{\partial L_D}{\partial \theta_0} = 0 \Rightarrow \sum_i \alpha_i y_i = 0$$

$$\nabla_\theta L_D = 0 \Rightarrow \theta = \sum_i \alpha_i x_i y_i$$

$$\frac{\partial L_D}{\partial \xi_i} = 0 \Rightarrow \alpha_i = C - \mu_i$$

Substituting all these in L_D, we obtain,

$$L_D = \sum_i \alpha_i - \frac{1}{2}\sum_i \sum_{i'} \alpha_i \alpha_{i'} y_i y_{i'} x_i^T x_{i'}$$

↳ Lagrange Dual objective function

Maximize using Quadratic Programming subject to $0 \le \alpha_i \le C$ & $\Sigma \alpha_i y_i = 0$

To get optimum solution, use Karush-Kuhn-Tucker conditions,

$$\alpha_i\left[y_i(\theta_0 + \theta^T x_i) - (1-\xi_i)\right] = 0 \rightarrow \alpha_i \neq 0 \text{ only for support vectors}$$

$$\mu_i \xi_i = 0 \rightarrow \mu_i \neq 0 \text{ only when } \xi_i \neq 0$$

$$y_i(\theta_0 + \theta^T x_i) - (1-\xi_i) \geq 0$$

SVM - Regularisation

$\min \frac{1}{2}\|\theta\|^2 + C\sum \xi_i$

γ ↑ Kernel

C ↑

SVM - Polynomial & Sigmoid Kernel application. [when does RBF not work]

RBF is slower than polynomial kernel; especially for large data or high dimension

Polynomial kernel is popular in NLP with $d=2$ [$d\uparrow \Rightarrow$ overfitting & numerical instability]

~~Ro~~ No free Lunch Theorem

SVM Multi-class

One vs. One or One vs. Rest.

Lagrange Multiplier [could be maxima/minima]

Support vectors are not necessarily the points closest to decision boundary ~~for~~ when slack variables used.

High γ RBF overfitting

LogReg → find prob & change threshold.

+ve θ_j ⇒ feature favours $\hat{y}=1$

θ_0 : intercept

θ : co-efficients

Missing values → take mean/mode or remove from data

~~Kernel~~ - ~~DOES NOT scale well to large training sets or large feature vectors~~

~~RBF is slower~~

Hyper plane - Decision Boundary

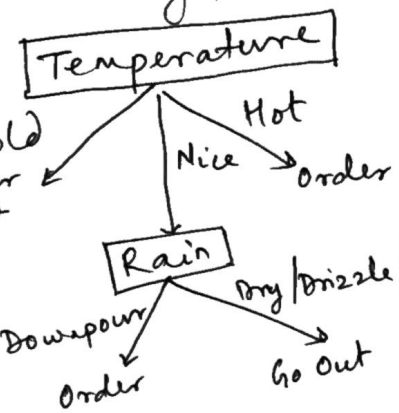
Hyper parameters (user tuned) vs. Parameter (internal model)

Generative vs. Discriminative

DECISION TREES

Represents hierarchical decision making process.

Dinner: Go Out or Order?



Each node represents an attribute or feature & each branch represents possible value for that attribute (can be non-numerical)

How to decide hierarchy?

Nodes with lowest entropy (best classification) is at the top/higher level.

A: Attribute

S: Total collection of examples

SA_v: Subset of S for which Attribute A has value 'v'.

H_S: Entropy of S

H_{SA_v}: Entropy of SA_v

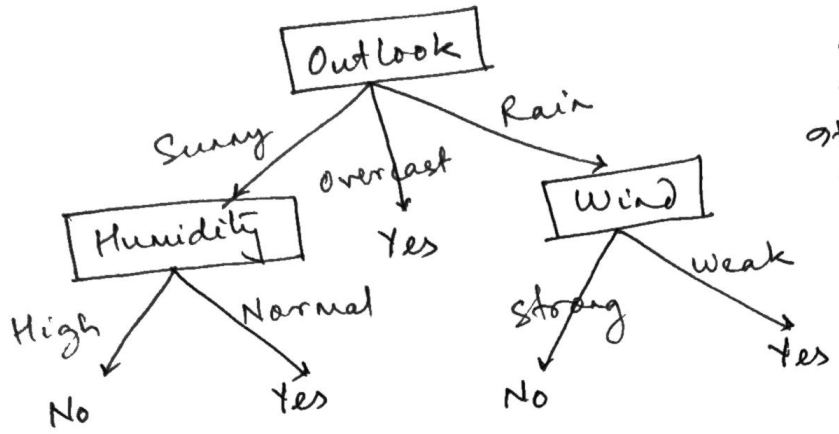
H_A: Entropy of A.

$$H_{SA_v} = \sum - p_i \log_2 p_i$$

$$H_A = \sum_{v \in A} \frac{|SA_v|}{|S|} H_{SA_v}$$

Information Gain: $G(S,A) = H_S - H_A$

ML by Tom Mitchell. [Play Tennis]



ID3:
Iterative
Dichotomiser 3

Gini index/impurity:

$$G_{AS_v} = 1 - \sum p_i^2$$

$$G_A = \sum \frac{|SA_v|}{|S|} G_{AS_v}$$

Works almost similar to entropy measure. but faster to calculate.

Overfitting

- Pre-pruning: stop split beyond a significance threshold
- Post-pruning: based on validation set
  [better than pre-prune since hard to decide threshold]

Missing Features

Assign most common value or using some probability measure.

