S. No.	Date	Title	Page No.	Teacher's Sign / Remarks

Lec1 : Supervised Learning Setup

Mathematical Notation

$$D = \{(\vec{x}_1, y_1), (\vec{x}_2, y_2), \ldots (\vec{x}_n, y_n)\} \subseteq \mathcal{X} \times \mathcal{Y}$$

$(\vec{x}_i, y_i) \sim P$

↳ distribution which we don't know.

$\mathcal{X} = \mathbb{R}^d$ (d-dimensional vector)

$\vec{x}_i \in \mathcal{X}$

$y_i \in \mathcal{Y}$

→ Examples of $\mathcal{Y}$

- Binary classification $\mathcal{Y} = \{0, 1\}$
  $\mathcal{Y} = \{-1, 1\}$
- multi-class classification : $\mathcal{Y} = \{1, \ldots K\}$
- Regression : $\mathcal{Y} = \mathbb{R}$

→ Examples of $\mathcal{X}$

patient data :

$$\vec{x}_1 = \begin{pmatrix} 1 \\ 62 \\ 183 \\ 120 \end{pmatrix} \begin{array}{l} \text{male/Female} \\ \text{age} \\ \text{Height in cm} \\ \text{Blood Pressure} \end{array}$$

$$\vec{y}_1 = [1] \quad \text{did Cure / did not Cure}$$

Lec2: Suprirsed Learning Setup Continued

$$D = \{(\vec{x}_1, y_1), \ldots (\vec{x}_n, y_n)\} \subseteq \mathcal{X} \times \mathcal{Y}$$

$\mathcal{Y}$ → space of labels

$\mathcal{X}$ → space of features

$(\vec{x}_i, y_i) \sim P$

P is illusive, if we would have known the distribution then there is no need of ML, we simply could have computed,

$$\cancel{P(y|x)} = \eta$$

Text :

$$\vec{x}_i = \begin{array}{c} \text{out} \rightarrow \\ \\ \\ \\ \\ \\ \\ \\ \text{the zetra} \rightarrow \end{array} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 5 \\ 0 \end{pmatrix} \in \mathbb{R}^{1M}$$

(take 1M words in english dict.)

Sparse Vector

↳ maximum no. of dimensions have 0.

(P) ~ training data

$\vec{x}_1, \vec{x}_2, \ldots \vec{x}_n$

$y_1, y_2, \ldots y_n$

→ ML Algo → h (program (function))

$h(\vec{x}_i) \approx y_i$

↓

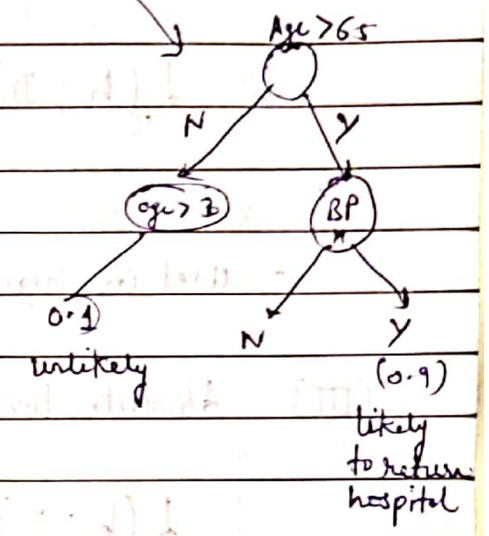
(P)

→ $h \in \mathcal{H}$ ← hypothesis Class

↳ set of possible functions we could learn.

You as a DS, its your job to pick the best.

→ Examples of h : - decision trees
- linear Classifier
- ANN
- SVM



Loss functions

Loss function is for introducing some kind of metric that we can measure the 'cost' of an incorrect decision.

(1) 0/1 - loss

$$\mathcal{L}_{0/1}(h; D) = \frac{1}{n} \sum_{(\vec{x}_i, y_i) \in D} \delta\left[h(\vec{x}_i) \neq y_i\right]$$

↳ $\begin{cases} 1 & ; \text{if } h(\vec{x}_i) \neq y_i \\ 0 & ; \text{otherwise} \end{cases}$

for instance,

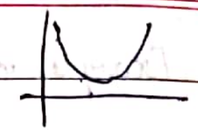
100 observation, 3 we got wrong. So we get,

$$\frac{1}{100} \times (1+1+1+0+0 \ldots) = \frac{3}{100}$$

3% of training set observations are wrong.

- lower is better.
- It can never be negative.

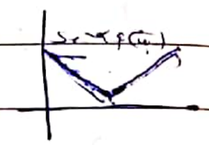
(II) Squared-loss



$$L_{sq}(h;D) = \frac{1}{n}\sum_{(\vec{x}_i,y_i)\in D}\left(h(\vec{x}_i)-y_i\right)^2$$

- used in regression problems.

(III) Absolute loss



$$L_{ul}(h;D) = \frac{1}{n}\sum_{(\vec{x}_i,y_i)\in D}\left|h(\vec{x}_i)-y_i\right|$$

- Using squared error is easier to solve, but using the absolute error is more robust to outliers.
- MAE is useful if the training data is corrupted with outliers.
- Big problem is gradient is constant throughout. while in MSE, gradient is large for higher loss & converges to zero, when loss is minimum.

→ Generalisation

- We are interested in expected value of loss.

$$E\left[L(h;(\vec{x},y))\right]_{(\vec{x},y)\sim P}$$

but since we donot have access to P.

TRAIN	TEST

↳ $h(x)$ Compute loss on test set

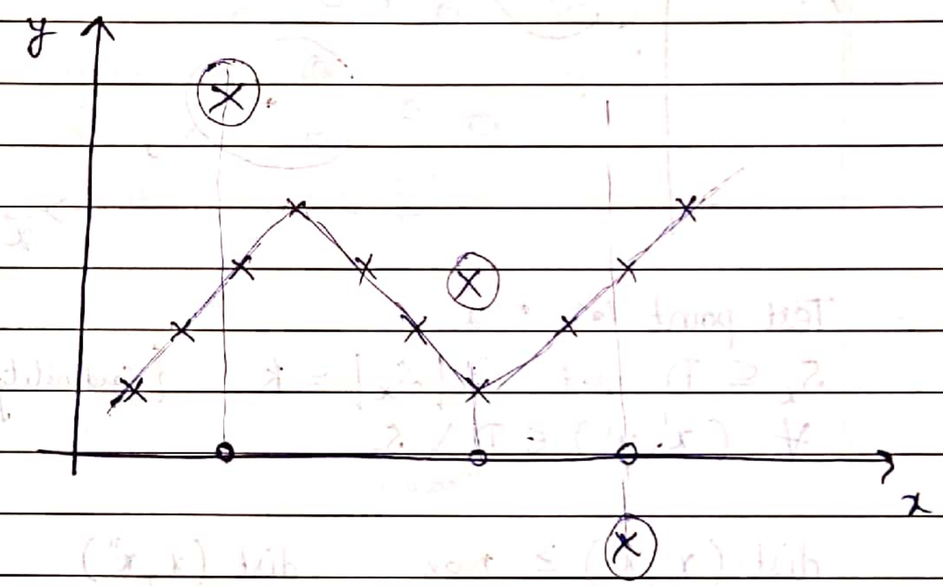
Lec 3 : K-nearest neighbors

D_{Tr}	D_{Val}	D_{Test}

$$E\left[l(h,(\vec{x},y))\right]_{(\vec{x},y)\sim P} \xleftarrow{n\to\infty} l(h,D_{Test}) = \frac{1}{n}\sum_{i=1}^{n} l(h,(x_i,y_i))$$

(∵ Weak law of large numbers)

→ $\mathcal{H}$



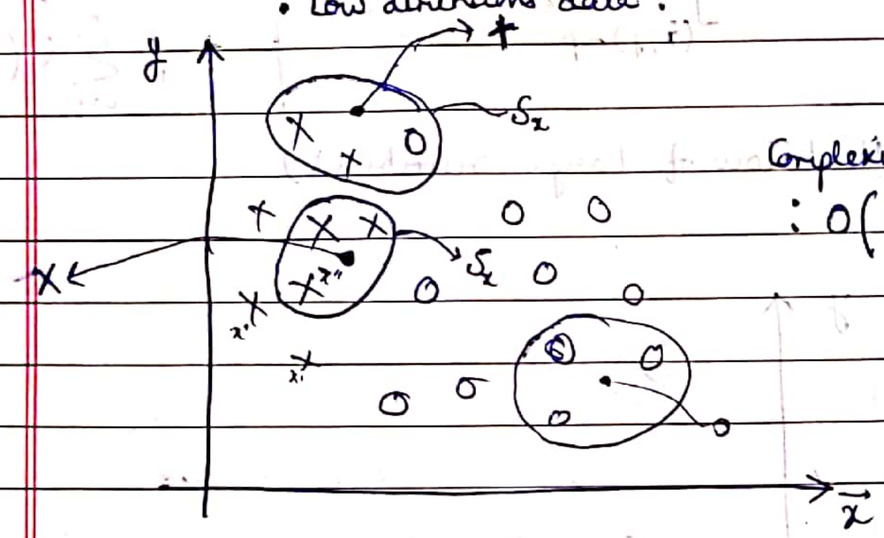
Here we have the data & we predicted a h from $\mathcal{H}$. but when we feeded test data to h, it was not good as expected.

So, we have assumption for different h. We make some assumptions drawn from the data & our h leverages those assumptions.

So, see data first & rule out lot of h if those assumptions are not according to specific h.

KNN (1967)

- Classifier / Regressor (Here we take mean of k-points)
- Assumptions :- data which is close to one class will belong to that class.
  - Similar pts. have similar labels.
  - Low dimensions data.



Complexity : $O(nd)$

Test point (•) : x

$S_x \subseteq D$ s.t. $|S_x| = k$ {Cardinality of $S_x = k$}

$\forall (x', y') \in D \setminus S_x$ (not in)

$$\text{dist.}(x, x') \geq \max_{(x'', y'') \in S_x} \text{dist.}(x, x'')$$

Minkowski Distance

$$\text{dist}(x, z) = \left(\sum_{r=1}^{d} \left| [x]_r - [z]_r \right|^p \right)^{1/p}$$

$r=1$ Age, $r=2$ Height

$x = \{1, 2\}$ $\quad x = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$ Age, Height (2×1) $\quad z = \begin{pmatrix} 3 \\ 4 \end{pmatrix}$ Age, Height (2×1) → d

$z = \{3, 4\}$

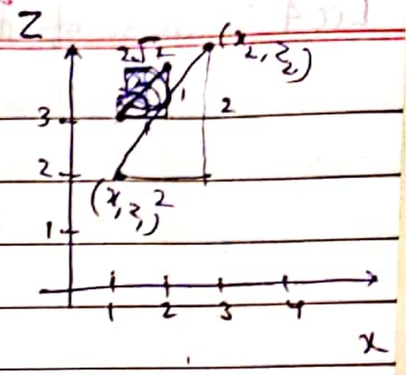
$p = 1$

$$\text{dist}(x, z) = \sum_{r=1}^{d} \left| [x]_r - [z]_r \right|$$

$$= \left| [x]_1 - [z]_1 \right| + \left| [x]_2 - [z]_2 \right|$$

$$= |1-3| + |2-4|$$

$$= 2 + 2 = 4$$



→ Manhattan distance

$p = 1$

$$\text{dist.}(x, z) = \sum_{r=1}^{d} \left| [x]_r - [z]_r \right|$$

→ Euclidian distance

$p = 2$

$$\text{dist.}(x, z) = \sqrt{\sum_{r=1}^{d} \left| [x]_r - [z]_r \right|^2}$$

→

$p = \infty \approx 1000$

diff. in age : 32

diff in Height : 15

$$\left(32^{1000} + 15^{1000} \right)^{\frac{1}{1000}}$$

$$\left(32^{1000} \right)^{\frac{1}{1000}} = 32$$

$p \to \infty$

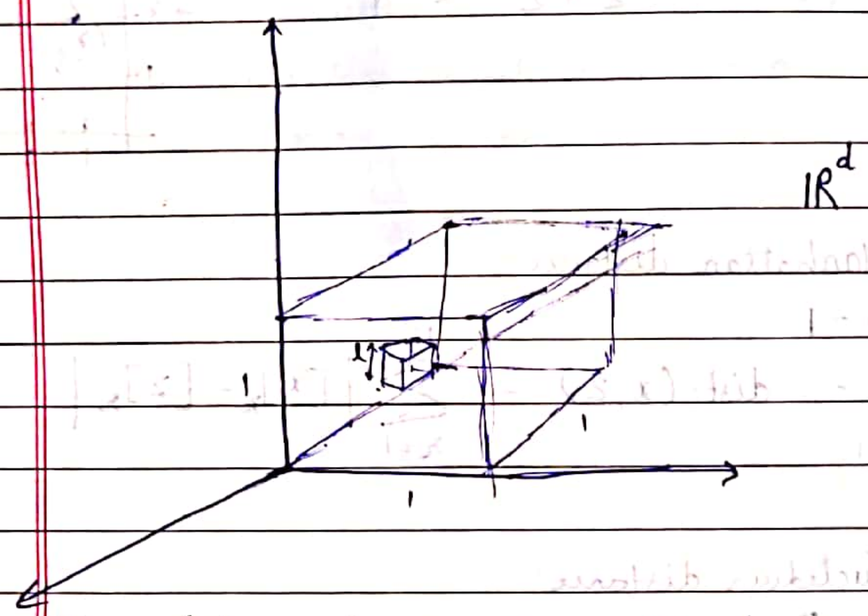
max distance of coordinates.

$$\lim_{p \to \infty} \text{dis}(x, z) = \max_{r = 1, 2, \ldots d} \left| [x]_r - [z]_r \right|$$

- All above distances can only be used for Cont. data
- for categorical, use Hamming distance.

$d(1101, 1001) = 1101 \oplus 1001 = 0100 = 1$ (No. of one)

Lec4: Curse of dimensionality / Perceptron



- Draw data uniformly within this hypercube (unit vol)
We would like to know the smallest cube of length l that contains K-nearest neighbours.

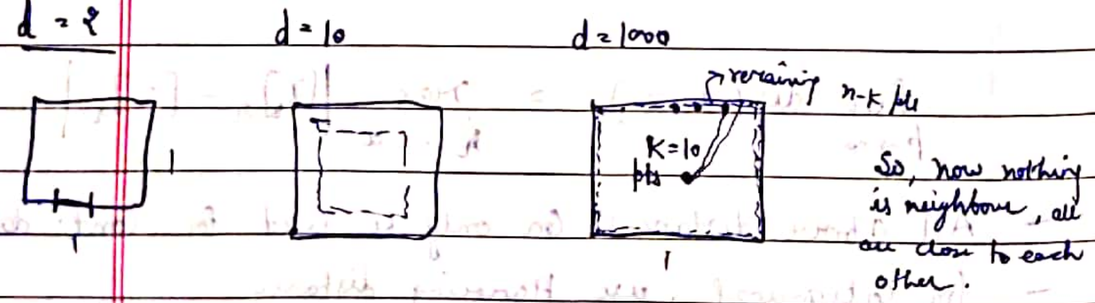
$$l^d \simeq \frac{K}{n}$$

$$l \simeq \left(\frac{K}{n}\right)^{1/d}$$

Say, $K = 10$, $n = 100$

$$l = \left(\frac{1}{100}\right)^{\frac{1}{d}}$$

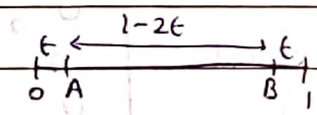
d	l
2	0.1
10	0.63
100	0.955
1000	0.9956



So, now nothing is neighbour, all are close to each other.

When the test data is inside the dotted volume & its K neighbours are ~~no~~ on the edge, ~~of~~ & remaining $n-K$ are also at the same distance. Then it will become a problem to select which K-points.

Intuition:



probability of lying on line AB is $1-2\epsilon$

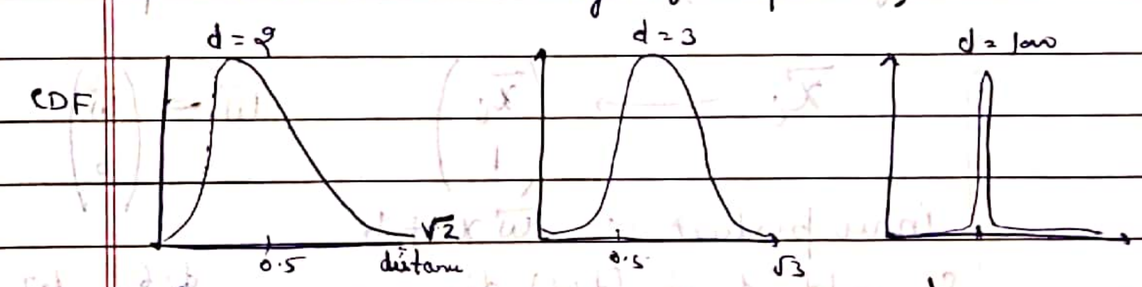
In d-dimension, prob. is $(1-2\epsilon)^d$,

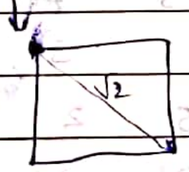
$$\lim_{d\to\infty} (1-2\epsilon)^d = 0$$

So, everything is on the edge.

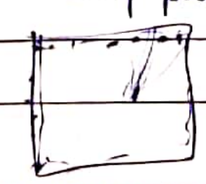
*** In high dimensional space, KNN fails miserably.

- PCA, LDA: dimensionality reduction should be in your tool bucket during data preprocessing, because most of the data doesn't require so many dimensions (like an image of a person).





distance is same for every point.

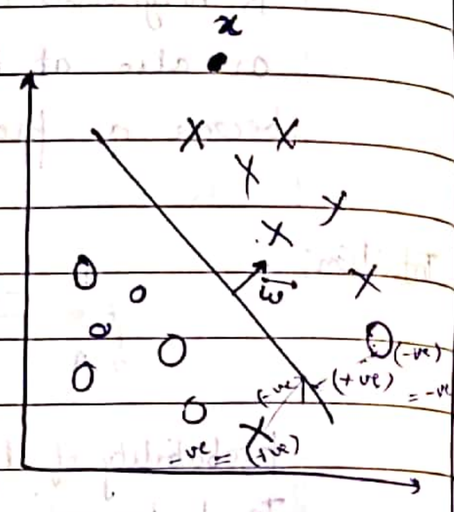


Perceptron (1957) [Ist ML Algo]

Assumptions:

- There has to have a hyperplane which separates the Classes.

In low dimensions, it might not hold but in high dimensions, it holds better.



$$\mathcal{H} = \{ x : w^T x + b = 0 \}$$

Check Sign

$(w^T x + b) > 0 \longrightarrow$ X

$(w^T x + b) < 0 \longrightarrow$ O

$(w^T x + b) = 0 \longrightarrow$ on the hyperplane

$$\mathcal{Y} = \{ -1, +1 \}$$

* We need to learn two things (w & b)

To get rid of two, we can do a transformation.

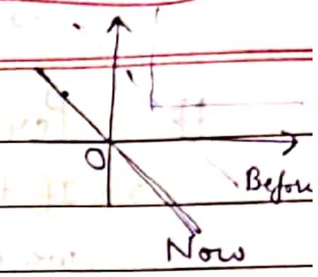
$$\vec{x}_i \longrightarrow \begin{pmatrix} \vec{x}_i \\ 1 \end{pmatrix} \qquad \vec{w} \Rightarrow \begin{pmatrix} \vec{w} \\ b \end{pmatrix}$$

inner product : $\vec{w} x + b$

So, add 1 in (d+1) dimension. d=6 d=7

$\vec{x}_1$	1	2	3	5	7	2	1
$\vec{x}_2$	5	8	2	1	3	0	1
$\vec{x}_3$	3	2	1	5	2	5	1
$\vec{x}_4$	2	3	7	3	2	4	1

$$\mathcal{H} = \{ x : w^T x = 0 \}$$



Perceptron algorithm

lec5:

initilise $\vec{w} = \vec{0}$

while True :

$m = 0$

for $(\vec{x}_i, y_i) \in D$:

if $y_i (\vec{w}^T \cdot \vec{x}_i) \le 0$: $\longrightarrow$ if the pair is misclassified

$\vec{w} \leftarrow \vec{w} + y_i \vec{x}_i$

$m \leftarrow m + 1$

if $m = 0$:

break

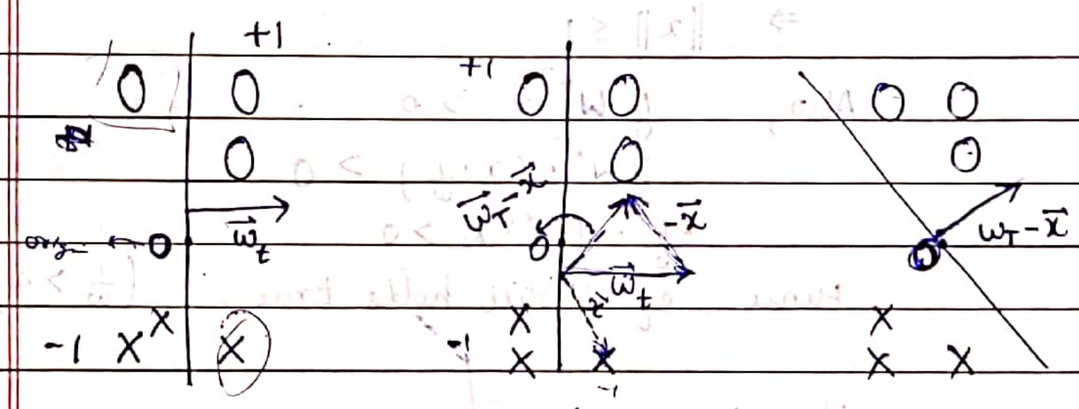
$\vec{w} = \vec{0}$

$(w + x)^T x$

$w^T x + x^T x$

making $w^T x$ bigger s.t. $y_i w^T x \ge 0$

Geometric Intuition



$w^T x \le 0$ (wrong x)

$(w + x)^T x \le 0$

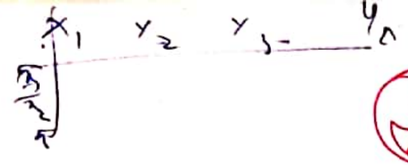
$(w + kx)^T x \le 0$

$w^T x + k x^T x \le 0$

$k \le \dfrac{-w^T x}{x^T x}$ After k iteration we get correct hyperplane.

Scanned with CamScanner

Lec 6



$W \to$ dabbu
$\omega \to$ omega

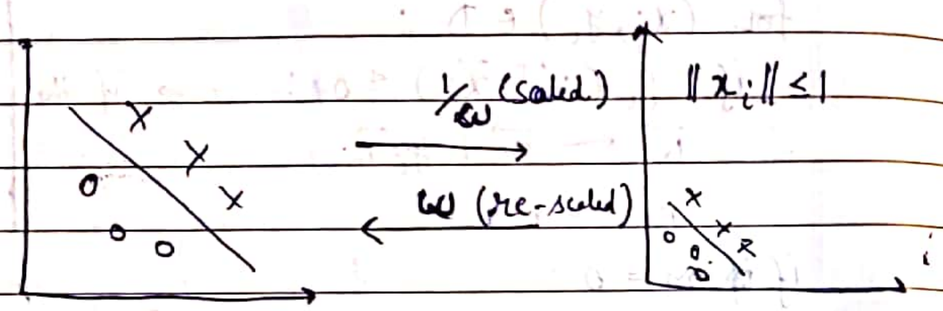
Perceptron Convergence Proof

> If there exists a linearly separable hyperplane, we are going to get it.