ADVANTAGE OF TREES

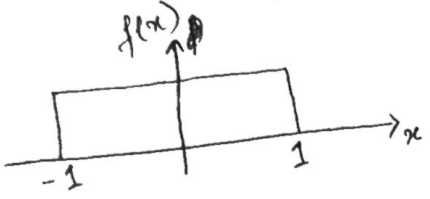
- Easy to interpret
- Easily handle mix data [Symbols | Numbers]
- Robust to outliers
- Scales well to large datasets
- ...

DISADV.

- Low bias & high variance
- Unstable due to hierarchical nature
  [small change in data can lead to large change in tree]

Addressed by Random Forest.

RANDOM FOREST



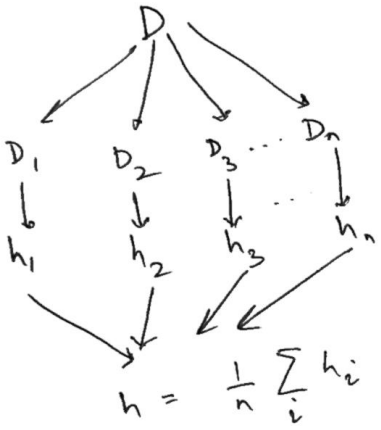
$-1 \le x \le 1 \qquad \mathbb{E}[X] = 0$

$X_1, X_2, X_3, \cdots, X_n$ iid rv.

$Y = \dfrac{X_1 + X_2 + \cdots + X_n}{n} \longrightarrow \mathbb{E}[X]$ as $n \to \infty$

Law of Large Numbers

standard method of reducing variance in prediction.



BAGGING (↗ Bootstrap Aggregating) : very effective way to reduce variance.

Each D_i has same size as D
[drawn randomly, can have same data point multiple times]

for each split, use a subset of features.

RANDOM FOREST : Bagging + Decision Trees.

Very resilient to "curse of dimensionality"
gets slower but still works very well.

only two hyper-parameters
n & $\sqrt{D}$

But looses interpretability.

~~For each D_i, select~~
~~choose features randomly.~~

Each decision tree is a weak learner
Bagging makes it a strong learner.

ENSEMBLE LEARNING

for ML methods with ~~poor~~ high bias, use BOOSTING.

STACKING

Bagging runs in parallel
Boosting is serial or sequential
[use training accuracy to assign higher weight to misclassified data & re-train. Do in loop for few times]

Logistic Regression vs. SVM vs. Random Forest.

LR	SVM	RF.
① Works best with linearly separable data	Can handle nonlinearity by using kernels.	Inherently nonlinear
② Easier to interpret results. Also gives probability estimates	Hard to interpret results no prob.	~~Same as SVM~~ hard to interpret but gives prob.
③ Usually fast	faster than LR	Usually very slow
④ Works better with balanced data		Can easily handle unbalanced data
⑤ Has problems with high dimensional data (lots of features)	same as LR	Easily handles curse of dimensionality (feature subset used)
⑥ Can't handle outliers	Handles outliers better	Can handle noisy data
⑦ better for 2-class	better for 2-class	Better for multi-class classification

BEST PRACTICE — Try all 3, tune hyper-parameters & choose the one that works best.

DECISION TREES

Data fragmentation: Continuous partitioning leaves less data for lower-level nodes leading to weak statistical support.

→ Can lead to multiple mis-classifications.

→ General problem in rule based learning.

NP-Complete:

A problem G_P in NP is NP-complete if every other problem in NP can be transformed (or reduced) to P_G in polynomial time.

obviously, G_P is also NP-hard.

but NP-hard does not imply NP-complete (eg. halting problem)

→ **Decision Trees are NP-complete.**

If you can solve Decision Tree problem in polynomial time, it would prove P=NP.

→ ID3 is a greedy, heuristic algorithm does best fit search for locally (no looking back) optimal entropy values. approximates the globally optimal solution.

→ Decision Tree: No real learning, purely rule based } Should it be called ML/AI?

→ In RF, choose subset of features, in each tree to avoid correlations between trees.

→ ~~ID~~ ID3 was originally for categorical values.
C4.5 allowed numerical values.
CART has binary ~~classification~~ tree & allows regression.

Gini represents the expected error resulting from labelling instances in the leaf randomly.

$p(1-p)$ ↳ coin toss variance.

Thresholding & Discretization.

Unsupervised thresholding : mean/median

Supervised " : lower convex hull

need to know number of 'bins' apriori

Requires careful data analysis

(mean/median may not be good classifiers)

ANN

- More general than

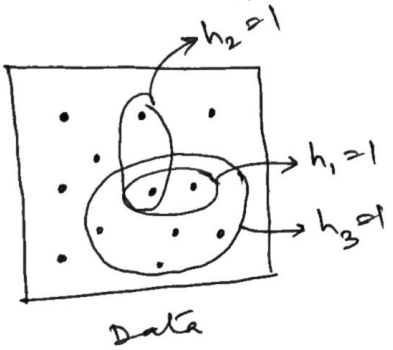
Let $h_j(x)$ & $h_k(x)$ be boolean-valued functions over X.

Then h_j is more-general-than-or-equal-to h_k ($h_j \geq_g h_k$)

iff $(\forall x \in X)\left[h_k(x) = 1 \Rightarrow h_j(x) = 1\right]$

if h_j is more-general-than h_k

then h_k is more-specific-than h_j

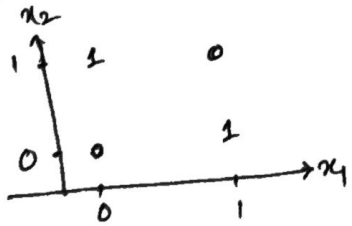


So $h_3 \geq_g h_1$

Perceptron = 1 neuron (node of hidden layer) in ANN. or output

Regularisation for ANN — L1, L2, Dropout.

XOR



SGD is not guaranteed to converge.

can use adaptive learning rate [not necessarily better]

Learning XOR

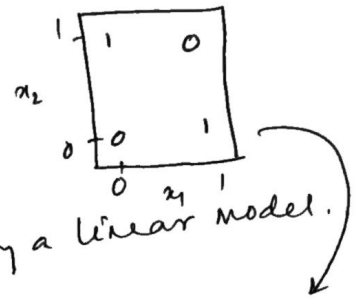
$X = \{[0,0], [0,1], [1,0], [1,1]\}$

MSE loss function.

$$J(\theta) = \frac{1}{4} \sum_{x \in X} \left(f^*(x) - f(x;\theta) \right)^2$$

$f(x; w, b) = x^T w + b$

Solving gives $w = 0$ & $b = 1/2$



=> XOR can't be implemented by a linear model.

ML non-linear function

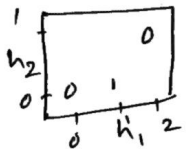
≡ Affine Transformation

+ Activation function

(Rectified Linear Unit, ReLU)

→ $g(z) = \max\{0, z\}$

$h_i = g(x^T W_{:,i} + c_i)$



$f(x; W, c, w, b)$

$= w^T \max\{0, W^T x + c\} + b$

$W = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$ $c = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$ $w = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$ $b = 0$.

Universal Approximation Theorem

- May not be able to find parameter values corresponding to the desired function
- Overfitting: May converge to the wrong function

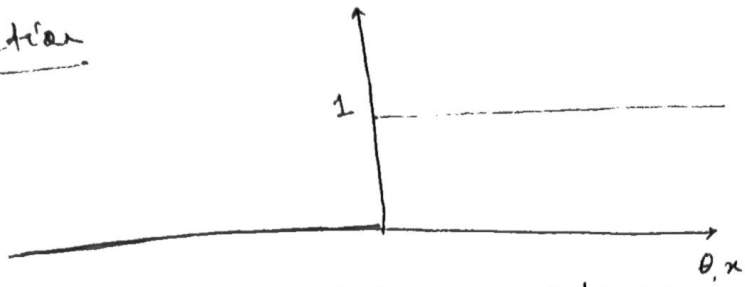
No Free Lunch Theorem

- There is no universally superior ML algo.

A feedforward network with single hidden layer is sufficient to represent any arbitrary function.

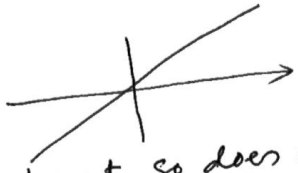
ANN ACTIVATION FUNCTIONS

Step function



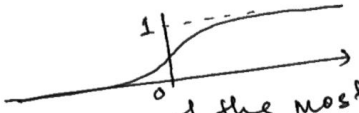
- Can be used in output layer
- derivative is always zero, so cant help in gradient descent & so not for hidden layers.

Linear function



- Gradient is a constant, so does not help in gradient descent
- Leads to overall linear classifier.

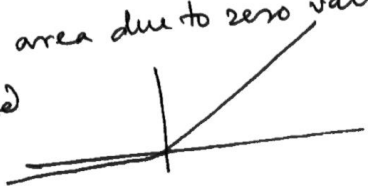
Sigmoid function



- good classifier, one of the most widely used.
- gives probability estimates
- has changing gradient
- but gradient saturates quickly leading to slow learning & can get stuck in local minima Vanishing gradient problem
- can use hyperbolic tangent fn. instead.

ReLU [Rectified Linear Unit] function

- Faster than sigmoid/tanh due to linearity & zero nature.
- But no learning in neg. area due to zero value.
- used Leaky ReLU instead



Softmax function

- Similar to sigmoid but helps in multiclass classification.
- usually used in output layer.

Usually, use ReLU in hidden layer & sigmoid/softmax in output layers.

~~K - nearest neighbour~~

ANN Optimization Methods

BATCH GRADIENT DESCENT

- usual gradient descent
- uses whole dataset for each update
- very slow & resource consuming
- can't use new data on the fly (no online update)
- usually converges to local minima for non-convex loss functions
- done over many epochs

STOCHASTIC GRADIENT DESCENT

- uses one data point at a time
- frequent updates with high variance
- random jumps allow convergence to ~~global minima~~ or potentially better minima.
  may (not ~~always~~ be) global since SGD can overshoot)
- faster convergence
- shuffle data for each epoch

MINI-BATCH GRADIENT

- SGD using small batch of dataset
- faster convergence than vanilla & less fluctuations than SGD with one data point
- usually called SGD.
- usually gets stuck at saddle points.

SGD + Momentum

- Add update vector of past time step
- helps avoiding being caught in ravines.
- like pushing a ball downhill.
- gradient terms add up if they point in same direction

NESTEROV SGD [NAG - Nesterov Accelerated Gradient]

- on SGD + momentum, need to slow down before gradient changes sign again.
- So estimate future parameter value using momentum term & then use gradient at that parameter value

$$v_t = \underbrace{\gamma v_{t-1}}_{\text{usually } 0.9} + \eta \nabla_\theta C(\theta - \gamma v_{t-1})$$

$$\theta = \theta - v_t$$

ADAGRAD

ADAM - faster
SGD - better generalised

- Perform larger updates for frequent parameters
- Helps in dealing with sparsity.
- Basically use different learning rate (η) for each parameter.
- η is modified for each parameter based on past gradients.
- But η can keep decreasing to zero in this method. Resolved by AdaDelta.