Assumption :

Theorem : $\exists\, w^*$ s.t. $\forall\, (\vec{x}, y) \in D \quad y w^T x > 0$

interested in $\|w^*\| = 1$



$$\omega = \max_x \|x\|$$

Scaling : $x \in x_i \cdot \frac{1}{\omega}$

$\Rightarrow \|x\| \le 1$

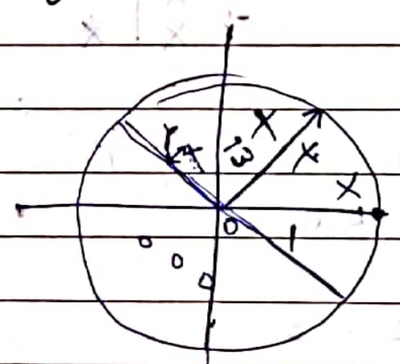
Also, $y w^{*T} \cdot x > 0$ ——— (*)

$y w^{*T} \cdot (x \cdot \frac{1}{\omega}) > 0$

$\frac{1}{\omega} \cdot y w^{*T} x > 0$

Hence, eq (*) still holds true. $(\frac{1}{\omega} > 0)$

idea :



Every datapoint lie in unit ~~circle~~ Sphere

- Margin (γ) – dist. from the closest pt. to hyperplane

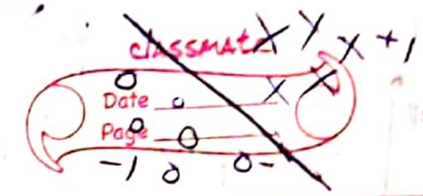
$$\gamma = \min_{(x,y) \in D} \frac{|x^T w^*|}{\|w^*\|} > 0$$

$H = \{ x : w^T x = 0 \}$

$y = +1 \Rightarrow w^T x > 0$
$y = -1 \Rightarrow w^T x < 0$ $\Big\}\ y w^T x > 0$

If we get $y w^T x < 0$;

$y = +1 \Rightarrow w \leftarrow w + x$
$y = -1 \Rightarrow w \leftarrow w - x$ $\Big\}\ w \leftarrow w + yx$



proof : $y w^T x \le 0$

- update : $w \leftarrow w + yx$

- Check what happens to $w^T w^*$

$$w^T w^* = (w + yx)^T w^*$$
$$= w^T w^* + \underbrace{y w^{*T} x}_{\ge \gamma}$$
$$w^T w^* \ge w^T w^* + \gamma$$

It changes by atleast γ every time we update w.

- Check what happens to $w^T w$

$$w^T w = (w + yx)^T (w + yx) = w^T w + \underbrace{2 y w^T x}_{<0} + \underbrace{y^2 x^T x}_{<1}$$
$$w^T w \le w^T w + 1$$

It changes by atmost 1 every time we update w.

- After M updates :

we know, $w^T w^* \ge w^T w^* + \gamma$ $\quad (w = 0$ at $t = 0)$

$$M\gamma \le w^T w^* = \underbrace{|w^T w^*| \le \|w\| \|w^*\|}_{\text{Cauchy Schwartz inequality}} = \|w\|$$

$$|\vec{a} \cdot \vec{b}| = |\vec{a}| \cdot |\vec{b}| \cos\theta$$
$$\le |\vec{a}|^2 |\vec{b}|$$

$$M\gamma \le \|w\| = \sqrt{w^T w} \le \sqrt{M}$$

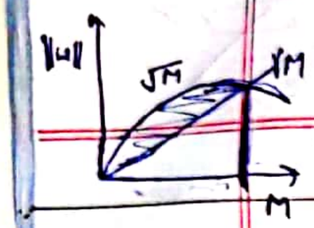
$$M\gamma \le \sqrt{M}$$

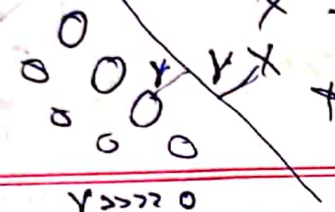
$$\boxed{M \le \frac{1}{\gamma^2}}$$

M must be less than $\frac{1}{\gamma^2}$ (constant)

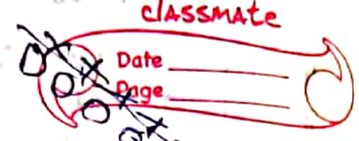
Number of steps required depends on how far two classes are separated.

Scanned with CamScanner





$\gamma >>>> 0$
M is less



$\gamma \approx 0$
$M \to \infty$

Lec 7: Estimating Probabilities from Data : MLE

- Classical Statistics
- Estimate distributions from the data.

- The purpose of MLE is to find the parameters which are most likely to have produced the observed data.

Simple Scenario : Coin toss

$D = \{\underset{x_1}{H}, T, T, H, H, H, T, T, T, \underset{x_n}{T}\}$

Estimate $P(H) = ?$

$$P(H) \simeq \frac{n_H}{n_H + n_T} = \frac{4}{10} = 0.4$$

→ Derive this more formally (MLE)

Step 1 : Assume the type of distribution your data was sampled from.

Step 2 : Set the parameters of this distribution so that the data you observed is as likely as possible.

problem : Coin toss
distribution : binomial distribution
parameters : n, θ

$n \to$ no. of independent tosses
$\theta \to$ prob. of Head ($P(H) = \theta$)

Formally, binomial dist. is defined as :

$$P(D;\theta) = \binom{n_H + n_T}{n_H} \theta^{n_H} \cdot (1-\theta)^{n_T}$$

it computes the probability that we would observe exactly n_H heads, n_T tails, if a coin was tossed $n = n_H + n_T$ times & its prob. of coming up heads is θ.

- MLE Principle : Find $\hat{\theta}$ to maximise the likelihood of the data, $P(D;\theta)$:

$$\hat{\theta} = \arg\max_{\theta} P(D;\theta)$$

{ At what θ, $P(D;\theta)$ is maximum

$$\hat{\theta} = \arg\max_{\theta} \log(P(D;\theta))$$

$$L = \log\binom{n_H + n_T}{n_H} + n_H \log\theta + n_T \log(1-\theta)$$

$$L' = 0$$

$$\frac{n_H}{\theta} + \frac{n_T}{(1-\theta)} = 0$$

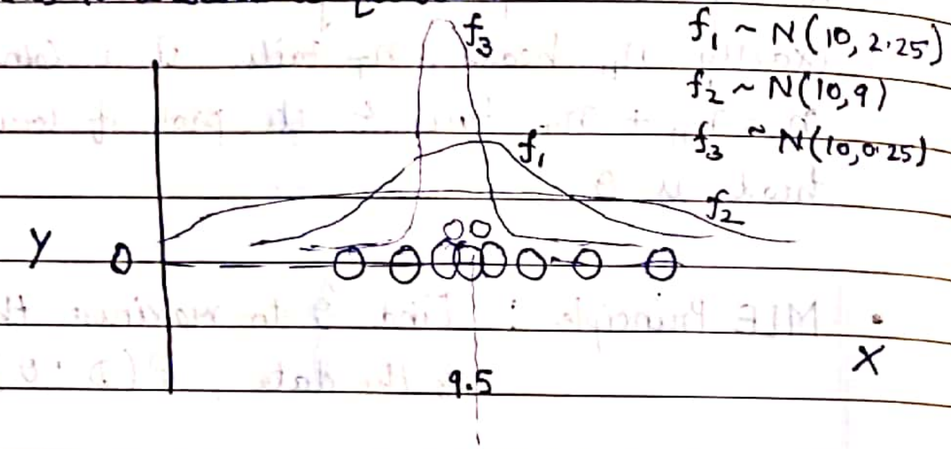
$$\hat{\theta} = \frac{n_H}{n_H + n_T}$$

- MLE gives the explanation of the data you observed.
- If n is large, MLE finds the true parameter.

(∵ If we have all T ; $\hat{\theta} = 0$ which can not be the case)

Scanned with CamScanner

Example : We have observed 10 data points . Each data pt. represents the length of time in seconds that it takes to answer a question .



$f_1 \sim N(10, 2.25)$
$f_2 \sim N(10, 9)$
$f_3 \sim N(10, 0.25)$

Step 1 : Data is drawn from Gaussian distribution.
Step 2 : parameters : μ, σ

→ Calculating Maximum likelihood Estimates

Suppose we have 3 data points

$$D = \{9, 9.5, 11\}$$

- We want to calculate the total prob. of observing all of data D . i.e. the joint prob. distribution of all observed data points .
- Assume each data point is independent of each other .

$$P(x; \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right)$$

$$P(D; \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x_1-\mu)^2}{2\sigma^2}\right) \times \frac{1}{\sigma\sqrt{2\pi}} \exp\left(\frac{-(x_2-\mu)^2}{2\sigma^2}\right) \cdots$$

$$P(\{x_1, x_2, \ldots x_n\}; \mu, \sigma) = P(x_1; \mu, \sigma) \cdot P(x_2; \mu, \sigma) \cdots P(x_n; \mu, \sigma)$$

$$P(\{9, 9.5, 11\}; \mu, \sigma) = P(9; \mu, \sigma) \cdot P(9.5; \mu, \sigma) \cdot P(11; \mu, \sigma)$$

$$= \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(9-\mu)^2}{2\sigma^2}\right) \cdot \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(9.5-\mu)^2}{2\sigma^2}\right) \cdot \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(11-\mu)^2}{2\sigma^2}\right)$$

$$\overbrace{\ln(P(D; \mu, \sigma))}^{L} = \ln\left(\frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(9-\mu)^2}{2\sigma^2}\right)\right) + \ln\left(\frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(9.5-\mu)^2}{2\sigma^2}\right)\right) + \ln\left(\frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(11-\mu)^2}{2\sigma^2}\right)\right)$$

$$L = -3\ln(\sigma) - \frac{3}{2}\ln(2\pi) - \frac{1}{2\sigma^2}\left[(9-\mu)^2 + (9.5-\mu)^2 + (11-\mu)^2\right]$$

$$\frac{\partial L}{\partial \mu} = 0 \qquad \frac{\partial L}{\partial \sigma} = 0$$

$$\mu = \frac{9 + 9.5 + 11}{3} \qquad \sigma =$$

SOURCE : MEDIAN = Towards data Science.

Scanned with CamScanner

Continue Lec 7

Simple Scenario : Coin toss with prior Knowledge

- Assume you have a hunch that θ is close to 0.5. But your sample size is small, so you don't trust your estimate.

Simple fix : Add 'm' imaginary throws ; Add m Heads & m Tails to your data

$$\hat{\theta} = \frac{n_H + m}{n_H + n_T + 2m} \quad \{+1 \text{ smoothing}\}$$

For large n, this is insignificant change.
For small-n, it incorporates your "prior belief" about what θ should be.

Classical Statistics		Bayesian Statistics
$P(D;\theta)$	$\longrightarrow$	$P(D\|\theta)$
- $\theta \to$ parameter		- $\theta \to$ random variable
- θ is not an event, where is $P(\theta)$?		- θ is associated with the prior belief.
		- If data is less, then we may see $P(\theta) \simeq 0$; which is unlikely to happen so, I want to incorporate prior belief in the estimation

NOTE : Bayesian statistics allows us to take initial prior belief into consideration.

$\to$ Which parameter makes our data more likely

$P(D|\theta)$ ← Likelihood (MLE maximises)
$P(\theta)$ ← Prior (initial belief)
$P(\theta|D)$ ← Posterior (MAP maximises this)

$\hookrightarrow$ given the Data, what is the most likely parameter

- MAP (Maximum a Posteriori Pro. Est.) : find $\hat{\theta}$ to maximise $P(\theta|D)$

Bayes Rule

$$P(\theta|D) = \frac{P(D|\theta)\,P(\theta)}{P(D)}$$

Assume $P(\theta)$ is a β-distribution

$$P(\theta) = \frac{\theta^{\alpha-1}(1-\theta)^{\beta-1}}{B(\alpha,\beta)} \quad \text{for } 0 \to 1$$

$$P(\theta|D) \propto P(D|\theta)\cdot P(\theta)$$

$$\propto \binom{n_H+n_T}{n_H}\theta^{n_H}\cdot(1-\theta)^{n_T}\cdot\theta^{\alpha-1}(1-\theta)^{\beta-1}$$

By maximising

$(\alpha-1)$ – H
$(\beta-1)$ – T

$$\boxed{\hat{\theta} = \frac{n_H+\alpha-1}{n_H+n_T+\alpha+\beta-2}}$$

$$B(\alpha,\beta) = \frac{\Gamma\alpha\cdot\Gamma\beta}{\Gamma\alpha+\beta} \qquad \Gamma \to \text{Gamma } f^n.$$

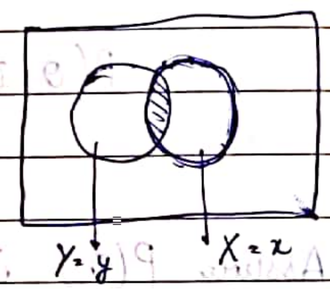
$$\Gamma n = (n-1)!$$

- The MAP estimate is identical to MLE with $\alpha-1$ hallucinated heads & $\beta-1$ hallucinated Tails.
- As $n \to \infty$, $\hat{\theta}_{MAP} \to \hat{\theta}_{MLE}$
- MAP is a great estimator if an accurate prior belief is available.

Naïve Bayes

Y – labels

X – feature



$D \sim P(Y, X)$

$$P(X=x) = \frac{\sum_{i=1}^{n} I(X_i = x)}{n}$$

$$P(Y=y \mid X=x) = \frac{P(Y=y, X=x)}{P(X=x)}$$

$$= \frac{\sum_{i=1}^{n} I(X_i = x \cap Y_i = y)}{n} \Big/ \frac{\sum_{i=1}^{n} I(X_i = x)}{n}$$

$$\boxed{P(Y=y \mid X=x) = \frac{\sum I(X_i = x \cap Y_i = y)}{\sum I(X_i = x)}}$$

→ In high dimensional data.

$P(Y=y \mid (X_1 = x_1, X_2 = x_2, \ldots X_d = x_d))$

Hardly we will find more than 1 element on conditioned set. So, not a good idea in practice.

Here, Naïve Bayes comes with a rescue.

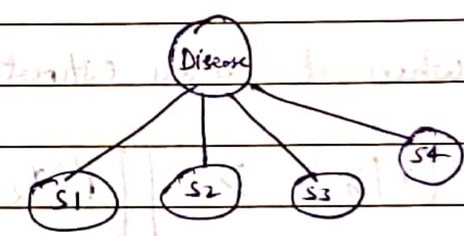
$$P(Y=y \mid X=x) = \frac{P(X=x \mid Y=y) \cdot \overbrace{P(Y=y)}^{\text{Prior}}}{P(X=x)}$$

Assumption :- features are independent given the label.
- Every feature is contributing prediction equally.

$$P(X=x \mid Y=y) = \prod_{\alpha=1}^{d} P(|\bar{X}|_\alpha = x_\alpha \mid Y=y)$$

(e.g. when there is Causal relationship then we can find independent features)

S1, S2, S3, S4 are symptoms given disease.



Bayes Classifier

$$h(\bar{x}) = \underset{y}{\operatorname{argmax}}\ P(y \mid \bar{x}) = \underset{y}{\operatorname{argmax}}\ \frac{P(\bar{x} \mid y) P(y)}{Z}$$

$$= \underset{y}{\operatorname{argmax}}\ P(y) \prod_{\alpha=1}^{d} P((\bar{x})_\alpha \mid y)$$

$$= \underset{y}{\operatorname{argmax}}\ \log P(y) + \sum_{\alpha} \log P((\bar{x})_\alpha \mid y)$$

Scanned with CamScanner

Ques : Example when it over estimates i.e.

$$P(\bar{x}|y) < \prod_{\alpha} P(x_\alpha|y)$$

Two spams : Nigeria prince & Viagra

$\bar{x} = \{$Viagra, Nigeria, Hi, if ...$\}$

$P(\bar{x} \mid y = \text{spam})$ is very low as it is hard to find Nigeria prince & Viagra in a spam mail, but when we decompose it.

$$\prod_{\alpha} P(x_\alpha|y) \text{ is high}$$

Ques : Example when it under estimate i.e.

$$P(\bar{x}|y) > \prod_{\alpha} P(x_\alpha|y)$$

Naïve Bayes in Categorical features

$P(x/y)$

$x \in \mathbb{R}^d$

$\alpha \in [0,d]$ $\quad x_\alpha \in \{F_1, F_2, \ldots F_k\}$

$$P(x_\alpha = j \mid y = c) = [\theta_{jc}]_\alpha$$

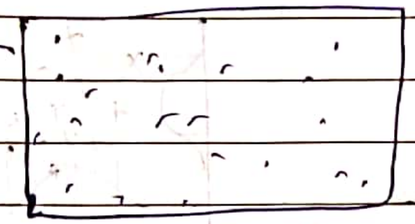
$$P(x_\alpha = j \mid y = c) = \frac{\sum_{i}^{n} I(y_i = c)\, I(x_{i\alpha} = j)}{\sum_{i=1}^{n} I(y_i = c)}$$

Spam filter uses naïve Bayes Classifier with multinomial distribution

c — Spam / Not Spam

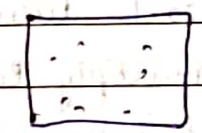
Multinomial Distribution

English words in total $(10^6) \approx d$



Start picking words randomly and store in the following box with replacement

$\mathbb{R}^d$



→ E-Mail

$x_\alpha \in \{0, 1, 2, \ldots m\}$ $\qquad m = \sum_{\alpha=1}^{d} x_\alpha$

$\bar{x} =$

x_1 'a'	5
x_2 'art'	0
x_3 'all'	0
⋮	
x_d	

m → No. of words in an email

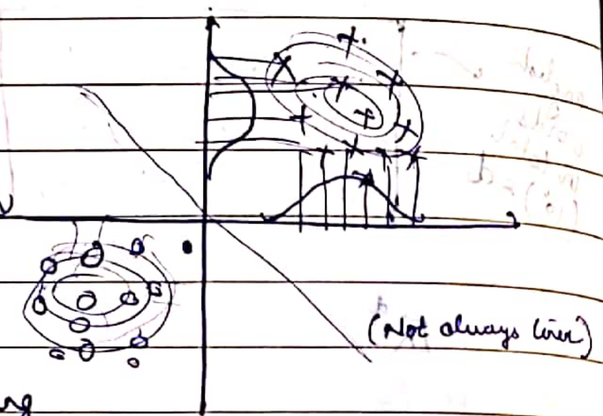
$$P(\bar{x} \mid m, y = c) = \frac{m!}{x_1! \ldots x_d!} \prod_{\alpha=1}^{d} \theta_{\alpha c}^{x_\alpha}$$

$\theta_{\alpha c}^{x_\alpha}$ → given that the class is c, prob. of drawing word α ('a') & i draw exactly x_α times.

- Use Naive bayes classifier mainly in text project.
- Cannot be used for regression.

- Naive Bayes Classifier is two step process

$P(x|y)$



① Model the distribution of each dimension & multiply each distribution; assuming they are independent, to get $P(x|y)$.

② Once we get $P(x|y)$; use bayes rule to answer the probability.

NOTE : If In step 1, Model dist. of each dimension is well looked, then we are going to get a good Naive Bayes Classifier.

Email Spam filter :

d → all english words

$$x_i = \begin{pmatrix} x_{i1} \text{ 'a'} & 3 \\ x_{i2} \text{ 'art'} & 2 \\ & 0 \\ & 5 \\ & 0 \\ x_{id} \text{ 'zebra'} & 0 \end{pmatrix} \leftarrow \mathbb{R}^d$$

$$P(\bar{x} | m, y=c) = \prod_{\alpha=1}^{d} (\theta_{\alpha c})^{x_\alpha}$$

Here, order doesn't matter that's why we have focused only on counts.

Estimate $\hat{\theta}_{\alpha c}$

$$\hat{\theta}_{\alpha c} = \frac{\sum_{i=1}^{n} I(y_i = c) x_{i\alpha} + 1}{\sum_{i=1}^{n} I(y_i = c) \underbrace{\left(\sum_{p=1}^{d} x_{ip}\right)}_{m} + d}$$

If data is Continous;

$x_\alpha \in \mathbb{R}$

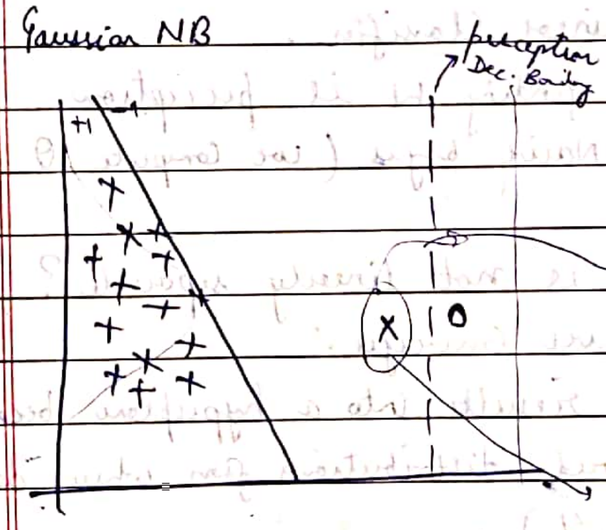
$$P(x_\alpha | y) = N(\mu_{\alpha c}, \sigma^2_{\alpha c})$$

$$\mu_{\alpha c} \leftarrow \frac{1}{n_c} \sum_{i=1}^{n} I(y_i = c) x_{i\alpha}$$

$$n_c = \sum_{i=1}^{n} I(y_i = c)$$

$$\sigma^2_{\alpha c} \leftarrow \frac{1}{n_c} \sum_{i=1}^{n} I(y_i = c)(x_{i\alpha} - \mu_{\alpha c})^2$$

Demo : Gaussian NB



Because, we assumed the dist. is gaussian but this has very low prob. of existing. (or it is an outlier)

This pt. is more likely to be drawn from ⊙ distribution

Lec11: Logistic Regression

$P(y|x) \propto P(x|y)P(y)$

$$P(y=+1|x) > P(y=-1|x)$$

$$P(y=+1)\cdot P(\bar{x}|y=+1) > P(y=-1)\cdot P(\bar{x}|y=-1)$$

$$P(+1)\cdot\left(\prod_{\alpha=1}^{d}\theta_{\alpha(+1)}^{x_\alpha}\right)\frac{m!}{x_1!\,x_2!\cdots x_d!} > P(-1)\cdot\frac{m!}{x_1!\,x_2!\cdots x_d!}\prod_{\alpha=1}^{d}\theta_{\alpha(-1)}^{x_\alpha}$$

Multinomial dist

$$\log P(+1)+\sum_{\alpha=1}^{d}x_\alpha\log\theta_{\alpha(+1)} > \log P(-1)+\sum_{\alpha=1}^{d}x_\alpha\theta_{\alpha(-1)}$$

$$\underbrace{\left[\log P(+1)-\log P(-1)\right]}_{b}+\sum_{\alpha=1}^{d}x_\alpha\underbrace{\left(\log\theta_{\alpha(+1)}-\log\theta_{\alpha(-1)}\right)}_{W_\alpha}>0$$

$$\Rightarrow W^Tx+b>0$$

- This is a linear Classifier.
- One way of finding W is perceptron.
- Another is Naive bayes (we compute θ using MLE)

Ques: What if data is not linearly separable?
- Perceptron never converges.
- Naive bayes results into a hyperplane because we have assumed distributions from where data came, (it didn't).