ADAM [Adaptive Moment Estimation]

Another method to compute adaptive learning rate.

Solving ODEs & PDEs using ANN

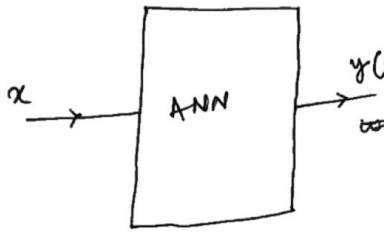
$\frac{dy}{dx} = -2xy$

$y(0) = 1$

$\Rightarrow y(x) = e^{-x^2}$

Analytical solution
Not always possible
Numerical Simulation
- Cannot give output beyond the range on which simulation performed
- Requires human analysis of results.

ANN
- Possibility of learning the actual function [Universal Approx. Thm.]



Train the ANN without having any knowledge of the analytic or numerical solution.

Need a loss (cost) function to be minimised

$$C = \left[\frac{dy}{dx} + 2xy\right]^2 + [y(0) - 1]^2$$

↳ Automatic Differentiation [AutoDiff] in Tensorflow

Different from both symbolic diff & numerical differentiation.

Symbolic Diff (eg. Mathematica) - Inefficient code
Numerical Diff - round-off errors
] unsuitable for higher & partial derivatives with many inputs needed in gradient descent.

AutoDiff is NOT numerical differences
It is both efficient & numerically stable.
(partly symbolic & partly numerical)

DEEP LEARNING

DL = ANN + Representation Learning

✓ Automatically discover the representations (abstract) needed for classification

✓ NO explicit feature selection

Mainly inspired by NLP, speech processing & Computer Vision.

Deep: Generally representation learning requires many layers of ANN.

CAP [Credit Assignment Path] : describes potentially causal connection between input & output.

DL usually has high CAP. [no specific threshold]

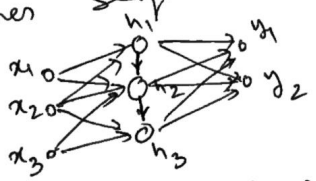
DL is computationally intensive & became popular largely due to GPUs.

NLP
- Symbolic [1950s - 90s]
- statistical [1990s -]
- Neural [2010s -]

DL Algorithms :

CNN - captures spatial information of images.

RNN - captures sequence information in NLP



vanishing & exploding gradients

LSTM - Additional structure in RNNs to decide which info to keep & which to discard.

Transformers - LSTMs are hard to train since it is sequential & can't be fully parallelised

Uses the "Attention Mechanism" only

Significant improvement in language translation accuracy.

Limitations or Criticisms of DL

- Lack of Theory & Explainability
  DL is essentially a blackbox
  We neither know the function learnt nor the features used for classification or other prediction.
  [especially an issue in medical diagnosis]
- Biases in prediction
  Algo can learn to classify based on "problematic" features
  Bias towards women & ~~other~~ minority groups (eg. automatic CV suggestions). for jobs & admission.
- Cyber Threat
  Easy to trick DL based surveillance systems
- Reliance on human microwork
  Labelling done by humans
  Continued demand for labelled data for calibration & updation of ANN.

Deep Learning is still Weak AI.

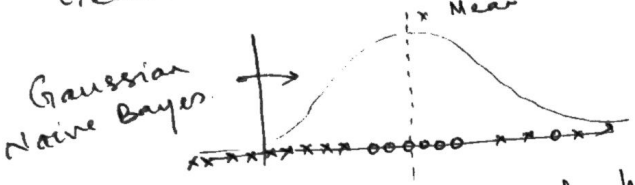
Awareness does NOT arise out of ignorance.

GENERATIVE vs. DISCRIMINATIVE MODELS

Bayesian Reasoning (BPE)

Function Approximation (MLE & MAP) ML

Generative models can go wrong when there are outliers.



SVM, for example, can handle this with ease.

Generative may work here if we remove outliers or make better choice of distribution, but that is often hard to do.

Traditional statistics: Data —statistics→ World state Estimate —Decision Theory→ Predictions

Machine Learning: Data → Predictions

Generative	Discriminative
Goal: To explain data (concept learning)	Goal: To make predictions (shortcut coaching)
More accurate when they use correct models	More robust against outliers & bad models
Explainable	Blackbox as we use more complex models
Generate realistic data samples (artwork, super-resolution, colorization, etc.)	
simulation & planning for reinforcement learning	

Generative Adversarial Networks (GANs)

GANs

Generators vs. **Discriminators**

Generate new data (fake) to fool the discriminator into believing that samples ~~data~~ generated is real

Identify fake (new generated) samples & force the generator to improve.

Applications: Art AI
Imaginary fashion models
Model dark matter distribution & predict Gravitational lensing

DeepFakes!!

MNIST handwritten digits

Bayesian Networks - Help in identifying how causes ~~for~~ generate ~~the~~ outcomes/events.
[work both ways bottom up & top down]

PRINCIPAL COMPONENT ANALYSIS [PCA] – I

Main objective is dimensionality reduction, i.e. to reduce the number of variables while preserving max info
Some loss of accuracy but leads to major simplifications in data analysis.

Step 1 : Feature Scaling

Important since other features with wider range will dominate.

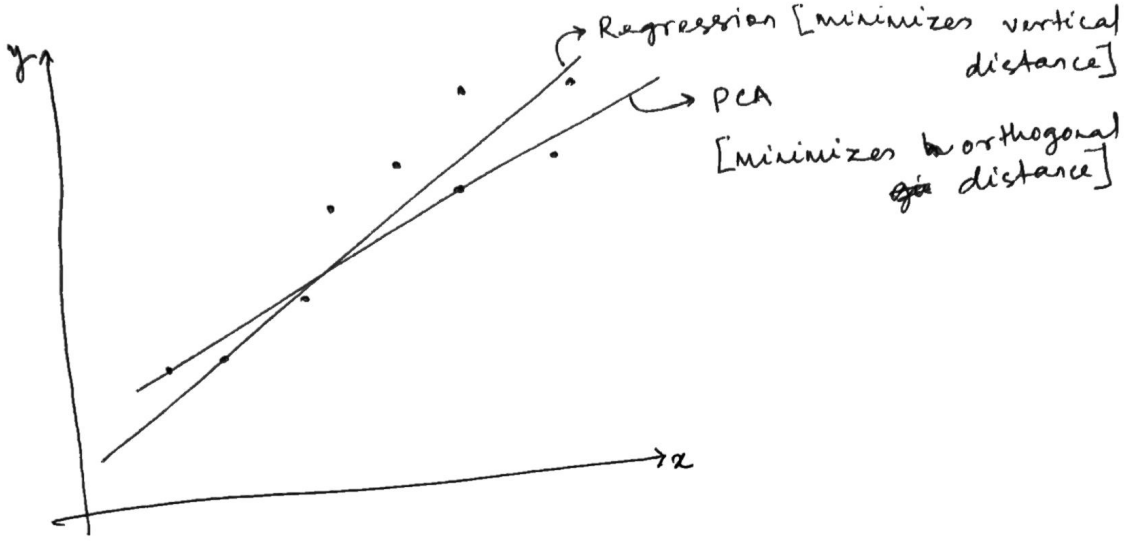
Step 2 : Covariance Matrix Computation

Variables can be correlated.
Covariance helps in finding these relationships

Step 3 : Find eigen values & eigen vectors of covariance matrix

- Eigen vectors are linear combinations of data points & chosen such that they are independent of each other.
- 10-dimensional data will give 10 eigen vectors.
- Represent directions that explain maximum amount of data.
- First principal component accounts for max variance & then the 2nd one & so on.
- Find eigen vectors $\{\vec{v}_i\}$ & eigen values $\{\lambda_i\}$
  Arrange $\{\lambda_i\}$ in decreasing order. $\lambda_i > \lambda_{i+1}$
  So $\vec{v}_1$ corresponds to first component
  $\vec{v}_2$ to second component, & so on.
- Keep a subset of the eigen vectors & discard the rest. This leads to dimensionality reduction since all the data points are now projected onto these eigen vectors through dot product.

Fraction of variance in data explained/captured by $\vec{v}_i = \dfrac{\lambda_i}{\sum_i \lambda_i}$


Regression [minimizes vertical distance]
PCA
[minimizes orthogonal distance]
y
x

PCA - II

$\{\vec{x}_n\}$ Data points $n = 1, 2, \ldots, N$
↓ column vectors
of Dimensionality D.
to be projected to dimension $M < D$.

Let $M = 1$ for simplicity

Need to find direction $\vec{u}_1$ of projected data.

Projected data points $\{\vec{y}_n\} = \{u_1^T x_n\}$

Mean: $\bar{y} = u_1^T \bar{x}$, where $\bar{x} = \frac{1}{N}\sum_n x_n$

Variance: $S_y = \frac{1}{N}\sum_n \{u_1^T x_n - u_1^T \bar{x}\}^2$

$= \frac{1}{N}\sum_n (u_1^T x_n - u_1^T \bar{x})(u_1^T x_n - u_1^T \bar{x})^T$

$= \frac{1}{N}\sum_n (u_1^T x_n - u_1^T \bar{x})(x_n^T u_1 - \bar{x}^T u_1)$

$= \frac{1}{N}\sum_n u_1^T (x_n - \bar{x})(x_n^T - \bar{x}^T) u_1$

$= u_1^T S_x u_1$, where $S_x = \frac{1}{N}\sum_n (x_n - \bar{x})(x_n - \bar{x})^T$

Objective: Maximize S_y keeping $\|u_1\| = 1$ fixed.

∴ find u_1 which maximizes

$L = u_1^T S_x u_1 + \lambda_1 (1 - u_1^T u_1)$
↳ Lagrange multiplier.

$\nabla_{u_1} L = 0 \Rightarrow S_x u_1 = \lambda_1 u_1 \Rightarrow u_1$ is the eigen vector of S_x & λ_1 is the eigen-value.

$\Rightarrow S_y = u_1^T S_x u_1 = \lambda_1$

$\Rightarrow$ choose u_1 to be the eigen-vector with max eigen-value so that S_y is maximised.

Same way $u_2, u_3, \ldots$ can be chosen.

If $M = D$, PCA still works & is a rotation of the coordinate axis to maximise variance along the new directions or components.

What if $D \gg N$? Very high dimensional data (small dataset of images)

Finding eigen vectors of $D \times D$ matrix has computational cost $\mathcal{O}(D^3)$.