Plus hyperplane not separating the classes, it separates the gaussian distributions from each other (G.N.B)

$X \in \mathbb{R}^n$

	X_1	X_2	X_3	...	X_n	Y
$\bar{x}_1 =$						
$\bar{x}_2 =$						

Form of $P(y|x)$ for Gaussian Naive Bayes Classifier

- $y \in \{0,1\}$ ($y_k = 1,0$) $\quad \pi = P(Y=1)$
- $X = \langle X_1, \ldots X_n\rangle$, where each X_i is a continuous random variable
- For each X_i, $P(X_i|Y=y_k)$ is a Gaussian distribution $N(\mu_{ik}, \sigma_i)$
- X_i's are independent

$$\longrightarrow P(Y=1|X) = \frac{P(Y=1)\,P(X|Y=1)}{P(X)}$$

$$= \frac{P(Y=1)\cdot P(X|Y=1)}{P(Y=1)\,P(X|Y=1)+P(Y=0)\cdot P(X|Y=0)}$$

$$= \frac{1}{1+\dfrac{P(Y=0)\cdot P(X|Y=0)}{P(Y=1)\cdot P(X|Y=1)}}$$

$$= \frac{1}{1+\exp\left(\ln\left(\dfrac{P(Y=0)\cdot P(X|Y=0)}{P(Y=1)\cdot P(X|Y=1)}\right)\right)}$$

$$= \frac{1}{1+\exp\left(\ln\left(\dfrac{1-\pi}{\pi}\right)+\sum_{i=1}^{n}\ln\left(\dfrac{P(X_i|Y=0)}{P(X_i|Y=1)}\right)\right)}$$

Scanned with CamScanner

$i = \alpha$

We know,

$\forall P(X_i \mid Y = y_k)$ is Gaussian distribution

$$\sum_{i=1}^{n} \ln\left(\frac{P(X_i \mid Y=0)}{P(X_i \mid Y=1)}\right) = \sum_{i=1}^{n} \ln\left(\frac{\frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left(\frac{-(X_i-\mu_{i0})^2}{2\sigma_i^2}\right)}{\frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left(\frac{-(X_i-\mu_{i1})^2}{2\sigma_i^2}\right)}\right)$$

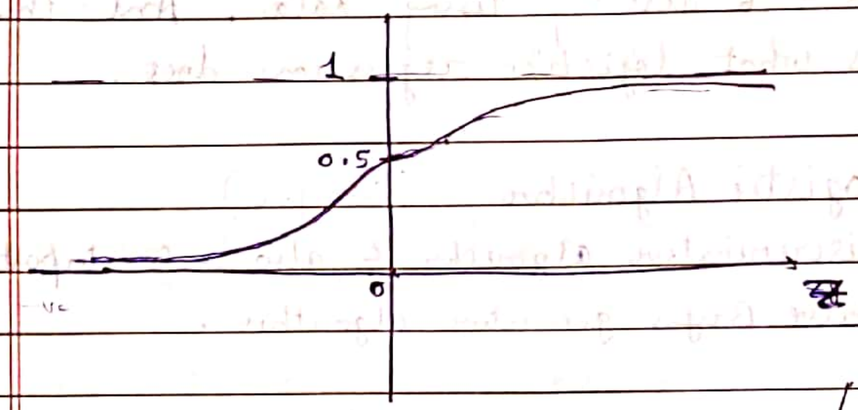
$$= \sum_{i=1}^{n} \ln \cdot \exp\left(\frac{(X_i-\mu_{i1})^2 - (X_i-\mu_{i0})^2}{2\sigma_i^2}\right)$$

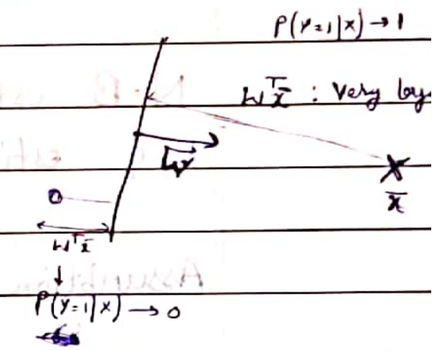
$$= \sum_{i=1}^{n} \left(\frac{\mu_{i0}-\mu_{i1}}{\sigma_i^2} X_i + \frac{\mu_{i1}^2 - \mu_{i0}^2}{2\sigma_i^2}\right)$$

$$P(Y=1 \mid X) = \frac{1}{1 + \exp\left(\ln\left(\frac{1-\pi}{\pi}\right) + \sum_{i=1}^{n} \frac{\mu_{i0}-\mu_{i1}}{\sigma_i^2} X_i + \frac{\mu_{i1}^2 - \mu_{i0}^2}{2\sigma_i^2}\right)}$$

$$\boxed{\begin{aligned} P(Y=1 \mid X) &= \frac{1}{1 + \exp\left(W_0 + \sum_{i=1}^{n} W_i X_i\right)} \\ W_0 &= \ln\left(\frac{1-\pi}{\pi}\right) + \sum_{i=1}^{n} \frac{\mu_{i1}^2 - \mu_{i0}^2}{2\sigma_i^2} \\ W_i &= \frac{\mu_{i0} - \mu_{i1}}{\sigma_i^2} \end{aligned}}$$

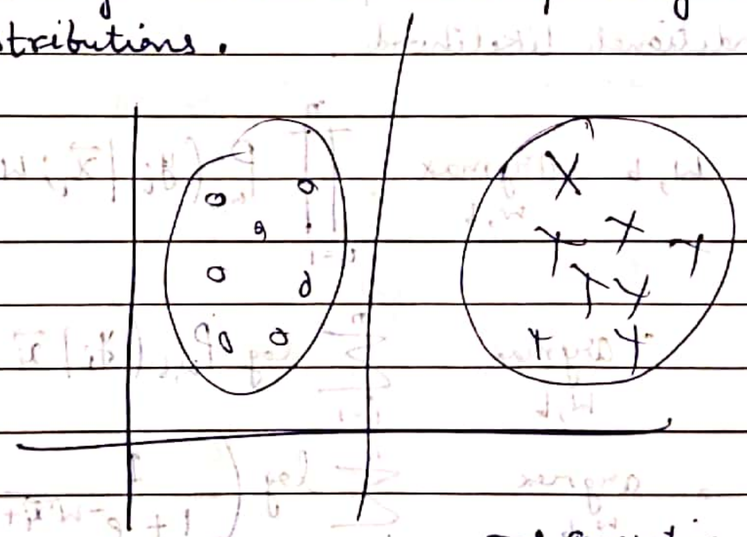
$$P(y=1 \mid \vec{x}) = \frac{1}{1 + e^{-\vec{w}^T \vec{x} + b}} \qquad y \in \{-1, 1\}$$





What we did so far?

(>) We model the distribution $P(X \mid Y = y_k)$ & estimates the parameters using MLE or MAP & got the above mentioned equation; which indirectly means we are separating the distributions.



→ generative algorithm

Now, idea is we have found out the closed form equation, why don't we estimate w & b directly using data. And this is what logistic regression does.

Logistic Algorithm (Classifier)

- discriminative algorithm & also a counterpart of Naive Bayes generative algorithm.

N.B estimates $P_\theta(x|y)$ & $P_\theta(y)$ & get $P(y|x)$
L.G estimates $P_{w,b}(y|x)$

Assumption : $P(y=1|\bar{x})$ has following form :

$$P(y=1|\bar{x}) = \frac{1}{1+\exp(-w^T x + b)}$$

→ MLE

In MLE we choose parameters that maximize the conditional likelihood.

$$w, b = \underset{w,b}{\operatorname{argmax}} \prod_{i=1}^{n} P_{w,b}(y_i | \vec{x}_i; w, b)$$

$$= \underset{w,b}{\operatorname{argmax}} \sum_{i=1}^{n} \log P_{w,b}(y_i | \vec{x}_i; w, b)$$

$$= \underset{w,b}{\operatorname{argmax}} \sum_{i=1}^{n} \log \left(\frac{1}{1+e^{-w^T \vec{x}_i + b}} \right)$$

avoid b;

$$\begin{array}{c|cccc|c} & x_1 & x_2 & x_3 & \cdots x_d & y \\ \hline \bar{x}_1 & & & & & \\ \bar{x}_2 & & & & & \\ \bar{x}_3 & & & & & \\ \vdots & & & & & \\ \bar{x}_n & & & & & \end{array}$$

$$= \underset{w}{\operatorname{argmax}} \sum_{i=1}^{n} \log \left(\frac{1}{1+e^{-w^T x_i y_i}} \right)$$

$$= \underset{w}{\operatorname{argmax}} -\sum_{i=1}^{n} + \log(1+e^{-w^T \bar{x}_i y_i})$$

$$= \underset{w}{\operatorname{argmin}} \sum_{i=1}^{n} \log(1+e^{-w^T \bar{x}_i y_i})$$

This is a minimisation problem & can be solved using iterative methods. [as it has no closed form sol.]

→ MAP : $\underset{w}{\operatorname{argmax}} \; P(w|D)$

$$\hat{w} = \underset{w}{\operatorname{argmax}} \; P(D|w) \cdot P(w)$$

Assume : $P(w) \sim N(\vec{0}; \sigma^2)$

$$\hat{w} = \underset{w}{\operatorname{argmax}} \; P(y|X; w) \cdot P(w)$$

$$= \underset{w}{\operatorname{argmax}} \prod_{i=1}^{n} P(y_i | \bar{x}_i; w) P(w)$$

$$= \underset{w}{\operatorname{argmin}} \sum_{i=1}^{n} \log(1+e^{-w^T \bar{x}_i y_i}) + \underbrace{\lambda w^T w}$$

min. this gives us simple solution for w. i.e hyperplane near to origin

$$\lambda = \frac{1}{2\sigma^2}$$

Also, this eqn has no closed form solution, so we will use gradient descent.

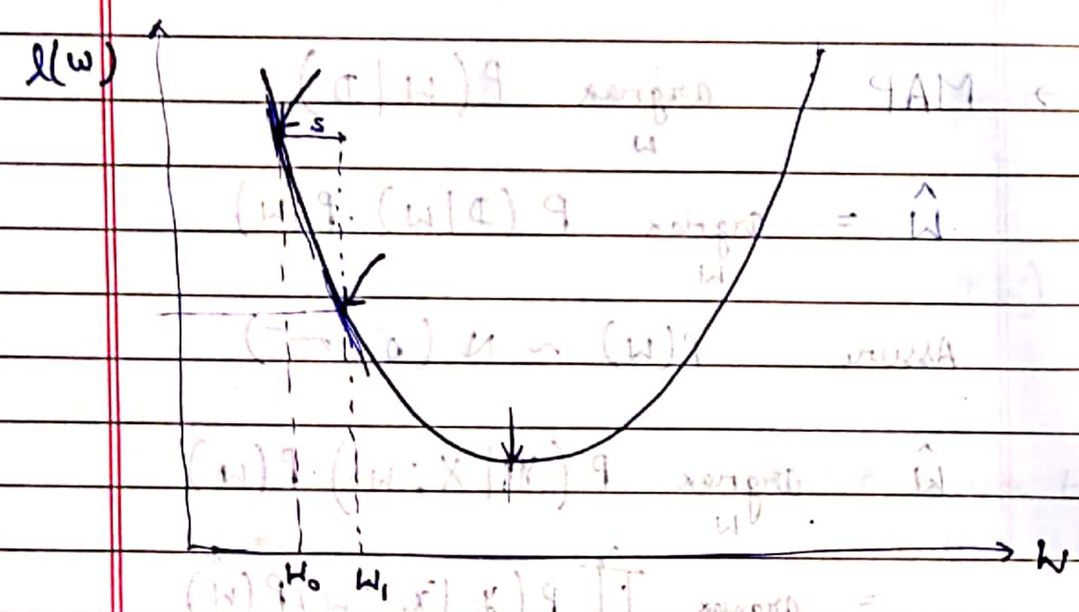
Lec 12: Gradient Descent / Newton's Method

MLE : $\max_{w} \prod_{i=1}^{n} P(y_i | \bar{x}_i)$ . . . | $\underbrace{\min_{w} \sum_{i=1}^{n} \log(1+e^{-w^T \bar{x}_i y_i})}_{l(w)}$

To do : $l(w) = \sum_{i=1}^{n} \log(1+e^{-w^T \bar{x}_i y_i})$

Minimize $l(w)$.

Conditions :- $l(w)$ must be Continous
- differentiable
- Convex.



Taylor Series

$$f(x=a) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \dots$$

eg:

$$f(x) = e^x$$

$$f(x=a) = e^a + \frac{e^a}{1!}(x-a) + \frac{e^a}{2!}(x-a)^2 + \dots$$

Put $a = 0$;

$$f(x) = 1 + x + \frac{x^2}{2!} + \dots$$

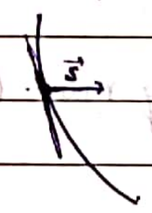
Also ;

$$l(\vec{w}+\vec{s}) \approx l(\vec{w}) + l'(\vec{w}) \cdot \vec{s} + \frac{l''(\vec{w}) \cdot \vec{s}^T \vec{s}}{2!} + \dots$$

$$l'(\vec{w}) = \vec{g}(\vec{w})$$

$$l(\vec{w}+\vec{s}) \approx l(\vec{w}) + g(\vec{w}) \cdot \vec{s} + \dots$$

→ Taylor series approximates the function linearly locally



→ Here it assumes that locally function is parabola.

$$l''(\vec{w}) = H(\vec{w})$$

$$\boxed{H_{ij} = \frac{\partial^2 l}{\partial w_i \partial w_j}}$$

$\vec{s}$ → it has to be a very small step.

- linear approximation can work really good (mostly used)
- Involving $H(\vec{w})$ can reduce the number of steps.

$\vec{s} = -\alpha \vec{g}(w)$

Gradient descent:

$$\vec{s} = -\alpha \vec{g}(w) \qquad (\alpha \to \text{learning rate}) \quad (\alpha > 0)$$

$$l(w + \underbrace{(-\alpha \vec{g}(w))}_{\vec{s}}) \simeq l(w) + \underbrace{(-\alpha g(w))}_{s} g(w)$$

$$\simeq l(w) - \underbrace{\underbrace{\alpha g(w)^T g(w)}_{+ve}}_{-ve} < l(\vec{w})$$

$\vdots$

Stop when $g(w)$ it becomes zero.

Ques : Which is best α, learning rate (Hyper-parameter)

→ ADAGRAD Algorithm

$\vec{s} = 0$

FOR

$$\vec{g} = \frac{\partial l}{\partial w}$$

$$\vec{s} \leftarrow \vec{s} + g \odot g$$

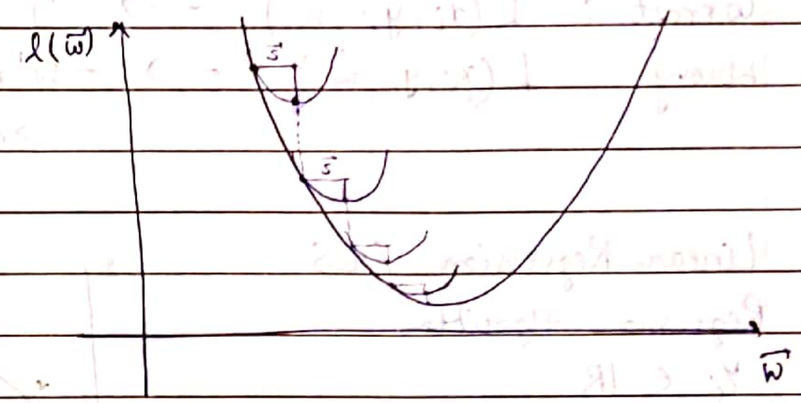
$$w_{t+1} \leftarrow w_t - \frac{\vec{g}}{\sqrt{s+\epsilon}}$$

if $\| w_{t+1} - w_t \| < \delta$

BREAK

→ Consider Second order ; [Newton's method]

$$l(\vec{w} + \vec{s}) \simeq l(\vec{w}) + g(\vec{w}) \cdot \vec{s} + \frac{1}{2} s^T H(\vec{w}) \vec{s}$$



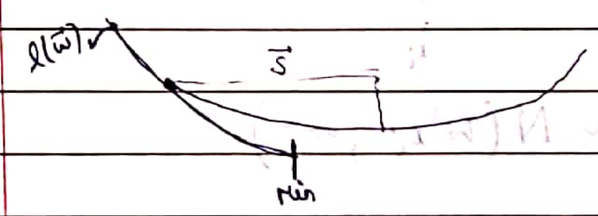
- Newton's method converges super fast.

$$\min_{\vec{s}} \; l(\vec{w} + \vec{s}) \simeq l(\vec{w}) + g(\vec{w}) \cdot \vec{s} + \frac{1}{2} s^T H(\vec{w}) \vec{s}$$

$$g(\vec{w}) + H(\vec{w}) s = 0$$

$$s = H^{-1}(\vec{w}) g(\vec{w})$$

Downside : Most of the time, it doesn't work.

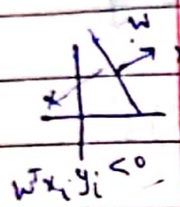


In this case, we computed $g(\vec{w})$ & $H^{-1}(\vec{w})$ which yields $\vec{s}$, which seems very large, eventually violates taylor series.

And you will land on moon (:p)

Lec 13 : Linear / Ridge Regression

$$l(w) = \underbrace{\sum_{i=1}^{n} \log\left(1 + e^{-w^T x_i y_i}\right)}_{L(x_i, y_i, w)}$$

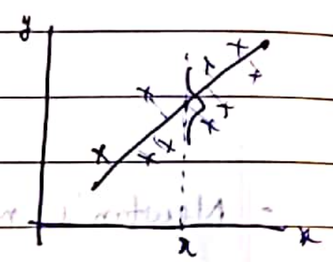


Correct : $L(x_i, y_i, w) \longrightarrow 0$

Wrong : $L(x_i, y_i, w) \longrightarrow \underbrace{-w^T x_i y_i}_{>0}$

Linear Regression OLS

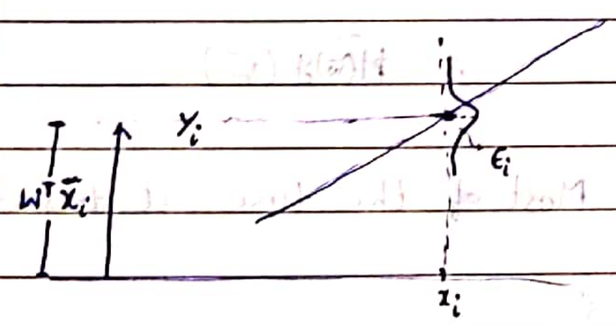
- Regression Algorithm
- $Y_i \in \mathbb{R}$



Assumptions

- Correspondance between X & Y must be linear.
- For each value of x ; y has a Gaussian distribution.

$$Y_i = w^T \vec{x}_i + \epsilon_i \quad ; \quad \epsilon_i \sim N(0, \sigma^2)$$



$$Y_i \sim N(w^T x_i, \sigma^2)$$

In other words, we assume that the data is drawn from a 'line' $w^T x$ through the origin (add I in d+1 dimension).

For each data point with feature x_i, the label y_i is drawn from a Gaussian with mean $w^T x_i$ & var. σ^2.

$$P(x_i | y_i) = \frac{P(y_i | x) P(y_i)}{P(x)}$$

$$P(y_i | \bar{x}_i ; w) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(w^T x_i - y_i)^2}{2\sigma^2}}$$

MLE : $\underset{w}{\arg\max} \prod_{i=1}^{n} P(y_i | \bar{x}_i ; w)$

$$\underset{w}{\arg\max} \sum_{i=1}^{n} \log P(y_i | \bar{x}_i ; w)$$

$$\underset{w}{\arg\max} \sum_{i=1}^{n} \log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) + \left(-\frac{(w^T x_i - y_i)^2}{2\sigma^2}\right)$$

$$\underset{w}{\arg\max} \sum_{i=1}^{n} \log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) - \frac{(w^T x_i - y_i)^2}{2\sigma^2}$$

$$\underset{w}{\arg\min} \sum_{i=1}^{n} \frac{(w^T x_i - y_i)^2}{2\sigma^2}$$

$$\underset{w}{\arg\min} \frac{1}{2\sigma^2} \sum_{i=1}^{n} (w^T x_i - y_i)^2$$

$$\underset{w}{\arg\min} \sum_{i=1}^{n} (w^T x_i - y_i)^2$$

Let's say, you get loss of 7.5 which is meaningless. So, we divide by no. of observation.

$$\underset{w}{\arg\min} \frac{1}{n} \sum_{i=1}^{n} (w^T x_i - y_i)^2$$

$$l(w) = \frac{1}{n} \sum_{i=1}^{n} (w^T x_i - y_i)^2$$

MAP : $P(w \mid y_i, x_i, \ldots x_n) = \dfrac{P(D \mid w)\, P(w)}{Z}$

$w \sim N(\vec{0}, \tau^2)$

$\underset{w}{\operatorname{argmax}}\ P(w \mid y_i, x_i, \ldots x_n)$

$\underset{w}{\operatorname{argmax}}\ P(y_i, x_i, \ldots x_n \mid w) \cdot P(w)$

$\underset{w}{\operatorname{argmax}} \prod_{i=1}^{n} P(y_i \mid x_i; w)\ P(w)$

$\underset{w}{\operatorname{argmax}} \sum_{i=1}^{n} \log\left(P(y_i \mid x_i; w)\right) + \log\left(P(w)\right)$

$P(x) = \frac{1}{\sqrt{2\pi\tau^2}} e^{-\frac{x^2}{2\tau^2}}$

$\underset{w}{\operatorname{argmin}}\ \frac{1}{2\sigma^2} \sum_{i=1}^{n} (w^T x_i - y_i)^2 + \frac{1}{2\tau^2} w^T w$

$\underset{w}{\operatorname{argmin}}\ \frac{1}{2\sigma^2} \sum_{i=1}^{n} (w^T x_i - y_i)^2 + \frac{n}{2\tau^2} w^T w$

$\underset{w}{\operatorname{argmin}}\ \frac{1}{n} \sum_{i=1}^{n} (w^T x_i - y_i)^2 + \lambda \|w\|_2^2$

$\boxed{\lambda = \frac{\sigma^2}{n\tau^2}}$

$\boxed{L(w) = \frac{1}{n} \sum_{i=1}^{n} (w^T x_i - y_i)^2 + \lambda \|w\|^2}$

OLS	Ridge Regression
- Squared loss	- Squared loss
- No regularization	- l2 - regularization

Lec14 : (Linear) Support Vector Machines

$L(w) = \frac{1}{n} \sum_{i=1}^{n} (w^T x_i - y_i)^2 + \lambda \|w\|_2^2$

$X^T = [x_1 \ \ldots \ x_n]$

$Y^T = [y_1 \ \ldots \ y_n]$

$L(w) = (Xw - Y)^2 = (Xw - Y)^T (Xw - Y)$

$= w^T X^T X w - 2 Y^T X w + Y^T Y$

$L'(w) = 2 X^T X w - 2 X^T Y$

$L'(w) = 0$

$w = (X^T X)^{-1} X^T Y$ → Closed form solution

$O(d^3)$

$S_{T\,O(d^2)}$

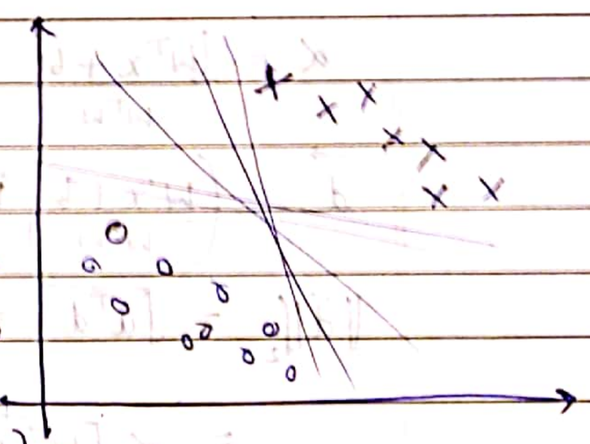
Downside : - $X^T X$ can be gigantic & storing it in memory is not possible.