Define X: $(N \times D)$ dimensional centred matrix
n^{th} row given by $(x_n - \bar{x})^T$

Covariance Matrix: $S = \frac{1}{N} X^T X$ → $D \times D$ dimensional

$$\frac{1}{N} X^T X u_i = \lambda_i u_i$$

$$\Rightarrow \frac{1}{N} X X^T X u_i = \lambda_i \underbrace{X u_i}_{v_i} \quad : \text{Multiplying both sides by } X$$

$$\Rightarrow \frac{1}{N} X X^T v_i = \lambda_i v_i$$

→ $N \times N$ Dimensional
$\mathcal{O}(N^3)$ instead of $\mathcal{O}(D^3)$
has the same eigen values as S
[$D - N + 1$ eigen values of S are zero]

One limitation of PCA is that it is limited to linear transformations of data.
Can use kernel PCA to do non-linear transformations.

$x_n = \begin{pmatrix} x_n^{(1)} \\ x_n^{(2)} \end{pmatrix}$ $\quad n = 1, 2, \cdots, N$

$D = 2$

To be projected to $M = 1$

$y_n = u^T x_n \qquad u = \begin{pmatrix} u^{(1)} \\ u^{(2)} \end{pmatrix}$

y-Variance: $S_y = \frac{1}{N} \sum_n (y_n - \bar{y})^2$

$= \frac{1}{N} \sum_n \left(u^T x_n - u^T \bar{x} \right)^2$

$= \frac{1}{N} \sum_n \left(u^T x_n - u^T \bar{x} \right) \left(u^T x_n - u^T \bar{x} \right)^T$

$= \frac{1}{N} \sum_n \left(u^T x_n - u^T \bar{x} \right) \left(x_n^T u - \bar{x}^T u \right)$

$= \frac{1}{N} \sum_n \left(u^T x_n x_n^T u - u^T x_n \bar{x}^T u - u^T \bar{x} x_n^T u - u^T \bar{x} \bar{x}^T u \right)$

$= \frac{1}{N} \sum_n u^T (x_n - \bar{x})(x_n - \bar{x})^T u$

$= u^T \underset{2 \times 2}{S_x} u$

Maximize S_y while keeping $\|u\| = 1$, constant.

$\Rightarrow S_x u = \lambda u$

$\Rightarrow S_y = u^T \lambda u = \lambda$.

for $D > 2$ & $M > 2$, choose eigen vectors corresponding to max eigen-value of the co-variance matrix.

K-MEANS CLUSTERING

Scheme to divide N data points $\{x_j\}$ into K clusters $\{c_i\}$

$r_{ji} = \begin{cases} 1, & \text{if } x_j \text{ is assigned to } c_i \\ 0, & \text{if } x_j \text{ is assigned to other clusters} \end{cases}$

Distortion Measure (Loss function): $J = \sum_{j=1}^{N} \sum_{i=1}^{K} r_{ji} \| \vec{x}_j - \vec{\mu}_i \|^2$

Find r_{ji} & $\vec{\mu}_i$ which minimize J.

Iterative Process

step 0 - choose $\vec{\mu}_i$ randomly from given data points.

step 1 - $r_{ji} = 1$ for (j, i) such that $\vec{\mu}_i$ is the closest mean vector for given point $\vec{x}_j$
else $r_{ji} = 0$

step 2 - for r_{ji} estimated in step 1, update $\vec{\mu}_i$

$$\nabla_{\mu_i} J = 0 \Rightarrow 2 \sum_{j=1}^{N} r_{ji} (\vec{x}_j - \vec{\mu}_i) = 0$$

$$\Rightarrow \vec{\mu}_i = \frac{\sum_j r_{ji} \vec{x}_j}{\sum_j r_{ji}}$$

Number of points in cluster c_i

Iterate till convergence. (guaranteed)
could be local minima

Convergence is slow & several variations available.

Lossy Data Compression

For each of the N points, only store the identity of the cluster to which it belongs.
& values of the K cluster centers, $\vec{\mu}_i$
uses significantly less data if $K \ll N$.
also called vector quantization
& $\vec{\mu}_i$ code-book vectors

Example of data compression

Each image has N pixels of {R, G, B}
values stored with 8-bit precision.
So each image needs 24N bits

K-classes needs $\log_2 K$ bits

code-book vectors need 24K bits

∴ Total no. of bits = $24K + N\log_2 K$.

∴ Compression Ratio = $\frac{24K + N\log_2 K}{24N} \times 100\%$

$= \left(\frac{K}{N} + \frac{1}{24}\log_2 K\right) \times 100\%$

$= 13.94\%$ for $K = 10$ & $N = 100 \times 100$

K-Means	KNN
Unsupervised K = no. of clusters into which the given data needs to be classified.	Supervised K = no. of nearest neighbors to use for classification of a new data point

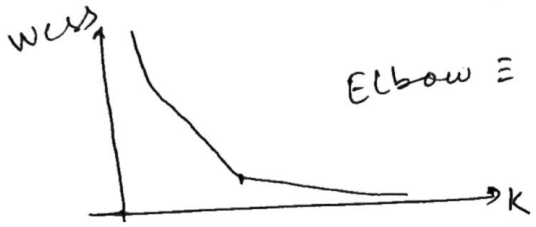
K-Mean & K-NN

K-Mean

- What will you do if number of clusters is not known?

WCSS [within Cluster Sum of Squares]

$$WCSS(K) = \sum_{i=1}^{K} \sum_{j=1}^{N} \left(x_j - y_{j|i} \mu_i\right)^2$$



Elbow ≡ Sudden change of behaviour

- Can we do image clustering using this method?
  - Similar images may have different orientation
- How to handle very large dataset?
  - use minibatches.
- How to classify a new data point?

K-NN

small k - more noisy prediction
Large K - more stable prediction

- How to use K-NN for regression?