- Computing $(X^T X)^{-1}$ can take a life

That's why we do gradient descent.

Support Vector Machines (1994)

$y_i \in \{-1, 1\}$

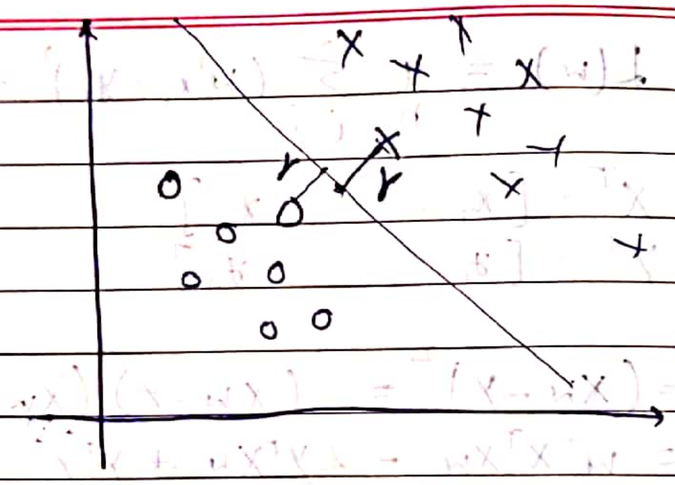


If data is linearly separable then we have many hyperplane which separates the classes. (perceptron did this for us).

But, we donot know which one to find. Hence, perceptron becomes weak in such scenario.

Scanned with CamScanner

Cannot admit the usage of SVM (patent)

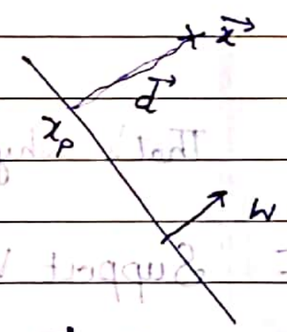


Intuition : SVM finds the best hyperplane which has the maximum margin.

Margin :

$$\mathcal{H}_{w,b} = \{x : w^T x + b = 0\}$$

$$\vec{x}_p = \vec{x} - \vec{d}$$



$$w^T \vec{x}_p + b = 0$$

$$w^T(\vec{x} - \vec{d}) + b = 0$$

$$w^T(\vec{x} - \alpha \vec{w}) + b = 0$$

$$\alpha = \frac{|w^T x + b|}{w^T w}$$

$\vec{d} \parallel \vec{w}$

$\vec{d} = \alpha \vec{w}$

$$\vec{d} = \left(\frac{w^T x + b}{w^T w}\right) \vec{w}$$

$$\|\vec{d}\|_2 = \sqrt{d^T d} = \sqrt{\alpha w^T \cdot \alpha w} = \sqrt{\alpha^2 w^T w}$$

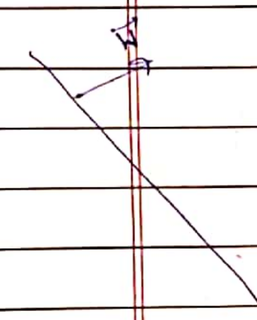
$$= \alpha \sqrt{w^T w}$$

$$= \frac{|w^T x + b|}{w^T w} \cdot \sqrt{w^T w}$$

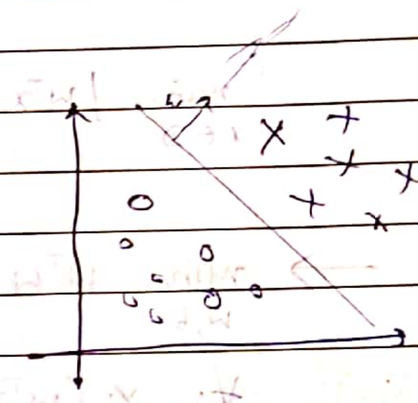
$$\|d\|_2 = \frac{|w^T x + b|}{\|w\|_2}$$

$$\gamma(w, b) = \min_{x \in D} \frac{w^T x + b}{\|w\|_2}$$

Solve : $\max_{w,b} \gamma(w, b)$



It maximises the margin from both classes.



Constraints :

① $\forall_i \quad y_i (w^T x_i + b) \geq 0$

Problem : $\max_{w,b} \left[\min_{x \in D} \frac{|w^T x + b|}{w^T w} \right]$

$\forall_i \quad y_i (w^T x_i + b) \geq 0$

$$> \max_{w,b} \frac{1}{w^T w} \left[\min_{x \in D} |w^T x + b| \right]$$

$$\mathcal{H} = \{x : w^T x + b = 0\}$$

Re-scales w & b s.t.

② $$\min_{x \in D} |w^T x + b| = 1$$

$$\max_{w,b} \frac{1}{w^T w}$$

$$\min_{w,b} w^T w$$

$\rightarrow \min_{W,b} W^T W$

$\forall_i \quad Y_i (W^T x_i + b) \geq 0$

$\min_{x \in D} |W^T x + b| = 1$

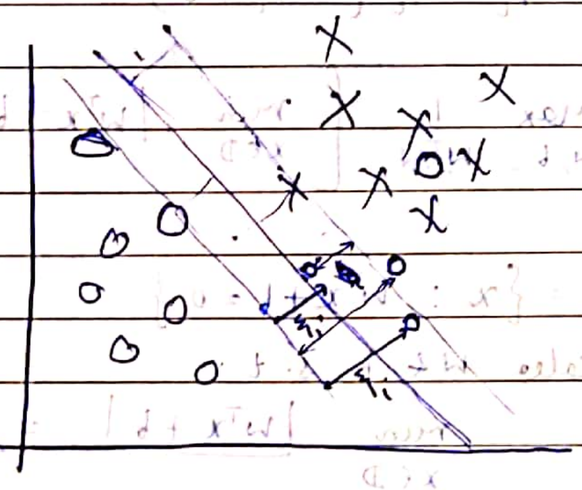
$\Leftrightarrow \forall_i \; Y_i (W^T x_i + b) \geq 1$

$\rightarrow \min_{W,b} W^T W$

$\forall_i \quad Y_i (W^T x_i + b) \geq 1$

This problem is quadratic optimisation problem. The objective is quadratic & the constraints are linear.

It can be solved using QCQP (Quadratically Constrained quadratic program) solver.



Above problem has no feasible solution, because data is not linearly separable. This mainly arises in case of low dimensional data.

$C \rightarrow$ Hyper-parameter

SVM with Soft Constraints

$$\min_{W,b} W^T W + C \sum_{i=1}^{n} \xi_i$$

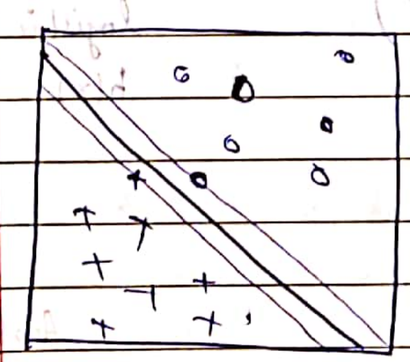
$$\text{s.t.} \quad \forall_i \quad Y_i (W^T x_i + b) \geq 1 - \xi_i$$

$$\forall_i \quad \xi_i \geq 0$$

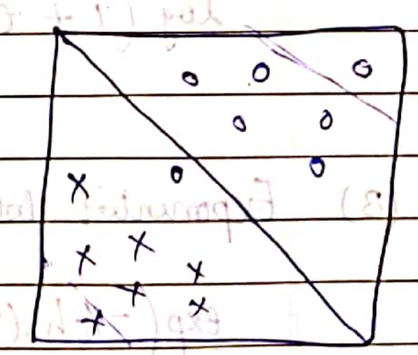
- The slack variable ξ_i allows the input x_i to be closer to the hyperplane.
- But there is penalty in the objective function.
- If C is very large, the SVM becomes very strict & tries to get all points correctly classified.
- If C is very small, the SVM becomes very loose & may sacrifice some points to obtain a simpler method.

$$\xi_i = \begin{cases} 1 - Y_i (W^T x_i + b) & ; Y_i (W^T x_i + b) < 1 \\ 0 & ; Y_i (W^T x_i + b) \geq 1 \end{cases}$$

$$\rightarrow \min_{W,b} \underbrace{W^T W}_{\text{L2-Regularisation}} + \underbrace{C \sum_{i=1}^{n} \max(1 - Y_i (W^T x_i + b), 0)}_{\text{Loss function}}$$



C = 100



C = 0.01

$$\min_{w,b} \underbrace{w^T w}_{\text{regulariser}} + C \underbrace{\sum \max\left(1 - y_i(w^T x_i + b), 0\right)^2}_{l(w)}$$

- If $\|w\|_2$ is small, simple solution (in SVM large margin)

$$\rightarrow \min_{w} \frac{1}{n} \sum_{i=1} l(h_i(x_i), y_i) + \lambda R(w)$$

$\rightarrow$ Empirical Risk minimisation

Loss function

Classification losses :

(I) Hinge Loss :

$$\max\left[1 - h_w(x) y_i, 0\right]^p$$

In SVM, $h(x) = w^T x + b$

$p = 1$; SVM

$p = 2$; Sq. loss SVM

(2) log-loss

$$\log\left(1 + e^{-y_i h_w(x_i)}\right)$$

logistic regression
$h(x) = w^T x$

(3) Exponential loss :

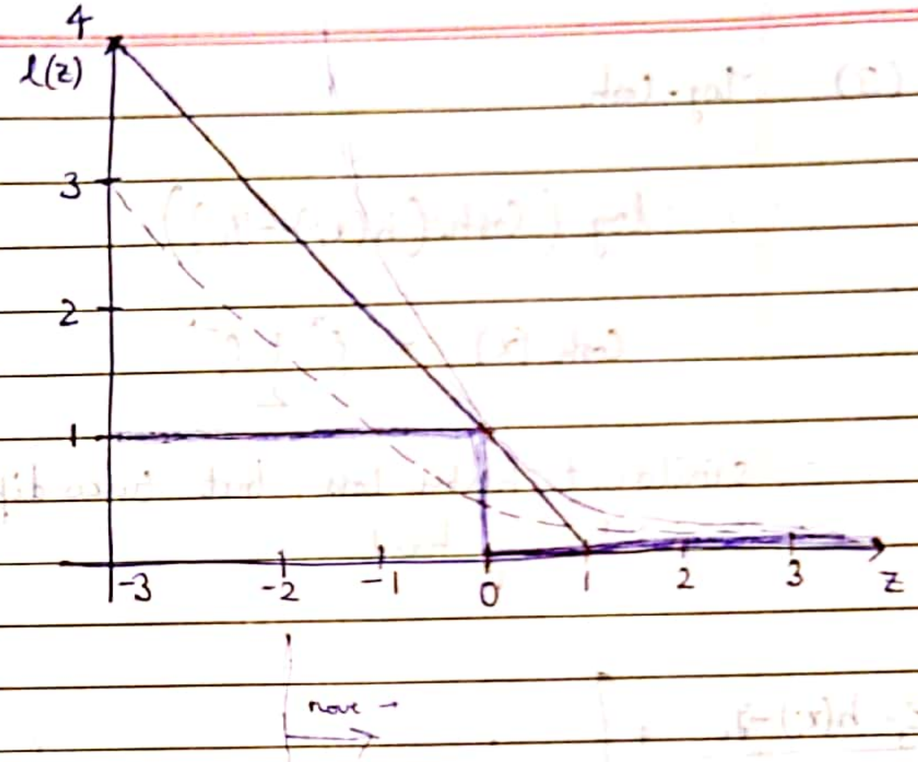
$$\exp\left(-y_i h_w(x_i)\right)$$

Adaboost

$z > 0$; Correct
$z < 0$; Incorrect
$(w^T x + b \gtrless 0)$

$z = h_w(x) y_i$

$l(z)$



Regression Losses :

$y_i \in \mathbb{R}$

(I) Squared loss

$$l_2 = \left(h_w(x_i) - y_i\right)^2$$

(2) Huber loss

$$l(w) = \begin{cases} \frac{1}{2}\left(h(x_i) - y_i\right)^2 & ; |h(x_i) - y_i| < \delta \\ \delta\left(|h(x_i) - y_i| - \frac{\delta}{2}\right) & ; \text{otherwise} \end{cases}$$

- "Best of both worlds" of Squared-loss & Absolute loss
- Also known as 'smooth Absolute loss'.
- once differentiable

Scanned with CamScanner

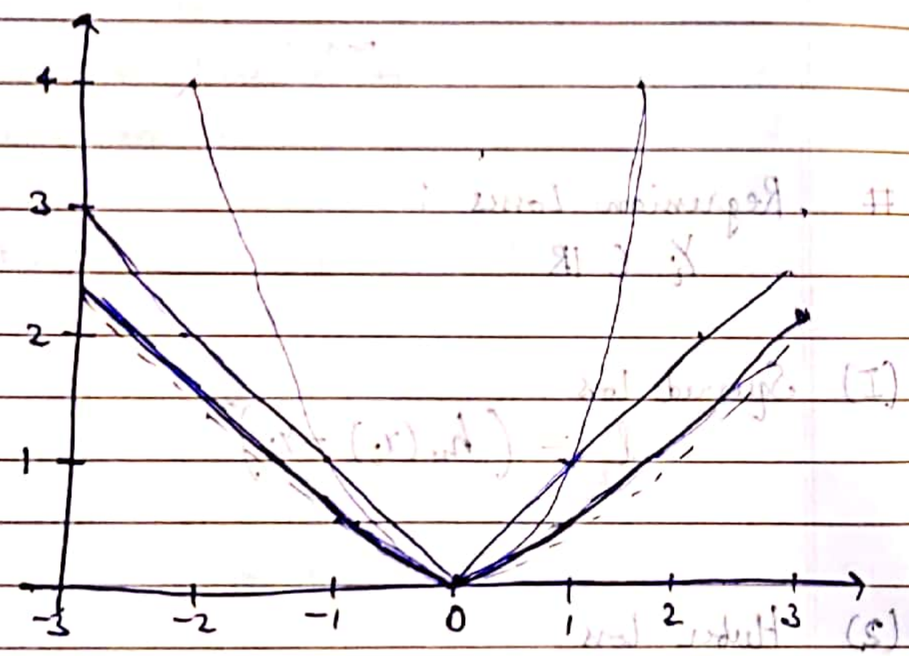
(3) log-cosh

$$\log\left(\cosh\left(h(x_i) - y_i\right)\right)$$

$$\cosh(x) = \frac{e^{x} + e^{-x}}{2}$$

- Similar to huber loss, but twice differentiable.
- Computationally hard.

$z_i = h(x_i) - y_i$



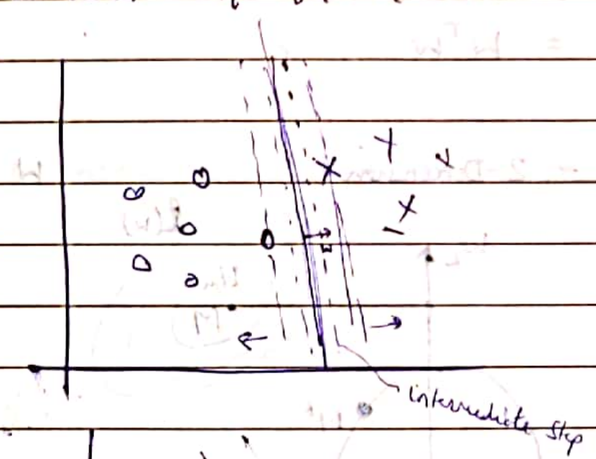
— Absolute loss
Squared loss
--- log cosh
Huber loss ($\delta = 1$)

Regularizers

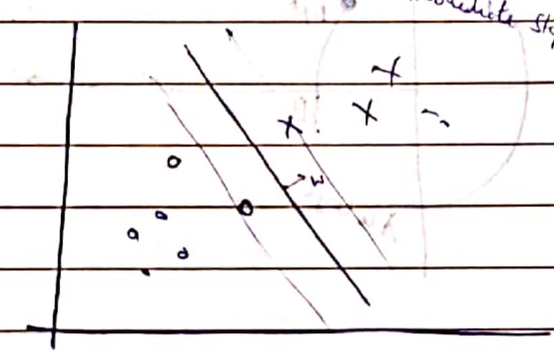
$$\min_{w} \frac{1}{n} \sum_{i=1}^{n} \ell\left(h_w(x_i), y_i\right) + \lambda r(w)$$

$r(w) = w^T w$
- SVM
- OLS using MAP
- logistic regression MAP

$\min w^T w \quad \forall_i \; y_i(w^T x_i + b) \geq 1$



When we make our w small then margin has to go outwards ($\because$ $w \cdot x \geq 1$)
small large



The moment you hit data pt. you have to stop.

- Objective function says, get me the smallest possible w & also the largest margin.
- Regularizer in SVM says find the hyperplane with max. margin.
- In general, it says I want simpler solution.

* Optimisation theory

- Re-write the problem as :

Lagrangian formulation

$$\min_{W} \frac{1}{n} \sum_{i=1}^{n} l(h_W(x_i), y_i) + \lambda r(W)$$

s.t. $r(W) \leq B$

If λ is small ; ignoring regulariser

If λ is large ; minimising $r(W)$ & after solving we get $\min(r(W)) = B$

L2 - Regulariser

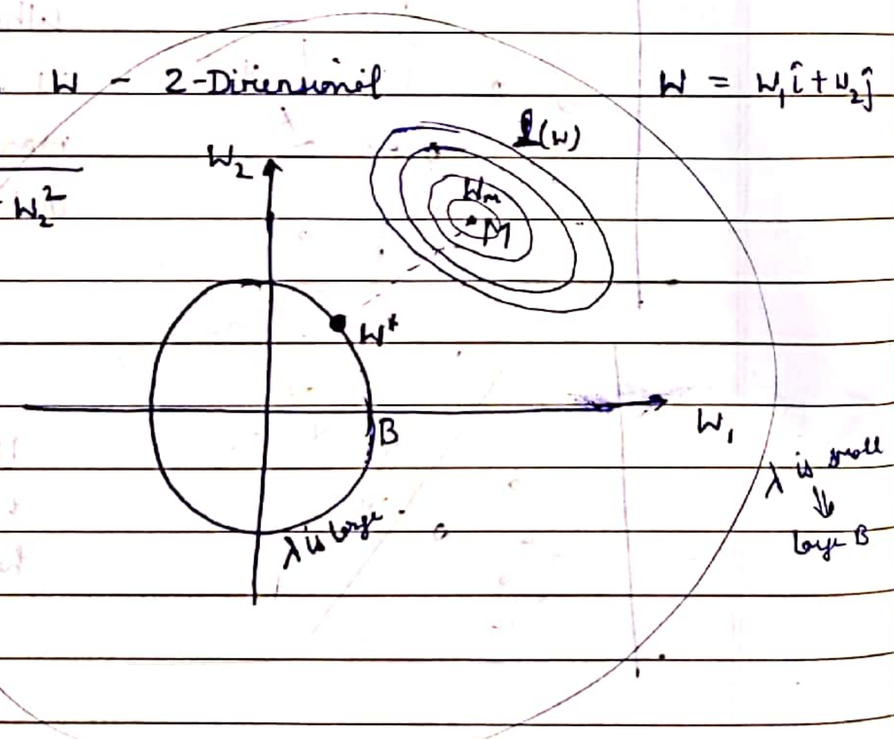
$$r(W) = W^T W$$

Assume W - 2-Dimensional $\qquad W = W_1\hat{i} + W_2\hat{j}$

$$W^T W = \sqrt{W_1^2 + W_2^2}$$

$$W^T W \leq B$$

$$W_1^2 + W_2^2 \leq B^2$$



- Minimising loss function, we get M point (W_m) but regulariser here is a constraint which says, W has to be on the circle of radius B.
- Hence, W^* is the optimal value of W.

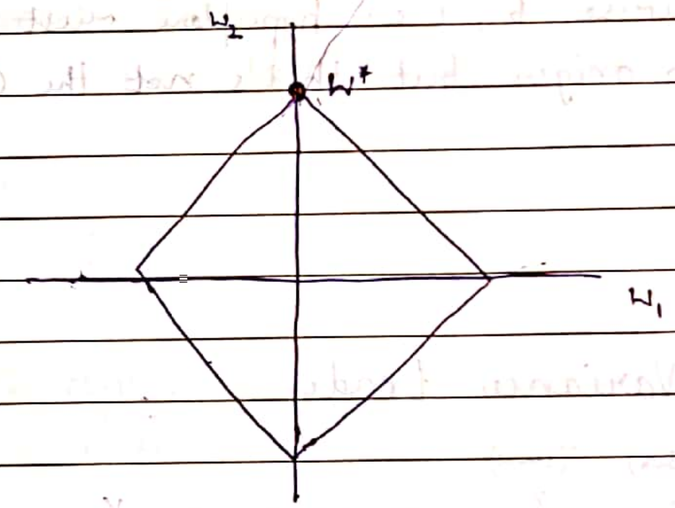
L1 - regulariser

$$|W| = \sum_{i=1}^{d} |W_i|$$

$$r(W) = \lambda |W|$$

$|W| \leq B$ — Simplicity

$W^* = W_2\hat{j}$



- Advantage : helps in ruling out features i.e. In above case ; $W_1 = 0$ which says feature in $\hat{i}$ direction doesn't contribute in prediction.
- Sparse W^* matrix.
- Downside : Not convex

→ Loss and REGULARIZER

Elastic Net

$$\lambda |W| + \mu \|W\|_2$$

$$\min_{W} \frac{1}{n} \sum_{i=1}^{n} (W^T x_i - y_i)^2 + \lambda |W| + \mu \|W\|_2^2$$

$\lambda, \mu \in [0, 1)$

Lasso

$$\min_{W} \frac{1}{n} \sum_{i=1}^{n} (W^T x_i - y_i)^2 + \lambda |W|$$

REVIEW

Ques: Why shouldn't you incorporate the bias as a const. feature when working with SVM?

> Because we are not minimising b; as we minimise b, i.e. hyperplane must be close to origin but that's not the case.

Lec 19:

Bias - Variance trade

	(Rooms) X_1	(Bath) X_2	(Lawn) X_3	X_4	Y
$\vec{x}_1$	4	3	1	2	y_1
$\vec{x}_2$					y_2
$\vec{x}_3$					y_3
⋮					⋮
$\vec{x}_n$					y_n

→ $D = \{(\vec{x}_1, y_1) \ldots (\vec{x}_n, y_n)\}$, drawn i.i.d from some distribution $P(X,Y)$
(Rooms, bath, ... identical & independent)

$y \in \mathbb{R}$, $x \in \mathbb{R}^d$

→ for any input $\vec{x}$, there might not exist a unique label y.

$\mathcal{A}$ → SVM, KNN, NB, etc

- You could imagine two houses with identical descriptions selling at different prices. So, for given input $\vec{x}$, there is a distribution over possible labels.

Expected Label

$$\bar{y}(\vec{x}) = E_{y|x}[Y] = \int_y y \cdot P(y|x)\, dy$$

$$h_D = \mathcal{A}(D)$$

The expected label denotes the label you would expect to obtain, given a $\vec{x}$.

- For a given h_D, learned on Data D with Algorithm $\mathcal{A}$, we can compute the generalisation error.