K-Mean	K-NN
- Unsupervised	- Supervised
- K- need to be determined by elbow method	- K ~ $\sqrt{N}$
- Does not work well with cluster of different sizes	- Size difference is not a problem
- Used for clustering	- Used for classification or regression
- Eager learner	- Lazy learner

PROBABILISTIC GRAPH MODELS
[Bayesian Networks & Markov Network]

Probability Theory + Graph Theory

Directed Acyclic Graphs — Bayesian Networks
↓
AI & ~~Stats~~

Undirected Graphs — Markov ~~Net~~ Networks or MRF
[could be cyclic]
↳ Physics & Vision

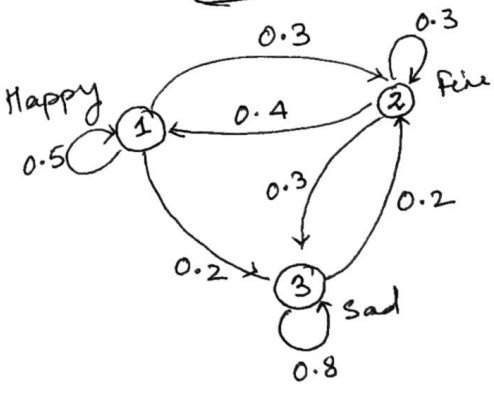
Chain Graph — both directed & undirected paths.

DAGs — Encode causal relationships
Easier to Learn from data
But cannot represent cyclic dependencies

PGM ≠ Markov chain
↳ Nodes are elements of state space & edges denote transition probabilities
↓
Nodes are random variables & edges denote conditional dependencies.

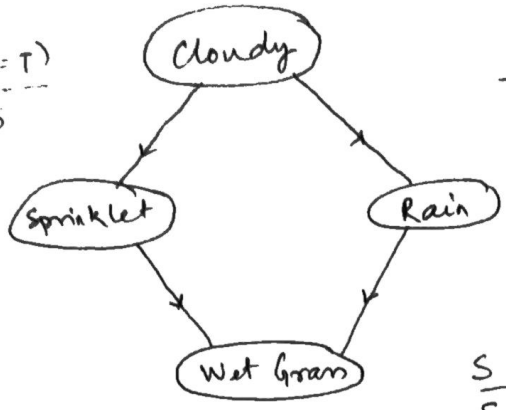
Markov: Future is independent of the past given the present.

Markov chain



P(C=F)	P(C=T)
0.5	0.5

C	P(S=F)	P(S=T)
F	0.5	0.5
T	0.9	0.1



C	P(R=F)	P(R=T)
F	0.8	0.2
T	0.2	0.8

S	R	P(W=F)	P(W=T)
F	F	1.0	0.0
T	F	0.1	0.9
F	T	0.1	0.9
T	T	0.01	0.99

~~P(S,~~

$$P(C,S,R,W) = P(C)P(S|C)P(R|C,S)P(W|C,R,S)$$

$$= P(C)\,P(S|C)P(R|C)\,P(W|R,S)$$

[Simplification of conditional probabilities]

Most common task : Probabilistic Inference.

$$P(S=T|W=T) = \frac{P(S=T \,\&\, W=T)}{P(W=T)}$$

Bottom Up →

$$= \frac{\sum_{R,C \in T/F} P(S=T \,\&\, W=T \,\&\, R=T/F \,\&\, C=T/F)}{P(W=T)}$$

Top Down ↳ $P(W=T|C=T)$

$$P(W=T) = \sum_{R,S,C \in T/F} P(S=T/F \,\&\, W=T \,\&\, R=T/F \,\&\, C=T/F)$$

$$= \frac{0.2781}{0.6471} = 0.430$$

∴ If grass is wet, it is more likely to be due to rain!

"Explaining away"

$$P(R=T|W=T) = \frac{0.4581}{0.6471} = 0.708$$

$P(S=T|W=T \,\&\, R=T) = 0.1945$ Berkson's Paradox

S & R actually are independent

Also called "Selection Bias"

EXPECTATION MAXIMIZATION

Bayesian Networks

Structure	Observability	Method
Known	Full	MLE
Known	Partial	Expectation Maximization (or Gradient Descent)
Unknown	Full	Search through Model space [NP-Hard]
Unknown	Partial	EM + Search Model space

Two coins with biases θ_A & θ_B

Pick a coin at random & toss it 10 times

Repeat this 5 times

Record $X = (x_1, x_2, \cdots, x_5)$ & $(z_1, z_2, z_3, z_4, z_5) = Z$

$x_i \in \{0, 1, 2, \cdots, 10\}$ = No. of heads observed

$z_i \in \{A, B\}$ = Coin chosen

Estimating θ_A & θ_B is easy if this full info is available

But what if z_i are <u>hidden variables</u>??

<u>Simple Approach</u> -

Assume some initial values for $\theta_A^{(0)}, \theta_B^{(0)}$

Then estimate $\{z_i\}$ based on which coin is more likely to generate the corresponding $\{x_i\}$

Estimate $\theta_A^{(1)}, \theta_B^{(1)}$ using this completed info

Repeat the above till convergence reached.

<u>EM Approach</u> - Compute probability of each possible completion of table.

Created weighted training set of all these possibilities

Then use a modified form of MLE.