Expected test error (given h_D)

$$E_{\vec{x}, y \sim P}\left[(h_D(x) - y)^2\right] = \int_x \int_y [h_D(x) - y]^2 P(x,y)\, dx\, dy$$

Expected Classifier (given $\mathcal{A}$)

X, Y are r.v.; X, Y made dataset D which is also a r.v; $\mathcal{A}$ is a function of r.v. D; so $\mathcal{A}$ is also a r.v, hence h_D is also a random variable.

$$\bar{h} = E_{D \sim P^n}[\mathcal{A}(D)] = \int_D h_D P^n(D)\, dD$$

↳ f^n

Expected test Error (given $\mathcal{A}$)

$$E_{\substack{(x,y)\sim P \\ D\sim P^n}}\left[(h_D(x)-y)^2\right] = \int_D\int_x\int_y (h_D(x)-y)^2 P(x,y)\,P(D)\,dx\,dy\,dD$$

- draw D from P^n distribution
- feed it to algorithm $\mathcal{A}$
- Compute h_D
- draw a test point $(x,y) \sim P$
- Compute error.

} repeat this process multiple no. of times.

D → training set

(x,y) → test points.

- This is what we are interested in.

Decomposition of Expected test Error:

$$E_{x,y,D}\left[(h_D(x)-y)^2\right] = E_{x,y,D}\left[\left[(h_D(x)-\bar{h}(x)) + (\bar{h}(x)-y)\right]^2\right]$$

$$= E_{x,y,D}\left[(h_D(x)-\bar{h}(x))^2\right] + E_{x,y,D}\left[(\bar{h}(x)-y)^2\right]$$

$$+ 2E_{x,y,D}\left[\underbrace{(h_D(x)-\bar{h}(x))}_{0}(\bar{h}(x)-y)\right]$$

$$\begin{cases} E_{x,y}\left[E_D\left[(h_D(x)-\bar{h}(x))(\underbrace{\bar{h}(x)-y}_{k})\right]\right] \\ E_{x,y}\left[E_D\left[(h_D(x)-\bar{h}(x)\right](\bar{h}(x)-y)\right] \\ E_{x,y}\left[\underbrace{(E_D[h_D(x)]-\bar{h}(x))}_{0}\cdot(\bar{h}(x)-y)\right] \end{cases}$$

$E_{x,y}[0] = 0$

$$- E_{x,y,D}\left[(\bar{h}(x)-y)^2\right] = E_{x,y}\left[\left[(\bar{h}(x)-\bar{y}(x)) + (\bar{y}(x)-y)\right]^2\right]$$

$$= E_{x,y}\left[(\bar{h}(x)-\bar{y}(x))^2\right] + E_{x,y}\left[(\bar{y}(x)-y)^2\right]$$

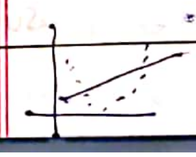
$$+ 2E_{x,y}\left[(\bar{h}(x)-\bar{y}(x))(\bar{y}(x)-y)\right]$$

$$\begin{cases} E_x\left[E_{y|x}\left[(\bar{y}(x)-y)(\bar{h}(x)-\bar{y}(x))\right]\right] \\ E_x\left[\underbrace{(\bar{y}(x)-E_{y|x}[y])}_{0}\cdot(\bar{h}(x)-\bar{y}(x))\right] \end{cases}$$

$$\longrightarrow = \underbrace{E_x\left[(\bar{h}(x)-\bar{y}(x))^2\right]}_{\text{BIAS}^2} + \underbrace{E_{x,y}\left[(\overset{\text{expected } y}{\bar{y}(x)}-\overset{\text{actual } y}{y})^2\right]}_{\text{NOISE}} + \underbrace{E_{x,D}\left[(h_D(x)-\bar{h}(x))^2\right]}_{\text{VARIANCE}}$$

Variance: It captures how much your classifier changes if you train on a different training set.
How 'over-specialized' is your classifier to a particular training set (overfitting)?

Bias: What is the inherent error that you obtain from your classifier even with infinite training data? This is due to your classifier being 'biased' to a particular kind of solution (eg. linear classifier).
Bias is inherent to your model.

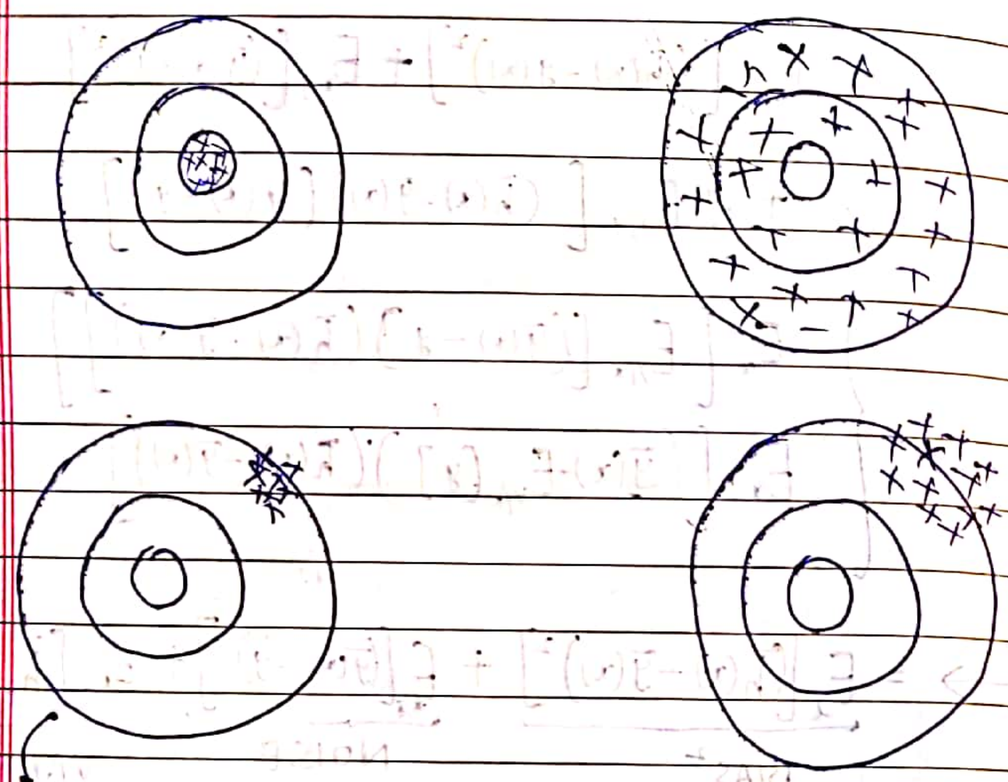


Variance of Bayes optimal classifier is 0
(∵ it separates distributions not Data)
Bias is also 0. (∵ $h(x) = \bar{y}(x)$)

Low Variance — High Variance

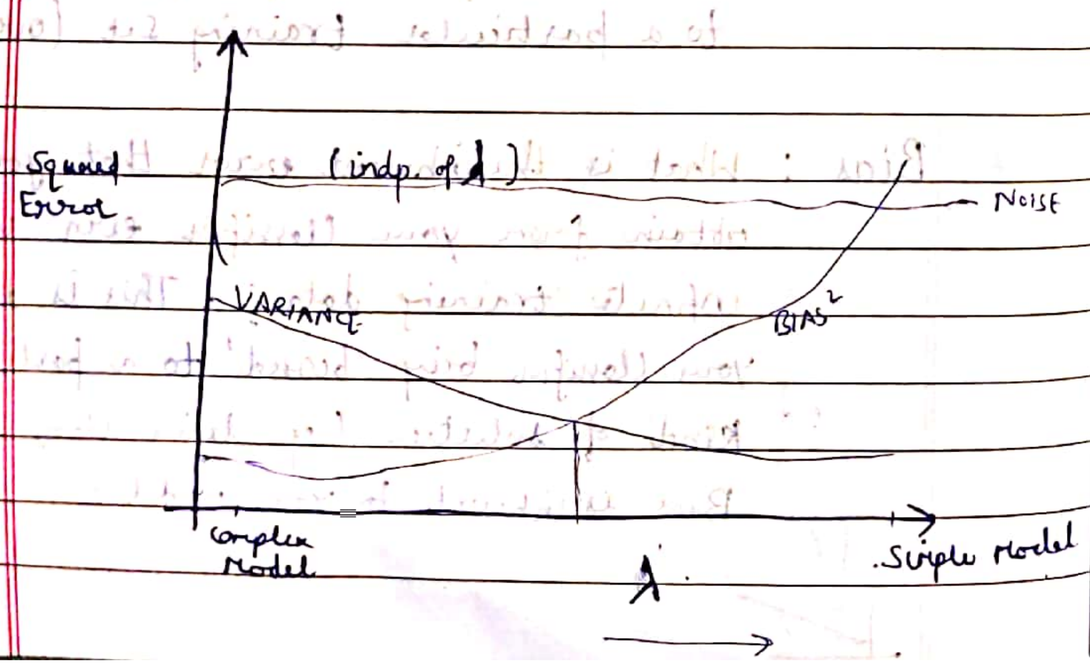
Low Bias

High Bias



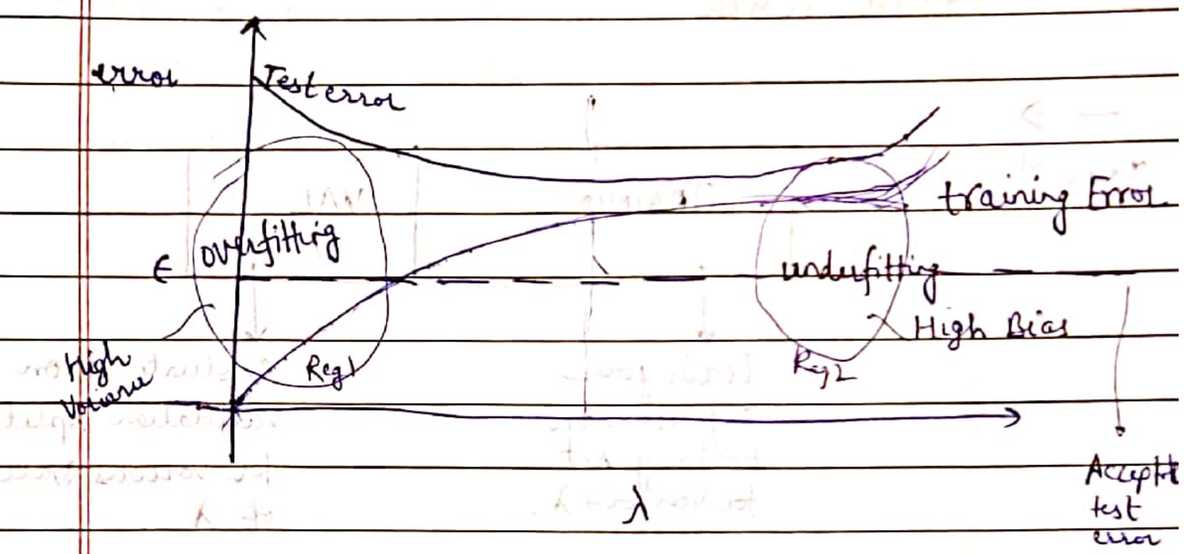
Variance is low, but
g'n biased towards diffent point.

→ NOISE can be reduced by :
- Change the distribution by filtering out outliers. i.e. data Cleaning.
- Add more features.



λ – Hyper parameter

$$\min_{w} \sum_{(x,y)\in D} l(x,y;w) + \lambda r(w)$$



Region 1 (High Variance)

– Symptoms
- Training error is much lower than test error
- Training error is lower than ε
- Test error is above ε.

– Remedies
- Add more training data
- Increase λ ; reduce Complexity
- Bagging.

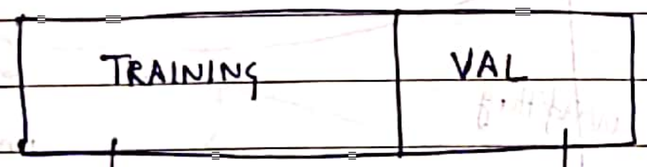
Region 2 (High Bias)
- Training error is higher than ε

Remedies
- Use more Complex models
- Add features
- Boosting

Identify the Sweet spot.

- K-fold Cross Validation
- grid Search

→ Trivial



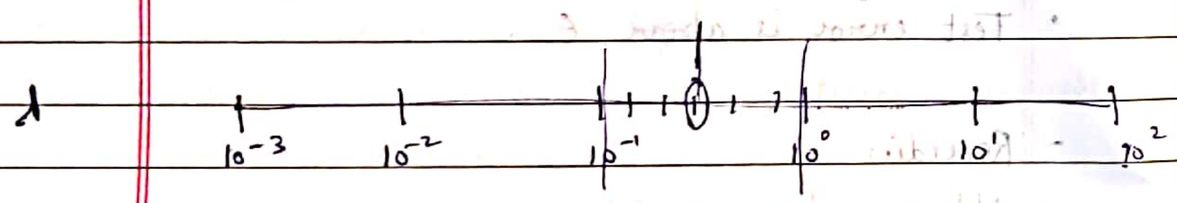
Train your algorithm on training set for various λ.

evaluate it on Validation split, for various value of λ

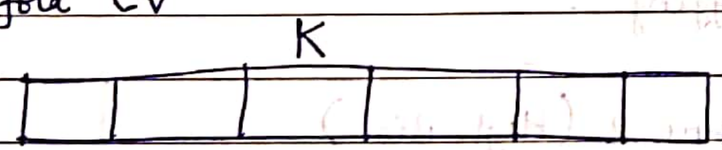
- Train the algorithm on Tr. with λ
- evaluate it on Val. set

} Repeat for diff. λ.

Telescopic approach.



→ K-fold CV



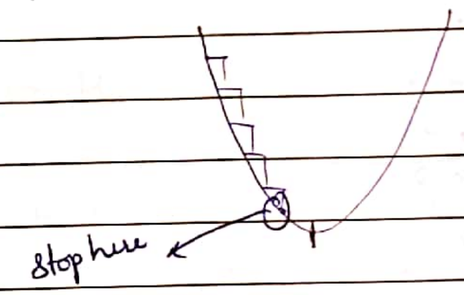
(K-1) → train

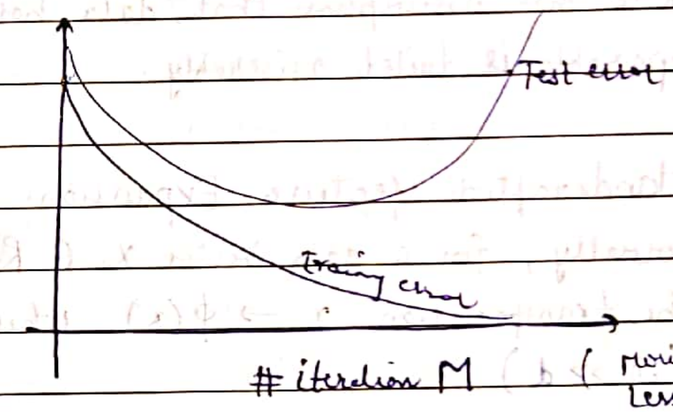
1 → Val

Do this iteration K times with diff Val set.

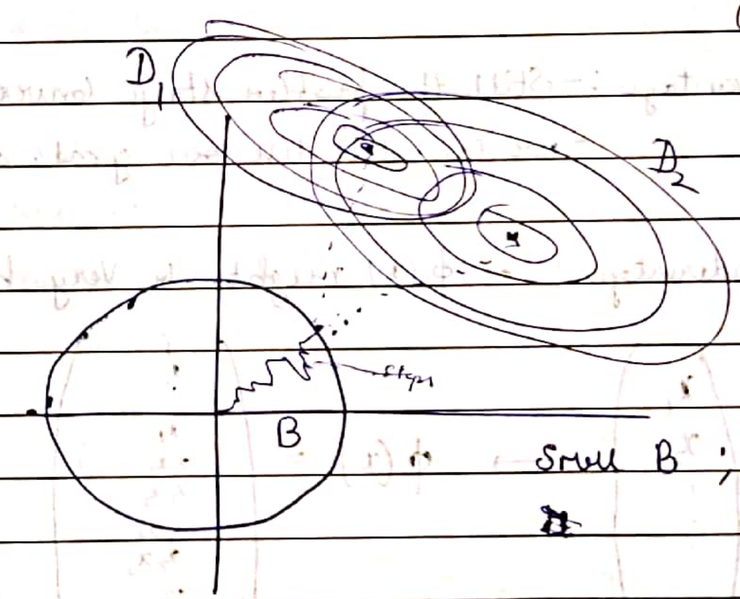
→ Another way of reducing overfitting is EARLY STOPPING

- part of an optimisation process, when we find W in correspondence to lower loss function after taking M steps, we reach to minima.





(More iteration Means Less regularisation (λ))

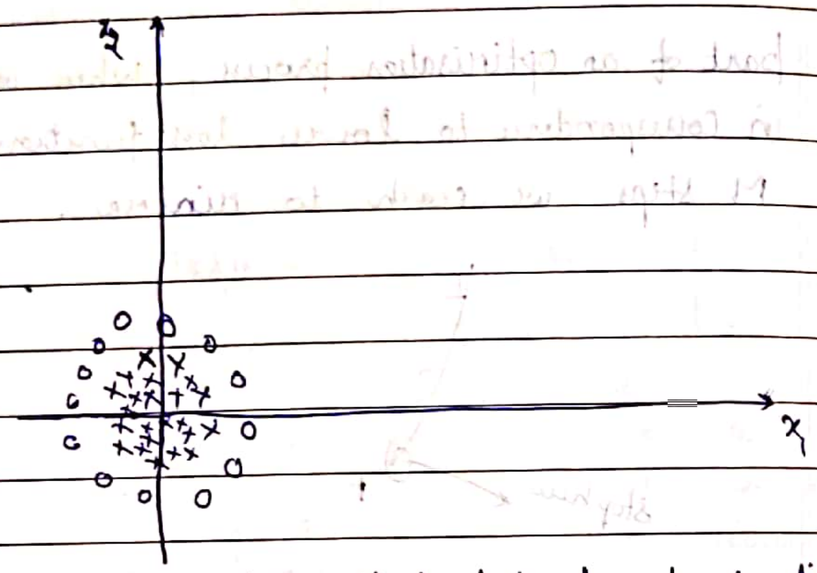


Small B ; large λ

When no. of steps are low in grad. descent, Variance of classifier is also low.

Lec 22 : Kernels

Kernels



- Here, our assumption that, data has to be linearly separable is failed miserably.

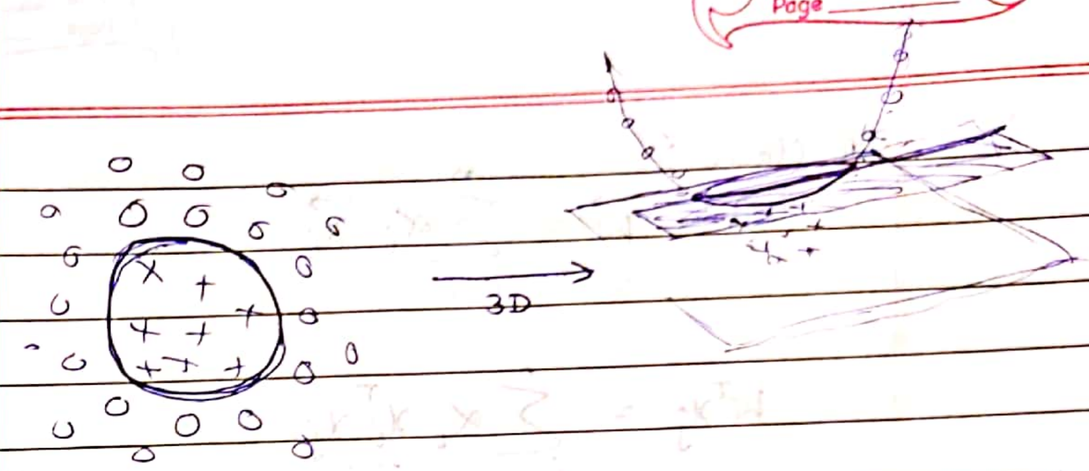
Handcrafted feature Expansion

- Formally, for a data vector $x \in R^d$, we apply the transformation $x \to \phi(x)$ where $\phi(x) \in R^D$ $(D >> d)$

Advantage :- Still the problem stays convex & well behaved.
- we can still use our grad. descent code.

Disadvantage : - $\phi(x)$ might be very high dimensional.

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix} \longrightarrow \phi(x) = \begin{pmatrix} x_1 \\ \vdots \\ x_d \\ x_1x_2 \\ x_1x_3 \\ \vdots \\ x_1x_d \\ x_2x_3 \\ \vdots \\ x_2x_d \\ \vdots \\ x_1x_2x_3 \\ \vdots \\ x_1x_2x_3 \ldots x_d \end{pmatrix}$$



- The problem is non linear, but we want to use a linear model (say SVM) which again says that bias is very high (as we are biased towards linear model).
- Adding more feature, like finding all non-linearity attribution can lead us to find a linear plane & hence decreases bias.

--- $x \in \mathbb{R}^d$, $\phi(x) \in \mathbb{R}^{2^d}$

If $d \to \infty$; $2^d \to \infty$, No way possible to write down the $\phi(x)$ vector.

If d is low, then it works perfectly.

Kernel Trick

- Express the linear classifier in terms of Inner product.

$$K_{ij} = x_i^T x_j \quad \text{(precompute this K matrix)}$$

$$l(w) = \sum_{i=1}^{n} (w^T x_i - y_i)^2$$

$$g = \frac{\partial l}{\partial w} = \sum_{i=1}^{n} \underbrace{2(w^T x_i - y_i)}_{\gamma_i} \cdot x_i = \sum_{i=1}^{n} \gamma_i x_i$$

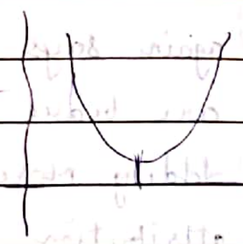
We claim ;

$$\vec{W} = \sum_{i=1}^{n} \alpha_i \cdot \vec{x_i}$$

$$W^T x_j = \sum_{i=1}^{n} \alpha_i \underbrace{x_i^T x_j}_{K_{ij}}$$

→ proof by Induction :

$W_0 = \vec{0} \quad \alpha_i = 0 \quad \forall_i$ ✓



iter:t ; $W = \sum_{i=1}^{n} \alpha_i \vec{x_i}$

$$W \leftarrow W - s\vec{g} = \sum_{i=1}^{n} \alpha_i \vec{x_i} - s\sum_{i=1}^{n} \gamma_i \vec{x_i}$$

$$= \sum_{i=1}^{n} \underbrace{(\alpha_i - s\gamma_i)}_{\text{new alpha}} x_i$$

New gradient descent :

$\alpha_i = 0 \quad \forall_i$

Compute γ_i

$\alpha_i \leftarrow \alpha_i - s\gamma_i$

- Initially, it was hard to Compute W for high dimensional data, but now we can Compute n α's as they are independent of the dimensions of the data.

Consequently, we can write $l(W)$ in terms of $l(\alpha)$

$$l(\alpha) = \sum_{i=1}^{n} \left(\sum_{j=1}^{n} \alpha_j \underbrace{x_j^T x_i}_{K(x_i, x_j)} - y_i \right)^2$$

During test-time ;

$$h(x_t) = W^T x_t = \sum_{j=1}^{n} \alpha_j x_j^T x_t$$

Inner-product Computation

$$x \rightarrow \phi(x) = \begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_d \\ x_1 x_2 \\ \vdots \\ x_{d-1} x_d \\ \vdots \\ x_1 x_2 \cdots x_d \end{pmatrix} \qquad \phi(z) = \begin{pmatrix} 1 \\ z_1 \\ \vdots \\ z_d \\ z_1 z_2 \\ \vdots \\ z_{d-1} z_d \\ \vdots \\ z_1 z_2 \cdots z_d \end{pmatrix}$$

$$K(x,z) = \phi(x)^T \phi(z) = 1 \cdot 1 + x_1 z_1 + x_2 z_2 + \cdots x_1 x_2 z_1 z_2 + \cdots (x_1 \cdots x_d)(z_1 \cdots z_d) \quad \Big\} \; O(2^d)$$

$$= \prod_{k=1}^{d} (1 + x_k z_k) \quad \Big\} \; O(d)$$

$$K(x_i, x_j) = K_{ij} = \phi(x_i)^T \phi(x_j) = \prod_{k=1}^{d} (1 + x_{ik} x_{jk})$$

This matrix has to be ...