$X = \{5, 9, 8, 4, 7\}$ $\quad Z = ?$

Simple Approach [MLE]

Assume $\theta_A^{(0)} = 0.60$ $\quad \theta_B^{(0)} = 0.5$

$P(x_1/A) = {}^{10}C_5 0.6^5 0.4^5 = 0.2$ | $P(x_1/B) = 0.25$

$P(x_2/A) = {}^{10}C_9 0.6^9 0.4^1 = 0.04$ | $P(x_2/B) = 0.0098$

$P(x_3/A) = {}^{10}C_8 0.6^8 0.4^2 = 0.12$ | $P(x_3/B) = 0.044$

$P(x_4/A) = {}^{10}C_4 0.6^4 0.4^6 = 0.11$ | $P(x_4/B) = 0.205$

$P(x_5/A) = {}^{10}C_7 0.6^7 0.4^3 = 0.22$ | $P(x_5/B) = 0.117$

$$\theta_A^{(1)} = \frac{9+8+7}{30} = 0.8 \qquad \theta_B^{(1)} = \frac{5+4}{20} = 0.45$$

Expectation Maximization

another way to estimate MLE/MAP

→ E-step

x_i	$P(x_i/A)$	$P(x_i/B)$	ω_{Ai}	ω_{Bi}	A	B
5	0.2	0.25	0.45	0.55		
9	0.04	0.0098	0.80	0.20		
8	0.12	0.044	0.73	0.27		
4	0.11	0.205	0.35	0.65		
7	0.22	0.117	0.65	0.35		

$\omega_{Ai} = \dfrac{p(x_i/A)}{p(x_i/A) + p(x_i/B)}$ $\qquad \omega_{Bi} = 1 - \omega_A$

M-step

$$\theta_A^{(1)} = \frac{\sum x_i \omega_{Ai}}{10 \sum \omega_{Ai}} = 0.712$$

$$\theta_B^{(1)} = \frac{\sum x_i \omega_{Bi}}{10 \sum \omega_{Bi}} = \frac{\sum x_i (1 - \omega_{Ai})}{10 \sum (1 - \omega_{Ai})} = \frac{\sum x_i - \sum x_i \omega_{Ai}}{10(5 - \sum \omega_{Ai})}$$

$= 0.583$

After 10 steps converges to $\theta_A^{(10)} \approx 0.80$

$\theta_B^{(10)} \approx 0.52$

May converge to local maxima for non-concave functions.

but convergence is guaranteed.

GAUSSIAN MIXTURE MODELS

Feature Extraction from speech data

Multi-object tracking

Essentially, used when data is multi-modal, i.e. there are many "peaks" in the distribution.

$$P(\vec{x}) = \sum_{i=1}^{K} \phi_i \mathcal{N}(\vec{x} \mid \vec{\mu}_i, \Sigma_i)$$

($\vec{\mu}_i$: Mean vector; Σ_i: Covariance Matrix)

Multi-variate Gaussian Distribution.

$$\mathcal{N}(\vec{x} \mid \vec{\mu}_i, \Sigma_i) = \frac{1}{\sqrt{(2\pi)^K |\Sigma_i|}} \exp\left[-\frac{1}{2}(\vec{x}_i - \vec{\mu}_i)^T \Sigma_i^{-1} (\vec{x}_i - \vec{\mu}_i)\right]$$

$\sum \phi_i = 1$ → weights [ratio (fraction) of points in each category]

Parameters estimated using Expectation Maximization since analytic solution is usually not possible.

EM is guaranteed to converge.

EM Algo.

E-step: using $\phi_i, \vec{\mu}_i$ & Σ_i

estimate $p(C_i/x_j)$ for each data point

↳ K classes/components

M-step: using assigned C_i for each x_j, [weighted]

update $\phi_i, \vec{\mu}_i$ & Σ_i

See next page

Initialisation:

Randomly pick $\vec{\mu}_i$ from the given unlabelled data, x_j

Set variance estimates to variance of whole data

set $\phi_i = \frac{1}{K}$ $\forall i$

OR, use K-means to find initial parameter values (speeds up considerably)

MNIST - K=10 components

Each image of size NXN is a vector of size N^2 of Bernoulli distributions (one per pixel) - PRML book

E-step

$$\gamma_{ji} = \frac{\phi_i \mathcal{N}(\vec{x}_j / \vec{\mu}_i, \Sigma_i)}{\sum_{i=1}^{K} \phi_i \mathcal{N}(\vec{x}_j / \vec{\mu}_i, \Sigma_i)} = P(C_i / \vec{x}_j)$$

M-step

$$\vec{\mu}_i = \frac{1}{N_i} \sum_{j=1}^{N} \gamma_{ji} \vec{x}_j$$

$$\Sigma_i = \frac{1}{N_i} \sum_{j=1}^{N} \gamma_{ji} (x_j - \vec{\mu}_j)(\vec{x}_j - \vec{\mu}_j)^T \rightarrow \text{updated value}$$

$$\phi_i = \frac{N_i}{N}, \qquad N_i = \sum_{j=1}^{N} \gamma_{ji} : \text{Effective no. of points in category 'i'}$$

Log-Likelihood

$$\ln p(X / \mu, \Sigma, \phi) = \sum_{j=1}^{N} \ln \left\{ \sum_{i=1}^{K} \phi_i \mathcal{N}(\vec{x}_j / \vec{\mu}_i, \Sigma_i) \right\}$$

Repeat the procen till LL or parameters converge.

EM required since analytical solution of parameters which maximise LL not available.

K-means	GMM
hard assignment of $\vec{x}_j$ to C_i, $\gamma_{ji} \in \{0, 1\}$	Soft assignment $0 \leq \gamma_{ji} \leq 1$
fast convergence even for high dimensional data	EM is slow
Assumes clusters to be spherical	Can handle ~~arbitrary~~ more complex shapes
Easier to interpret	Difficult to interpret