General Kernels

(I) Linear

$$K(x,z) = x^T z$$

- The linear kernel is equivalent to just using a good old linear classifier.
- But it can be faster to use a kernel matrix if the dimensionality d of the data is high.

(II) Polynomial

$$K(x,z) = (1 + x^T z)^p$$

(III) Radial Basis function (RBF) (aka Gaussian Kernel)

$$K(x,z) = e^{-\frac{|x-z|^2}{\sigma^2}}$$

(IV) Exponential Kernel

$$K(x,z) = e^{-\frac{|x-z|}{2\sigma^2}}$$

(V) Laplacian Kernel

$$K(x,z) = e^{-\frac{|x-z|}{\sigma}}$$

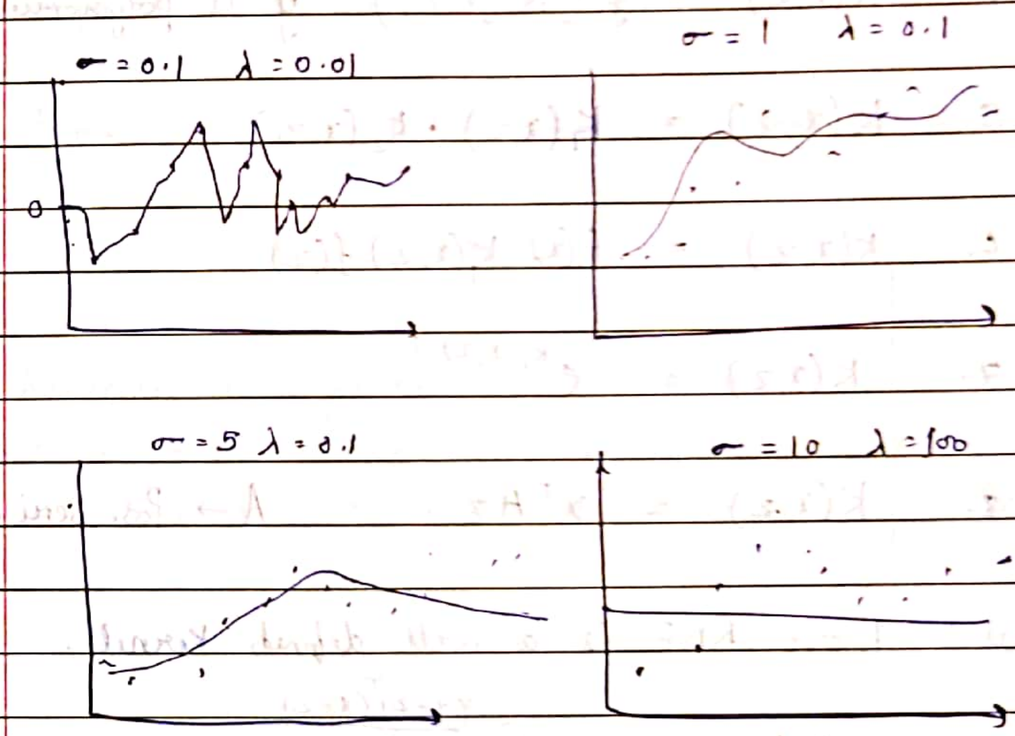
(VI) Sigmoid Kernel

$$K(x,z) = \tanh(ax^T + c)$$

Ques: Can any function $K(\cdot,\cdot) \to R$ be used as Kernel?

No, This is the case if & only if K is positive semi-definite.

A matrix $A \in R^{n \times n}$ is positive semi-definite if $\forall q \in R^n$, $q^T A q \geq 0$



Get the Right hyper-parameter & train accordingly.

Well defined Kernels

1. $K(x,z) = x^T z$

2. $K(x,z) = c K_1(x,z)$

3. $K(x,z) = K_1(x,z) + K_2(x,z)$

4. $K(x,z) = g(K_1(x,z))$ g is polynomial +ve coff.

5. $K(x,z) = K_1(x,z) \cdot K_2(x,z)$

6. $K(x,z) = f(x) K_1(x,z) f(z)$

7. $K(x,z) = e^{K_1(x,z)}$

8. $K(x,z) = x^T A z$ $A \rightarrow$ Pos. semi definite Matrix

Ques: Prove RBF is a well-defined Kernel.

$$K(x,z) = e^{-\frac{(x-z)^T(x-z)}{\sigma^2}}$$

Rule 1: $x^T z$

Rule 2: $\frac{2}{\sigma^2} x^T z$

Rule 7: $e^{\frac{2}{\sigma^2} x^T z}$

$f(x) = e^{-\frac{x^T x}{\sigma^2}}$

Rule 6: $f(x)\, e^{\frac{2}{\sigma^2} x^T z}\, f(z)$

$f(z) = e^{-\frac{z^T z}{\sigma^2}}$

Kernel on Sets:

$S_1, S_2 \subseteq \Omega$

$$K(S_1, S_2) = e^{|S_1 \cap S_2|}$$

Kernel Machines

In practice, an algorithm can be Kernelized in 2 steps:

1. Prove that the solution lies in the span of the training points

i.e. $w = \sum_{i=1}^{n} \alpha_i x_i$

2. Rewrite the algorithm & the classifier so that all training or testing inputs x_i are only accessed in inner-products with other inputs, e.g. $x_i^T x_j$

3. Define a Kernel function & substitute $K(x_i, x_j)$ for $x_i^T x_j$

Kernelized Linear Regression

- It minimizes the squared loss function.

$$\min_{w} \sum_{i=1}^{n} (w^T x_i - y_i)^2$$

- The prediction at a test point

$h(x) = w^T x$

$X = [x_1, x_2, \ldots x_n]$

$y = [y_1, y_2, \ldots y_n]^T$

- Sol: $w = (XX^T)^{-1} XY$

Scanned with CamScanner

Express the Solution as a linear Combination of train data

→ $$W = \sum_{i=1}^{n} \alpha_i x_i = X\vec{\alpha}$$

→ $$X\vec{\alpha} = (XX^T)^{-1}Xy$$

→ $$(X^TX)^{-1}X^TX\vec{\alpha} = (X^TX)^{-1}\underbrace{X^T(XX^T)^{-1}}_{I}Xy$$

$$\{X^T(XX^T)^{-1}X = I \;;$$
$$\underbrace{XX^T(XX^T)^{-1}}_{I}X = XI$$
$$IX = XI\}$$

→ $$\alpha = \underbrace{(X^TX)^{-1}}_{K}y$$

$K = K_{ij} = X_i^T X_j$

$$\boxed{\alpha = K^{-1}y}$$

If X^TX is not invertible then we add ridge

$$\boxed{\alpha = (K+\sigma I)^{-1}y}$$

↳ kernelized ridge regression

Ques: let $D = \{(x_1,y_1), \ldots (x_n,y_n)\}$
How can you kernelize nearest neighbour with euclidean distance.

$$\|x_1 - x_2\|^2 = (x_1-x_2)^T(x_1-x_2)$$
$$= x_1^Tx_1 - x_1^Tx_2 - x_2^Tx_1 + x_2^Tx_2$$
$$= x_1^Tx_1 - 2x_1^Tx_2 + x_2^Tx_2$$
$$K(x_1,x_2) = K_1(x_1,x_1) - 2K_2(x_1,x_2) + K_3(x_2,x_2)$$

$z \to$ test ~~point~~ data

Testing

$$h(z) = W^Tz$$
$$= \alpha^TX^Tz = \alpha^TK_*$$
$$= \alpha^T K(z,x)$$

$K_{ij} = K(z_i, x_i)$

K_* is the Kernel (vector) of test point with training points.

$$[K_*]_i = \phi(z)^T\phi(x_i)$$

- Kernelized linear regression is non-parametric algorithm, because here $\alpha_1, \ldots \alpha_n$ depends on number of training data not on the dimensions of the data.
- It is also a parametric algorithm because we compute

$$h(x) = W^T\phi(x)$$

- Kernel which does not give rise to high dimension of W, then it is also a parametric.
- If we use RBF then, W^T is highly dimensional then it becomes purely parametric algorithm ($\because$ fixed n α's)

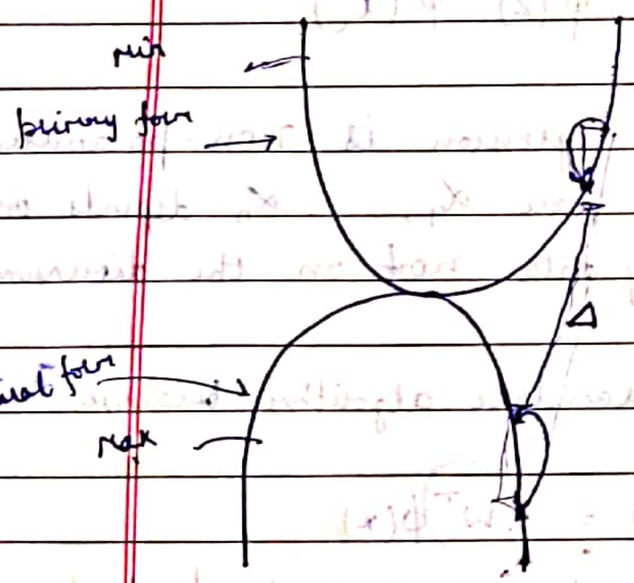
Kernel SVM

$$\min_{w,b} \; w^T w + C\sum_{i=1}^{n} \xi_i$$

$$\forall_i \quad y_i(w^T x_i + b) \geq 1 - \xi_i$$

$$\xi_i \geq 0$$

Another way of solving the optimisation problem :



we converge to minima when Δ becomes zero.

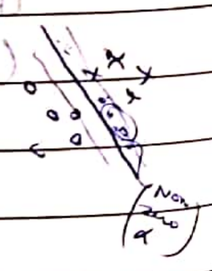
dual form

$$\min_{\alpha_1, \alpha_2, \ldots \alpha_n} \; \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j K_{ij} - \sum_{i=1}^{n} \alpha_i$$

s.t.

$$0 \leq \alpha_i \leq C$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0$$



$$w = \sum_{i=1}^{n} \alpha_i y_i \phi(x_i)$$

$$h(x) = \text{Sign}\left(\sum^{n} \alpha_i y_i K(x_i, x) + b\right)$$

from where 'b' will come ?

- one apparent problem with dual version is that b is no longer part of the optimisation

We know,

Support vectors are those with $\alpha_i > 0$.

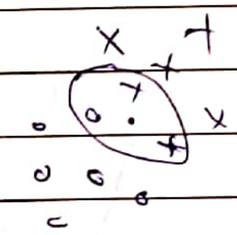
$$y_i\left(w^T \phi(x_i) + b\right) = 1$$

$$y_i\left(\sum_j y_j \alpha_j K(x_j, x_i) + b\right) = 1$$

$$b = y_i - \sum_j y_j \alpha_j K(x_j, x_i)$$

→

KNN



$$h(z) = \text{Sign}\left[\sum_{i=1}^{n} y_i \delta^m(x_i, z)\right]$$

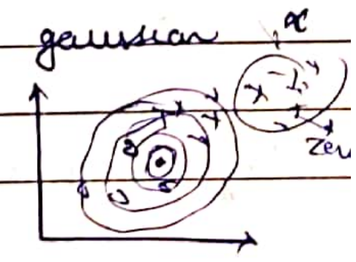
$$\delta^m \begin{cases} 1 & ; \text{ if NN} \\ 0 & ; \text{ o/w} \end{cases}$$

Compare it with Kernel SVM.

$$K(x_i, z) = e^{-\frac{(x_i - z)^2}{\sigma^2}}$$

SVM doing the same thing that K nearest neighbors doing, it does not make K fixed, it considers all the training data, but it makes gaussian around the ~~test~~ test point.

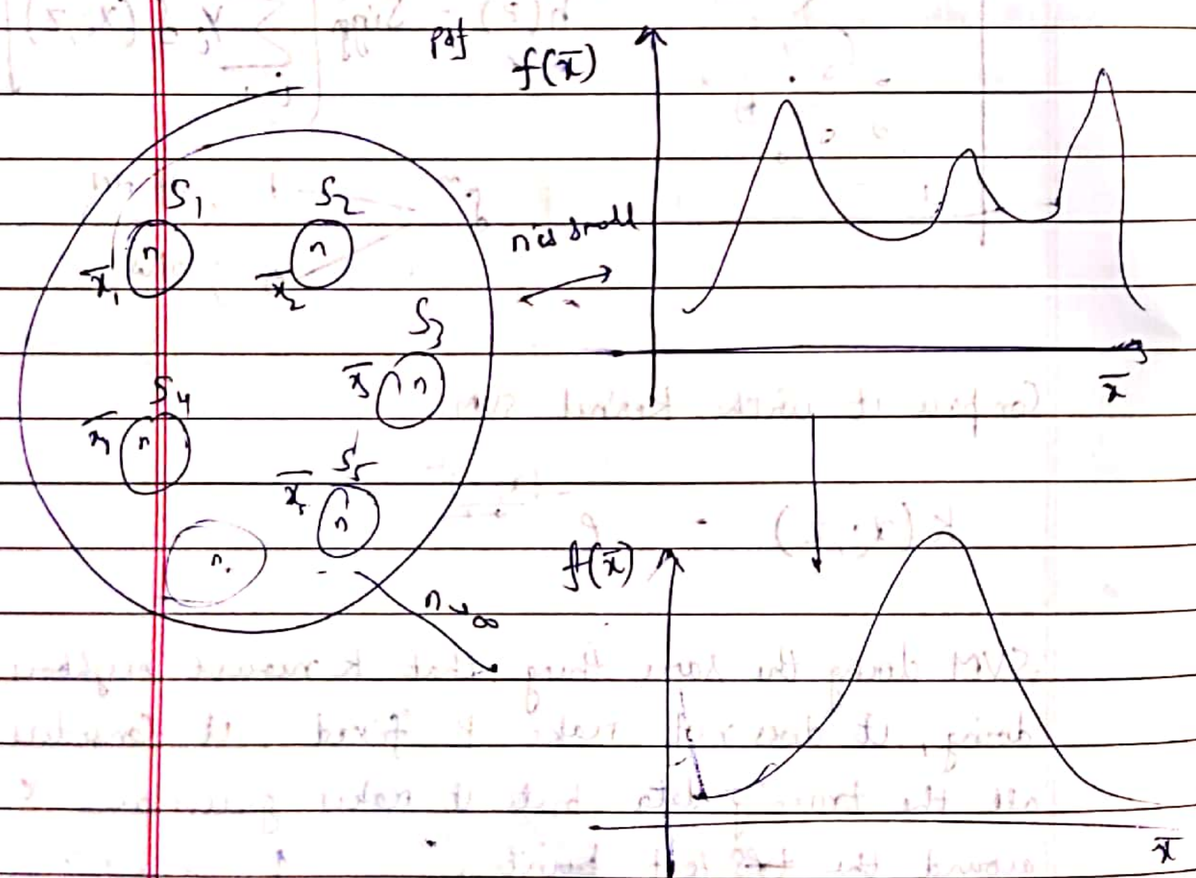
It's doing the weighted average of the neighbors.



SVM is a soft nearest neighbors.
In KNN, we have hard cut off for K,
In SVM, we have soft cutt off.

→ What about α?
See dual form, α are non negative. $(0 \le \alpha \le C)$
Assigning additional weights, most of the α's are zero which are obvious, so basically we are removing those training points as KNN does.

SVM especially in RBF Kernel acts like a smart nearest neighbor.

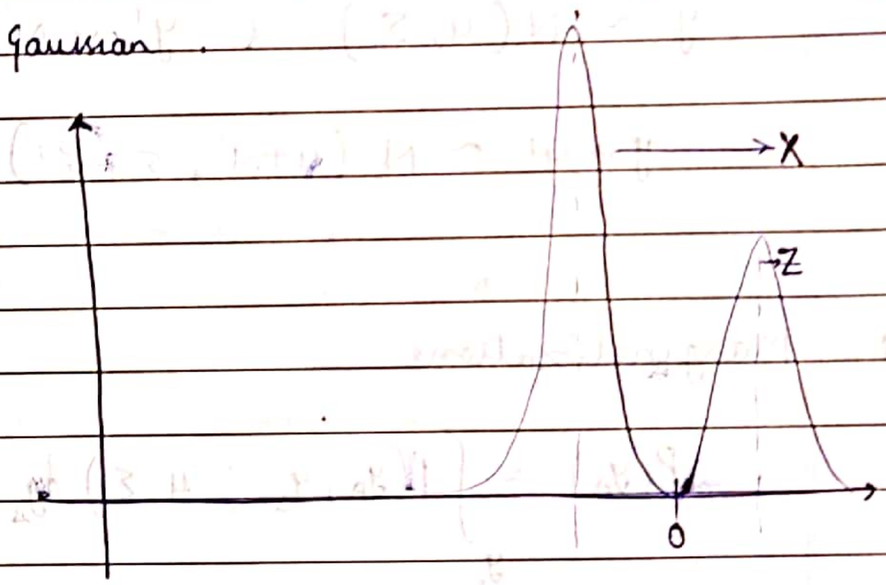


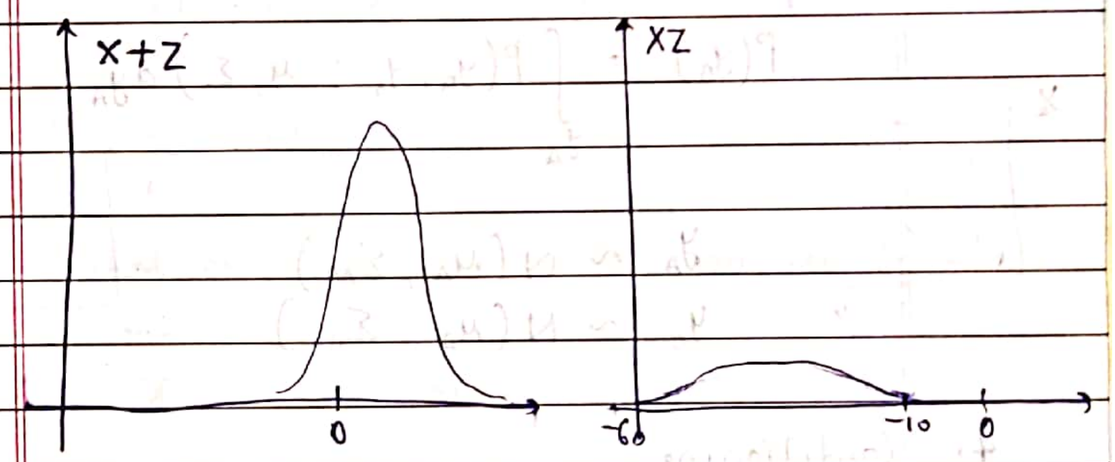
Height is also an average of many factors.

Lec 26 : Gaussian processes

- Height of People given Gender
- IQ of People.
- Many things around us are gaussian distribution because of the Central limit theorem.

- Gaussian distribution is a black hole of all the distributions, i.e. once a Gaussian, stays a Gaussian.





Let Gaussian r.v. $y = \begin{bmatrix} Y_A \\ Y_B \end{bmatrix}$, mean $\mu = \begin{bmatrix} \mu_A \\ \mu_B \end{bmatrix}$

& Covariance matrix $\Sigma = \begin{bmatrix} \Sigma_{AA}, & \Sigma_{AB} \\ \Sigma_{BA}, & \Sigma_{BB} \end{bmatrix}$

1. Normalisation

$$\int_y P(y;\mu,\Sigma)\,dy = 1$$

2. Summation :

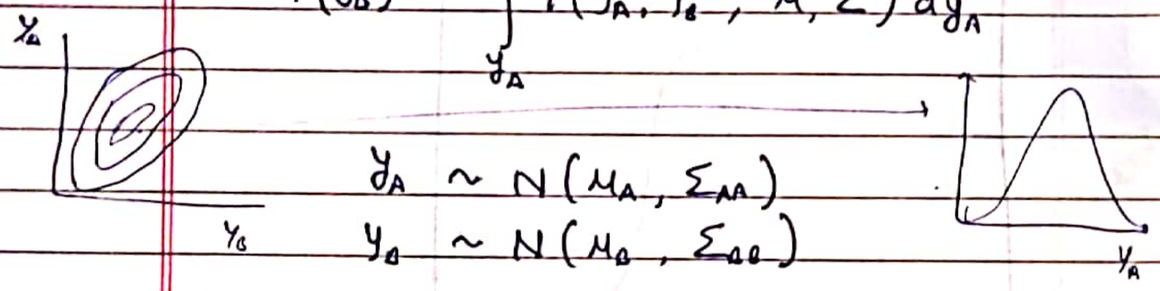
$$y \sim N(\mu,\Sigma) \quad \& \quad y' \sim N(\mu',\Sigma')$$

$$y+y' \sim N(\mu+\mu', \Sigma+\Sigma')$$

3. Marginalization

$$P(y_A) = \int_{y_B} P(y_A, y_B ; \mu, \Sigma)\, dy_B$$

$$P(y_B) = \int_{y_A} P(y_A, y_B ; \mu, \Sigma)\, dy_A$$



$$y_A \sim N(\mu_A, \Sigma_{AA})$$
$$y_B \sim N(\mu_B, \Sigma_{BB})$$

4. Conditioning

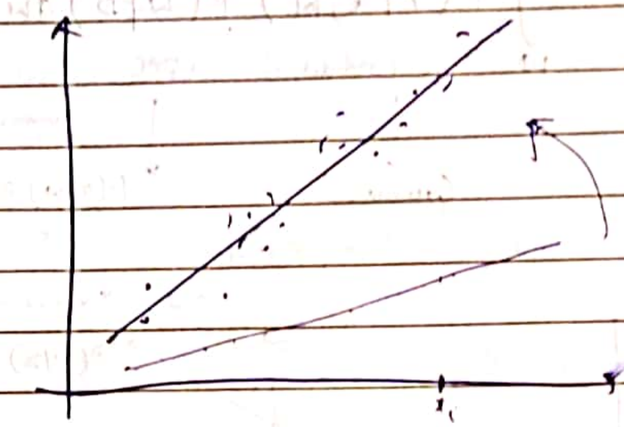
$$P(y_A|y_B) = \frac{P(y_A, y_B ; \mu, \Sigma)}{\int_{y_A} P(y_A, y_B ; \mu, \Sigma)\, dy_A}$$

is also a Gaussian

$$y_A|y_B \sim N\left(\mu_A + \Sigma_{AB}\Sigma_{BB}^{-1}(y_B-\mu_B),\ \Sigma_{AA} - \Sigma_{AB}\Sigma_{BB}^{-1}\Sigma_{BA}\right)$$

Gaussian process Regression

→ Linear Regression



$$f(x) = W^T x + \epsilon$$

Two ways to find W :

- MLE $\quad \underbrace{P(D|W)}_{\text{Gaussian}} = \prod_{i=1}^{n} P(y_i|x_i;W)$

- MAP $\quad P(W|D) \propto \dfrac{P(D|W)\cdot P(W)}{Z}$ → prior

$$P(y_i|x_i;W) \sim N(W^T x, \epsilon^2 I)$$

Flow : $D \xrightarrow{\text{Learn}} W \xrightarrow{\text{prediction}} y = W^T x$

Instead of learning W, we can use Bayesian approach.

$x \rightarrow$ test point

$$P(y|x,D) = ?$$

Here we donot committing any assumption & not focusing any model.

$$\int_W P(Y|x, D, W)\, dW$$

the moment I have W
$P(Y|x)$ depend on D
& W depends on D

$$P(Y|x,D) = \int_W \underbrace{P(Y|x,W)}_{\text{final Model}}\ \underbrace{P(W|D)}_{\text{MAP}}\, dW$$

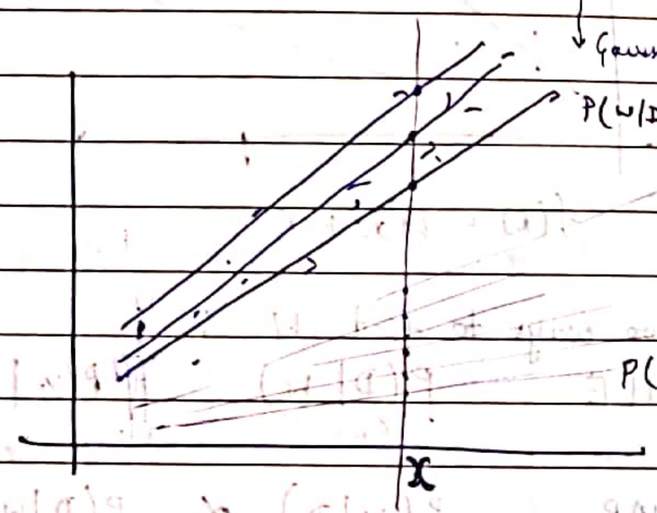
$P(Y|x,D)$ → Gaussian

final Model → Gaussian

MAP → $P(D|W)\,P(W)$ → Gaussian, Gaussian

$P(D|W)$ → Gaussian

$P(W|D)$ → large



$P(W|D)$ very small

final Y is the average of all possible lines. Dark lines are more likely so answer also lie close to thick lines.

Now, we have shown that $P(Y|x,D)$ is also a Gaussian distribution

$$P\left(\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \\ y_+ \end{bmatrix}, [x_1, \dots x_n], x_+\right) \sim N(\mu, \Sigma)$$

$$\Sigma = \begin{array}{c} \\ \text{Killer} \\ \text{Neigh} \\ \text{D.T} \end{array} \begin{array}{c} \begin{array}{ccc} \text{Kill} & \text{Neig} & \text{D.T} \end{array} \\ \begin{bmatrix} 1 & 0.9 & 0 \\ 0.9 & 1 & 0 \\ 0 & 0 & 10 \end{bmatrix} \end{array}$$

kernel (∵ pos. semi-definite)

To make prediction, Compute covariance matrix.

What we want is ~~large~~ a function which results in large no. if prices are highly correlated & very small if not the case.

$$K(x,z) = e^{-\frac{(x-z)^2}{\sigma^2}}$$

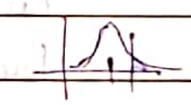
$$\Rightarrow P(Y|x,D) \sim N(0, \Sigma)$$

$$\Sigma = K \qquad K_{ij} = K(x_i, x_j)$$

prediction:

$$P(Y_+ | D, x_+) \sim N\left(\underbrace{K_*^T K^{-1} Y}_{\text{Kernel Regression}},\ \underbrace{K_{**} K^{-1} K_*}_{\text{Variance}}\right)$$

$$\Sigma = \begin{bmatrix} K & K_* \\ K_*^T & K_{**} \end{bmatrix}$$

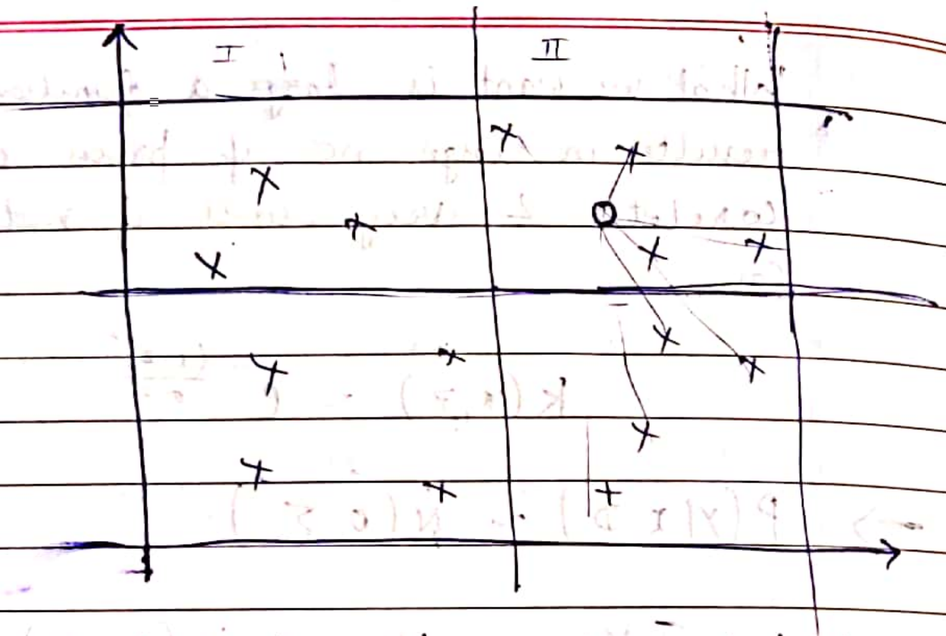


K → training Kernel Matrix
K_* → training - testing Kernel Matrix
K_*^T → testing - training Kernel matrix
K_{**} → testing Kernel matrix

Advantage:- While regressing, you have Variance with each regression of Y.

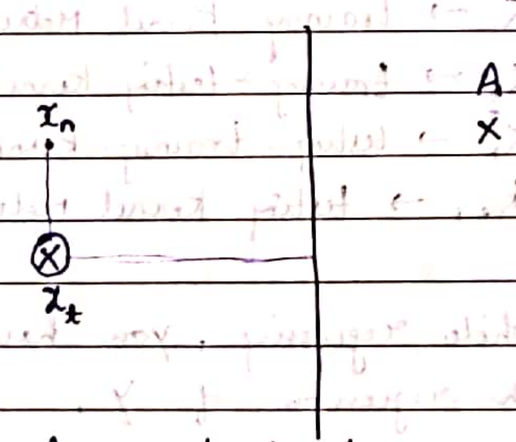
- Use to find the minimisation of a unknown expensive function.
- Also used to optimise hyper parameter of a ML Algorithm.

Lec27: KD Trees



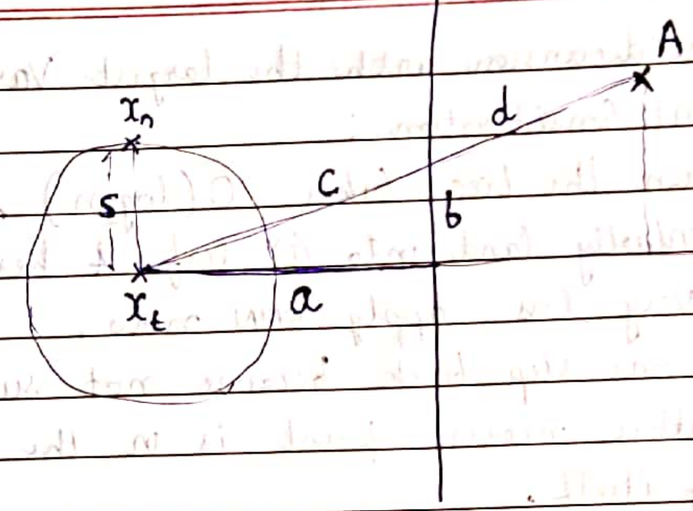
- KNN is much slower because (O) test point goes to every training point & asks are I close to you?
- One way is to find nearest neighbor is :
  Divide the data into two parts.
  Rule out the I half.
- Problem with the point near to plane.
- Solution : Increase the number of split.

Ques :



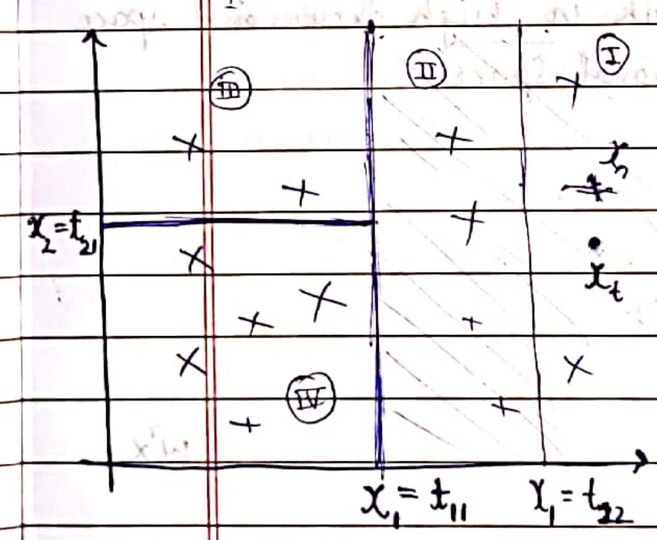
- prove point A must be further away from x_t provided that A lies on different split.

proof :

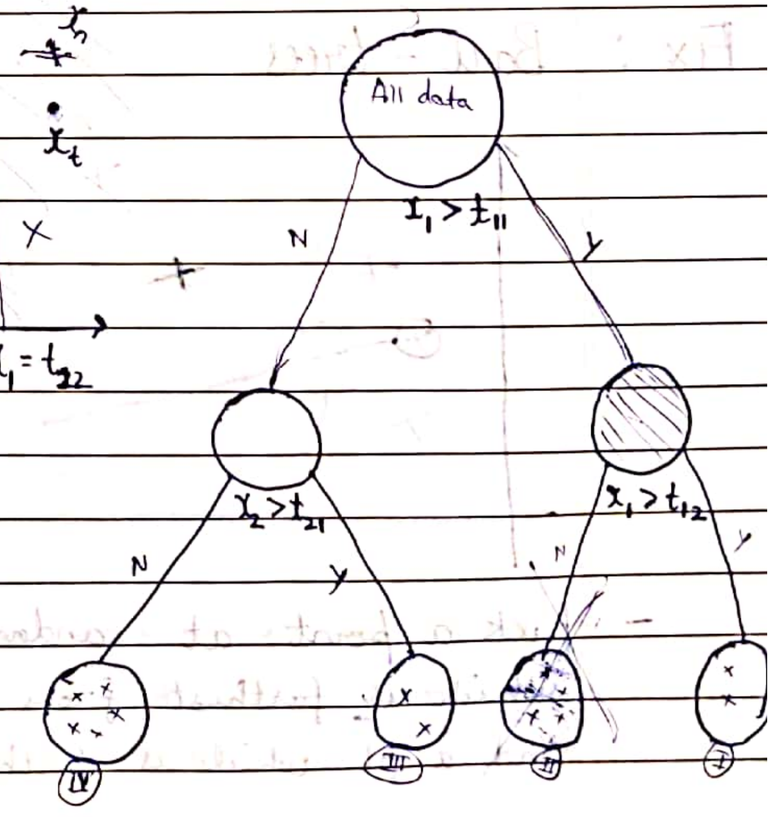


$$c + d = \sqrt{a^2+b^2} + d$$
$$\geq \sqrt{a^2} + d$$
$$\geq a \geq s$$

- KD trees is an idea to faster the KNN.



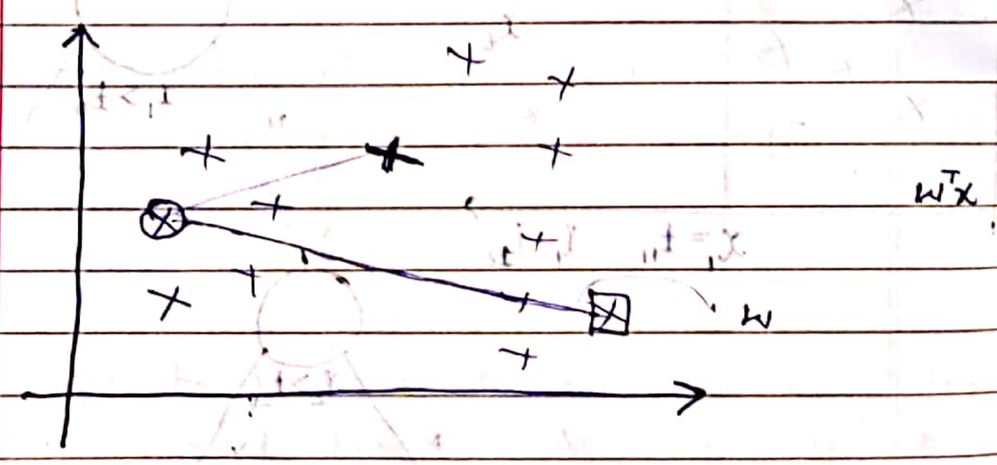
New Data Structure



- Take one diversion with the largest Variance (x_1)
- Test point Consideration :
  - Going down the tree takes $O(\log_2 n)$ seconds.
    - Eventually land into (I) leaf & here data is very less, apply KNN now.
    - Go one step back because not sure whether nearest point is in the same leaf itself.
  - In this case, we go back to ⊗ ; (I & II) which means decision boundary $x_1 = t_{12}$.
  - Now Check, is it worth going to ⊗.
  - It is worth only if the distance to the wall is less that current nearest neighbor.

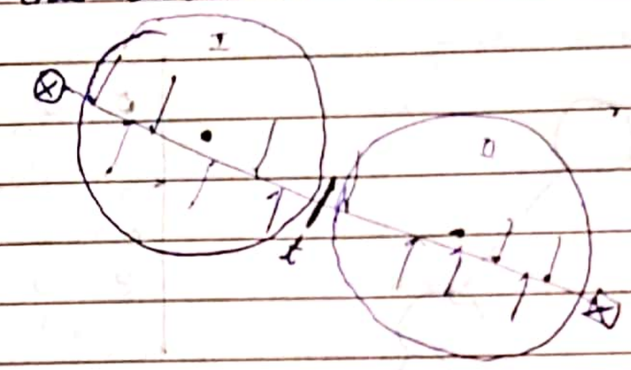
→ KD-trees don't work in high dimensional space.
→ Good in low-dimensional space.

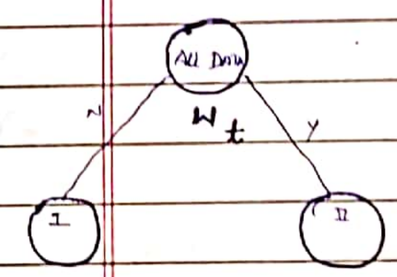
Fix : Ball - trees



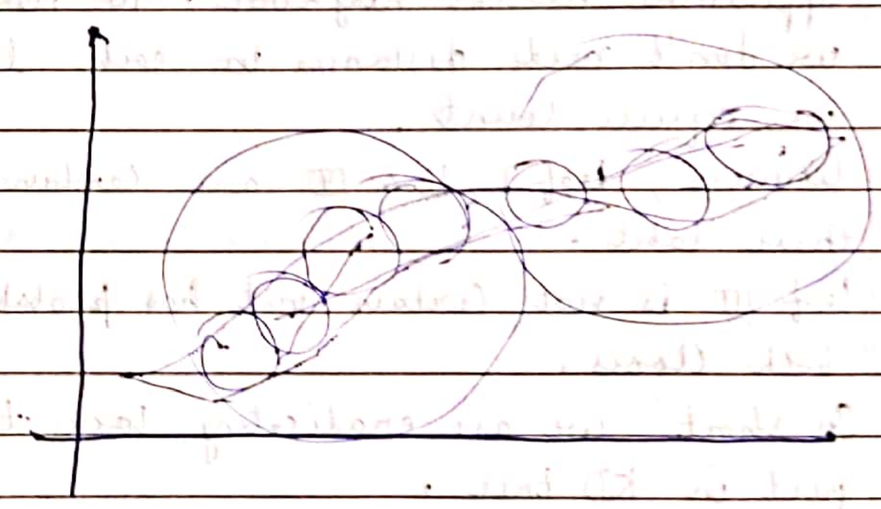
- pick a point at random x, find point which is farthest from x, ⊗. Again find a pt. which is farthest from ⊗, [x].

- We get a direction w, project all points on this direction.





- Ball trees tend to still work if data exhibits local structure (e.g. lies on a low-dim. manifold).

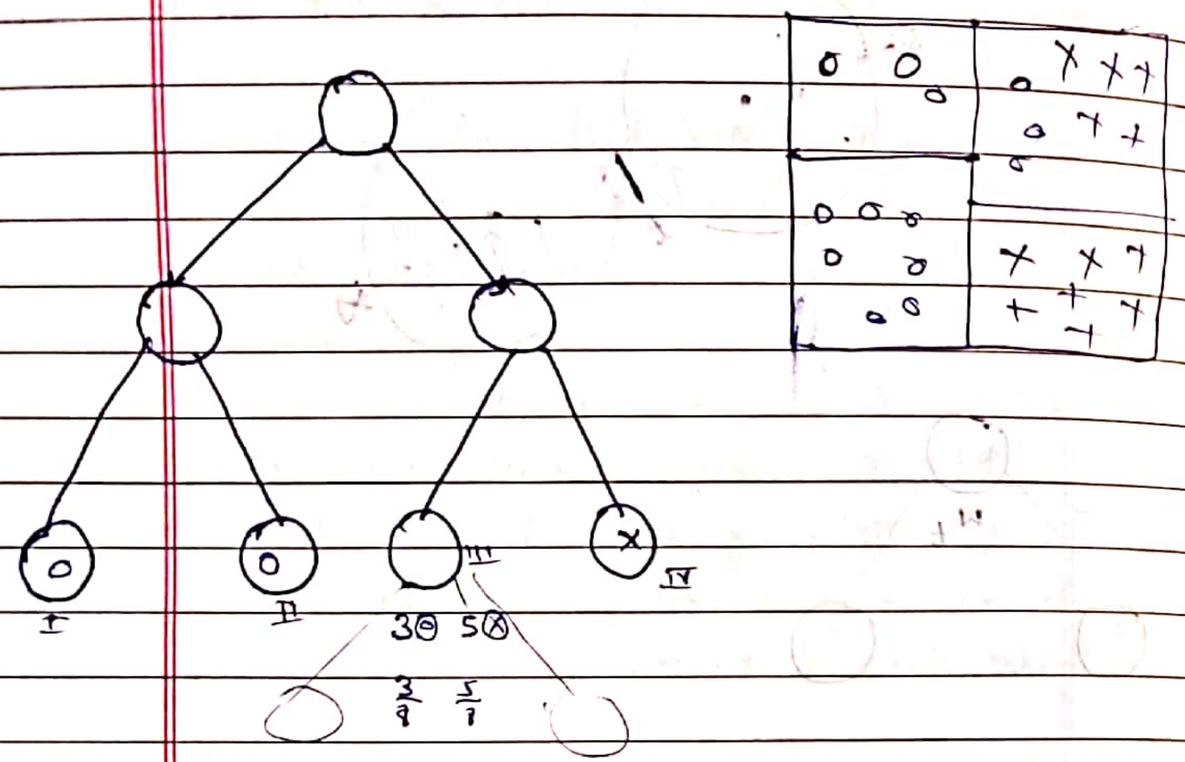


- In terms of Computation, we need to compute three distances & also few distances from the ball to the test point to rule out.

Lec 28 : Decision trees

Decision trees

- Make KD tree



- In KD tree, we compute exact nearest neighbor by one step back method.
- To make it more faster, we will use approximate nearest neighbor. To make prediction we don't need distances in each leaf. we require count.
- Likewise, leaf I, II & IV are certain with their label.
  Leaf III is not certain but has probability for both classes.
  In short, we are eradicating back tracing part in KD tree.

Ques : Can we split such that leaf III scenerio doesn't occur ?

- Yes, we can except when two identical features have different label.
- But splitting in such fashion is not a good practice, because it leads to overfitting.

High bias & High Variance is a function of depth of tree.

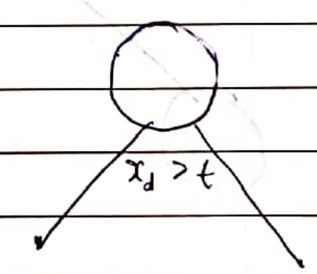
(underfitting) High bias → Small depth (High Train error)
(overfitting) High Variance → Large depth (High test error)

→ Goal of Decision tree
- To find the smallest tree with low training error.
- n-p hard problem

Problem : We don't want balance tree like in KD tree but we want pure leaf.

Impurity functions

- It measures how pure a set is.



(I) Gini impurity

$$S = \{ (x_1, y_1), \ldots (x_n, y_n) \}$$

$$S = S_1 \cup S_2 \cup \ldots S_K$$

$$P_K = \frac{|S_K|}{|S|}$$

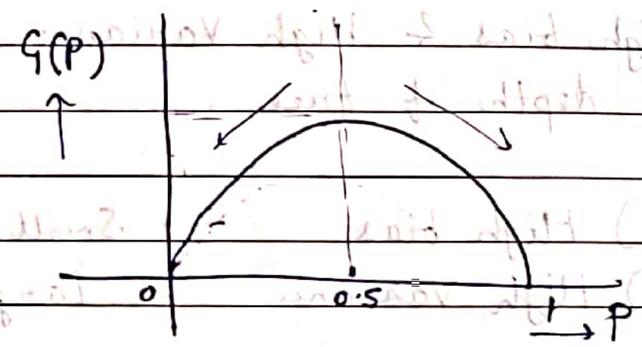
$K \rightarrow$ Class

$$S_K = \{ (x, y) \in S \mid y = K \}$$

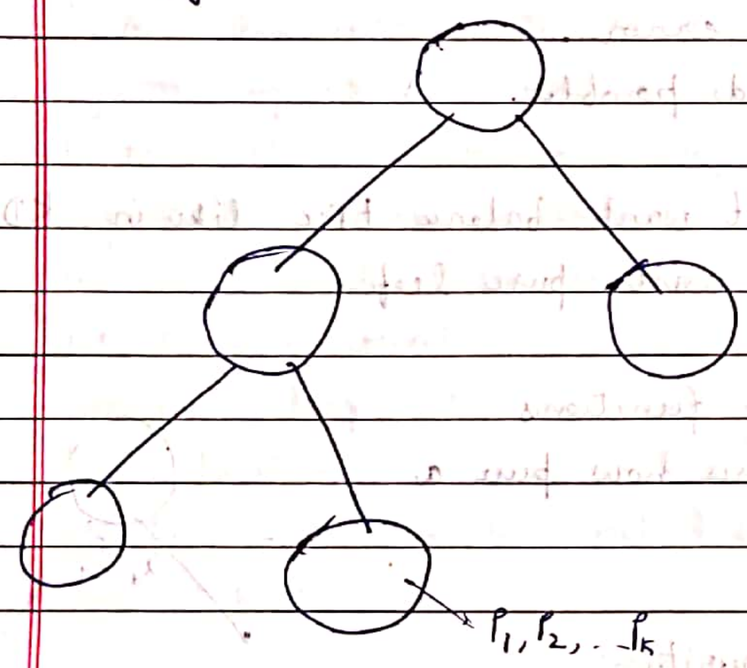
$$G_i(S) = \sum_{k=1}^{K} P_k(1-P_k)$$

e.g. two classes

$$G(S) = P(1-P) + P(1-P)$$
$$= 2P(1-P)$$



(II) Entropy



We have defined gini impurity which tells us about how good we are i.e. how pure our leaf & if not the case then split it further.

Worst Case (Entropy) : When prob. of each class in a leaf is same.

$$q_1 = q_2 = \cdots q_K = \frac{1}{K} \quad \left\{\text{Uniform distribution}\right\}$$

If so the case, we haven't achieved anything during the split.
So, Entropy depicts how good our decision tree is.

- K-L divergence computes how close two distributions are.

$$KL(P\|q) = \sum_{k=1}^{K} P_k \log \frac{P_k}{q_k}$$

$KL(P\|q) \to 0$ Similar distribution
$\to 1$ different

Goal : Maximize the Entropy.

$$KL\left(P\|\tfrac{1}{K}\right) = \sum_{k=1}^{K} P_k \log\left(\frac{P_k}{1/K}\right)$$

$$= \sum_{k=1}^{K} P_k \log(P_k) + \sum_{k=1}^{K} P_k \log K$$

$$= \sum_{k=1}^{K} P_k \log(P_k) + \underbrace{\log K}_{c}$$

$$\max_{P} \sum_{k} P_k \log(P_k)$$

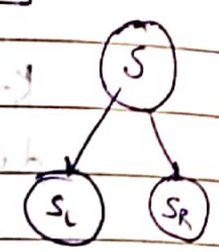
$$\min_{P} \underbrace{-\sum_{K} P_K \log(P_K)}_{\text{Entropy}}$$

$$\min H(S) \qquad O(ndK)$$

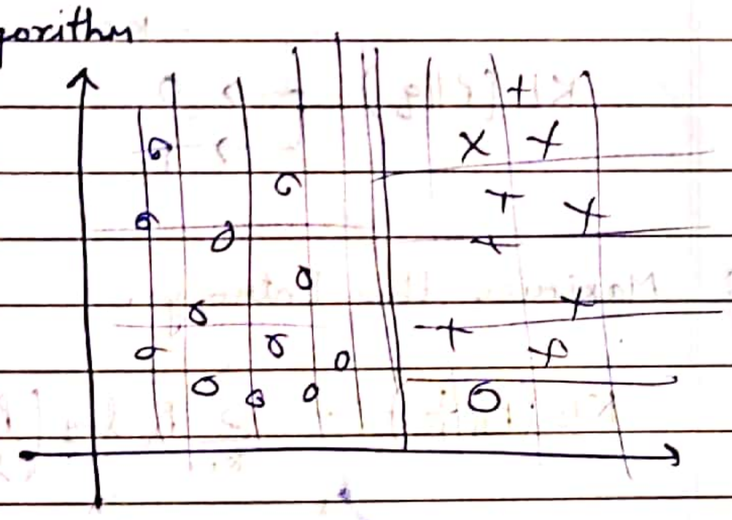
→ Entropy over tree.

$$\boxed{H(\text{Tree}) = P_L \cdot H(S_L) + P_R H(S_R)}$$

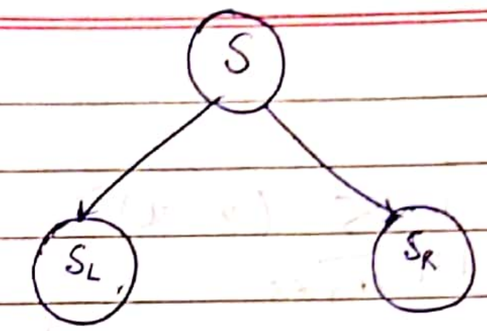
$$P_L = \frac{|S_L|}{|S|} \qquad P_R = \frac{|S_R|}{|S|}$$



Trivial Algorithm



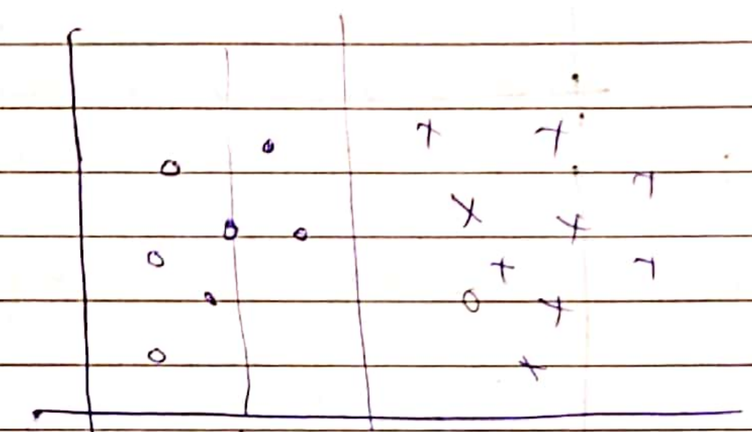
- Split, Compute entropy, repeat.
- Select that split which has low entropy.



Impurity of Split : $\frac{|S_L|}{|S|} H(S_L) + \frac{|S_R|}{|S|} H(S_R)$

K - Classes
$|S|$ - N
Splits - ND

$$O(ND \cdot KN) = O(DKN^2)$$



- Count of dots increase, use previous step (split) info & it reduces your complexity.

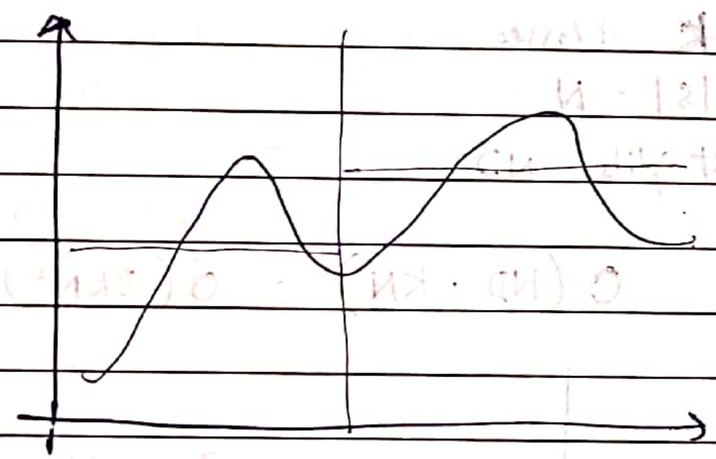
$$O(ND \cdot K\cancel{N}) = O(DKN)$$

CART

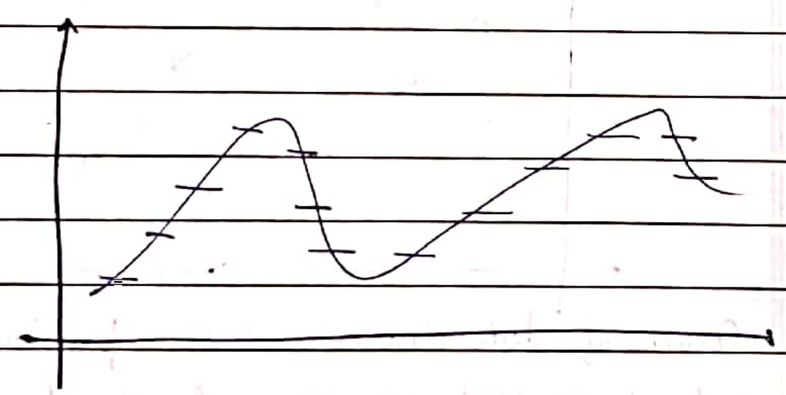
$$L(S) = \frac{1}{|S|} \sum_{(x,y)\in S} (y-\mu)^2$$

$$\mu = \frac{1}{|S|} \sum_{(x,y)\in S} y$$

{ use any Complex loss, advantage of CART }



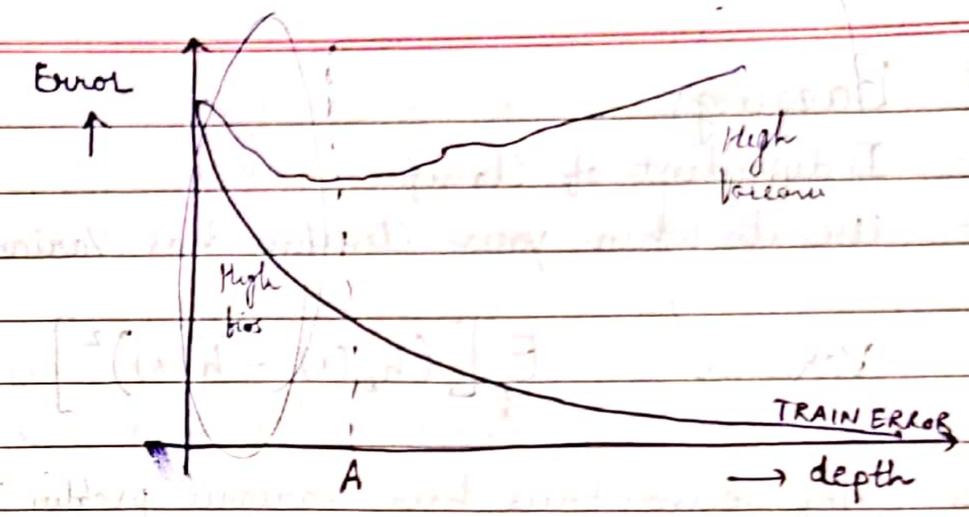
eventually;



Ques When to Stop ?

- Stop untill M points left in a leaf.
- Or define the hyperparameter 'max depth'. i.e. grow your tree untill some depth.

→ like in linear Classifier we had loss function & regularisation term, In Decision trees, we have loss function & max-depth (regularisers).



Downfall of Decision trees

From the above plot, it is seered that the sweet spot is at A.

This A can only take integer Value unlike λ in linear Classifier.

It doesn't allow us the fine grain but analysis of trade off between bias & Variance.

In practice, they don't really work good.

* Finding sweet spot is a challenge in decision tree.

FIX : Bagging & Boosting

Bagging

- Independent of Classifier
- Use it when your Classifier has Variance problem

$$\text{Variance} = E_{\substack{D \\ x}}\left[\left(h_D(x) - \bar{h}(x)\right)^2\right]$$

Ques: Why Decision trees have Variance problem?

- If we split untill we get a pure leaf, then that split is data specific. So surely it will not perform that good on different test set.

Let's say:

I have 'm' different datasets.

$D_1, D_2, \ldots D_M$

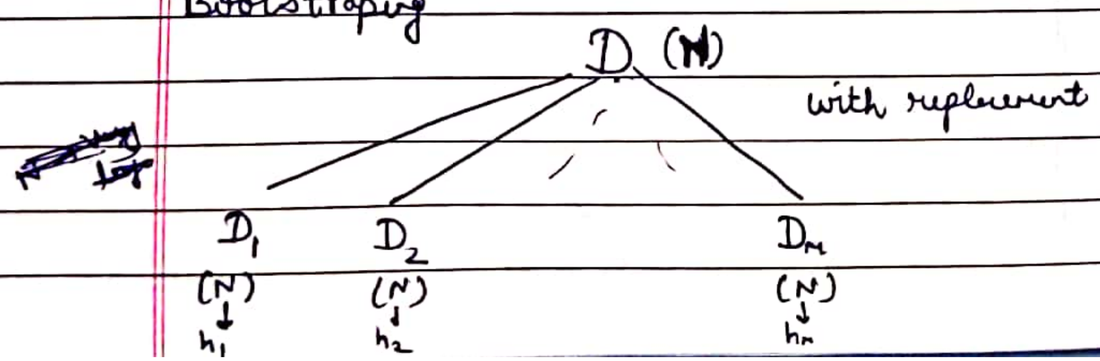
$$h(x) = \frac{1}{M}\sum_{j=1}^{M} h_{D_j}(x) \xrightarrow{M \to \infty} \bar{h}(x)$$

($\because$ By weak law of large number)

Catch: We donot have these 'm' datasets, we do have only 1 dataset.

** We make up these 'm' datasets using BOOTSTRAPING

Bootstraping

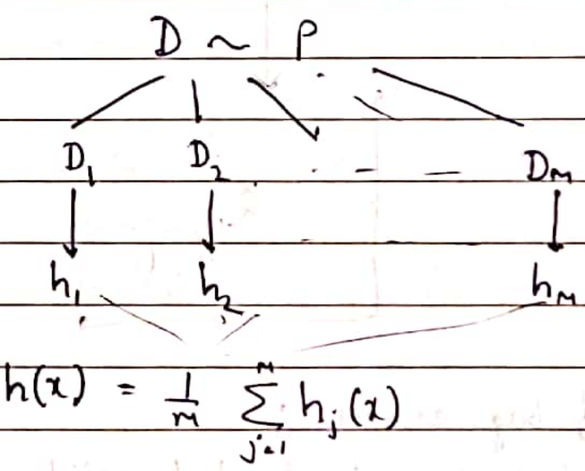


- Now, we take averaps of all different h. & if $m \to \infty$; $\text{Avg}(h_i) \neq \bar{h}(x)$; because $D_1, D_2, \ldots D_M$ are no longer independent. which Violates the assumption of weak law of large number.
- But it holds approximately.

Ques: Prove $D \sim P$; then $D_1, D_2, \ldots D_M \sim P$

- Let $Q(x, y \mid D)$ be a prob. distribution that picks a training sample (x_i, y_i) from D uniformly at random.

Random Forest

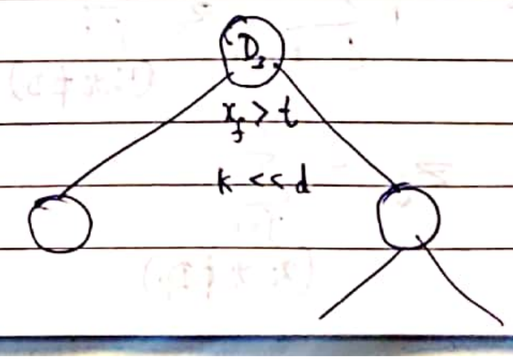


$$h(x) = \frac{1}{M}\sum_{j=1}^{M} h_j(x)$$

- It uses modified decision tree. i.e. it splits on K dimensions ($K \ll d$) to ensure different h_j.

K dimensions select them randomly

$K = \lfloor \sqrt{d} \rfloor$

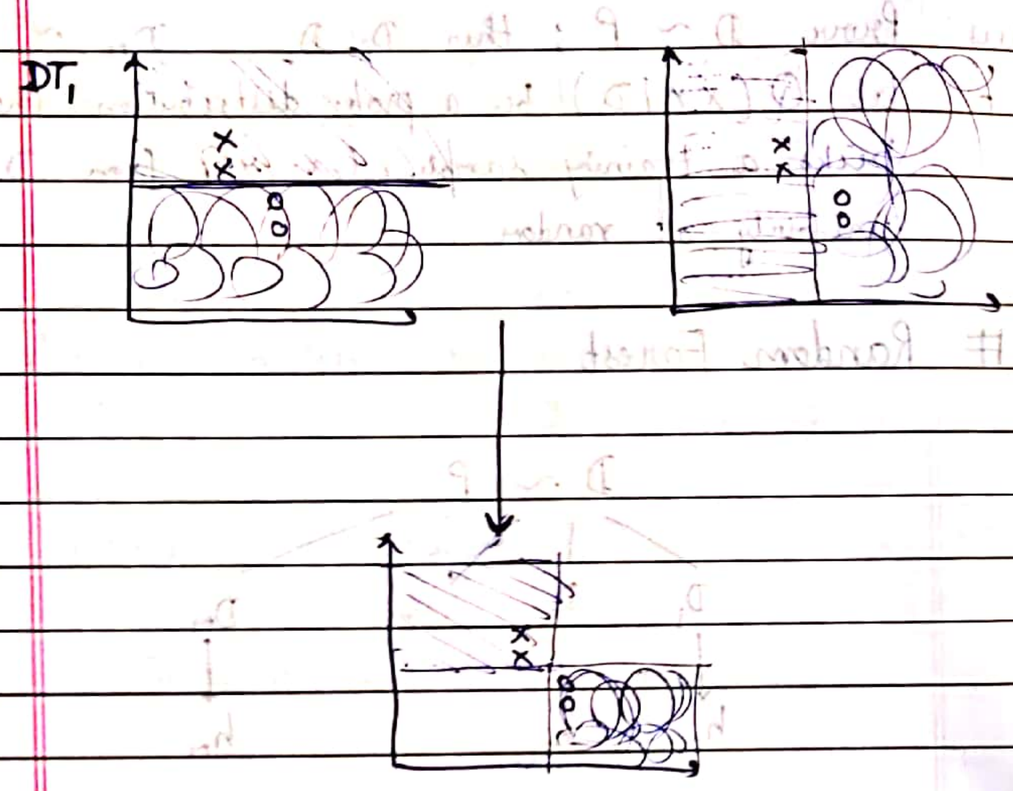


→ Advantage

(1)- You donot need to do feature scaling.

→ Hyperparameter (only two)

- $K = \lceil \sqrt{d} \rceil$
- m (trees) → as large as possible ← 100, 1000



(2)- out of bag error:

- No need of training & test split in every bagging algorithm (classifier).

$$\varepsilon_{OOB} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{1}{z_i}\sum_{\substack{j=1 \\ (x_i, y_i \notin D)}}^{m} l\left(h_j(x_i), y_i\right)\right)$$

$$z_i = \sum_{\substack{j=1 \\ (x_i, y_i \notin D_j)}}^{M} 1$$

Boosting

$$H(x) = \sum_{j=1}^{M} \alpha_j h_j(x)$$

weak learner (high bias)

- Reduces Bias.

$$l(H) = \frac{1}{n}\sum_{i=1}^{n} l\left(H(x_i), y_i\right)$$

Now, we have more weak learners (h), which h to be added to H.

$$\underset{h \in \mathcal{H}}{\operatorname{argmin}}\ l\left(H + \alpha h\right)$$

$\mathcal{H}$ → Class of weak learners (classifier with high bias)

- We had many weak learners that means their training error can never tends to zero.

So, can we come up with an algorithm who considers all these weak learners & ends up in strong learner (H)

$$H(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$$

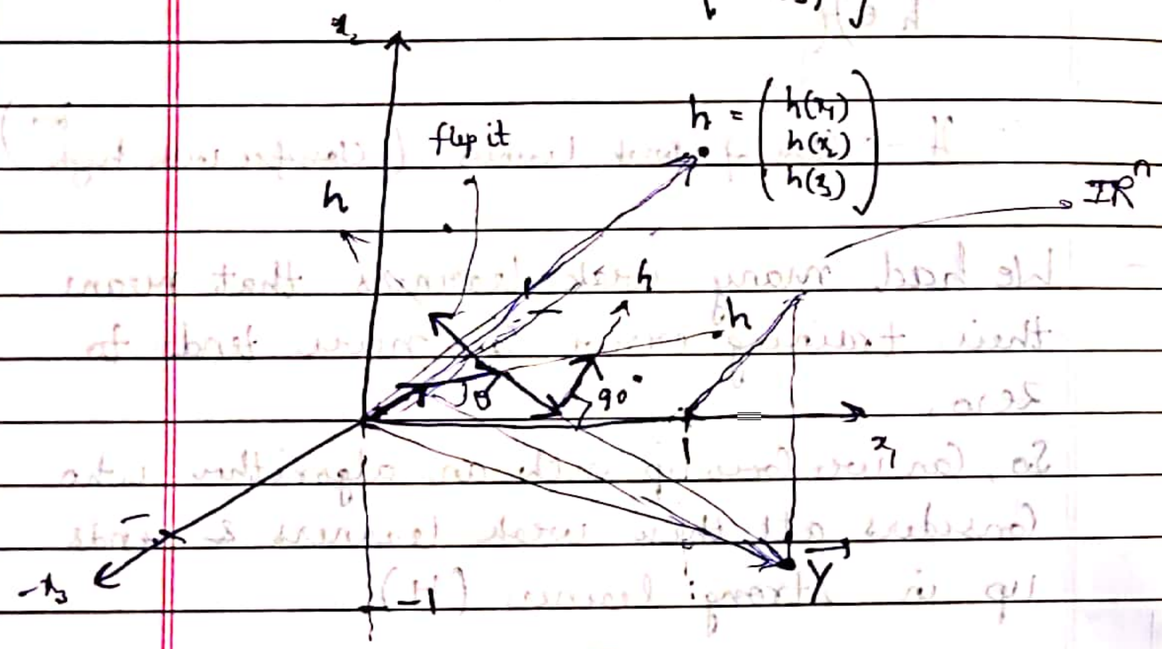
$h_t(x)$ → Weak learner (a ~~worse~~ just better than a coin toss in classification setting)

Intuition

$$D = \{ (x_1, 1), (x_2, -1), (x_3, 1) \}$$

$$Y = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix}$$

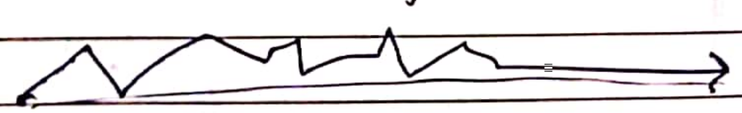
$$(x_1, x_2, x_3) \longrightarrow \begin{pmatrix} H(x_1) \\ H(x_2) \\ H(x_3) \end{pmatrix} \leftarrow \vec{H}$$



Can we take all weak learners 'h' to get at $\vec{Y}$ (i.e. training error zero).

Problem is when your step vector is orthogonal to the $\vec{Y}$.

Analogy : Drunk man reaching home.



$\langle \vec{a}, \vec{b} \rangle \rightarrow$ inner product

$$H(x) = \sum_{t=1}^{T} \alpha h_t(x)$$

$$h = \underset{h \to \mathcal{H}}{\arg\min} \; l(H + \alpha h)$$

It can be any loss ;

$$l(H) = \frac{1}{n} \sum_{i=1}^{n} (H(x_i) - y_i)^2$$

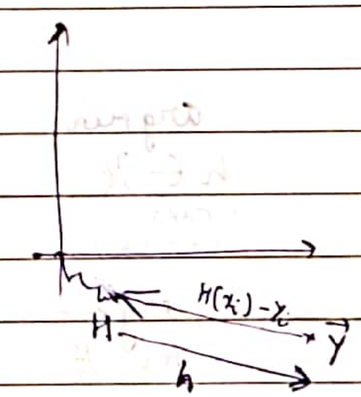
Using Taylor series :

$$l(H + \alpha h) \approx \underbrace{l(H)}_{\text{const.}} + \left\langle \frac{\partial l}{\partial H}, \alpha h \right\rangle$$

$$h = \underset{h \to \mathcal{H}}{\arg\min} \left\langle \frac{\partial l}{\partial H}, \alpha h \right\rangle$$

$$= \underset{h \to \mathcal{H}}{\arg\min} \sum_{i=1}^{n} \frac{\partial l}{\partial H(x_i)} \cdot \alpha h(x_i)$$

$$\frac{\partial l}{\partial H(x_i)} = H(x_i) - y_i$$



This is known as ANYBOOST.
(Gradient descent in functional space)

Google, yahoo, all search Engine using GBRT to predict what you are looking for.

Gradient Boosting trees :

$Y_i \in \mathbb{R}$

$h \leftarrow$ CART of limited depth (depth 4-6)

$\alpha \leftarrow$ small Constant

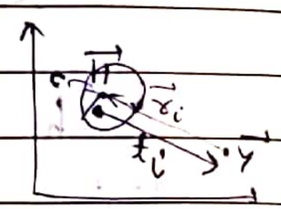
$$\underset{h \in \mathcal{H}}{\operatorname{argmin}} \; \alpha \sum_i r_i \, h(x_i) \qquad r_i = \frac{\partial l}{\partial H(x_i)}$$

$\mathcal{H} \rightarrow$ CART (d = 4,6)

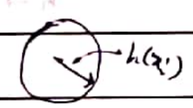
$$= -\frac{\partial l}{\partial H(x_i)}$$

$$\underset{h \in \mathcal{H}}{\operatorname{argmin}} \; -2 \sum_i t_i \, h(x_i)$$

CART (d = 4,6)



$$\left\{ \begin{array}{l} \sum_i t_i^2 \longrightarrow \text{Const.} \\ \sum_i h(x_i)^2 = c = 1 \end{array} \right\}$$

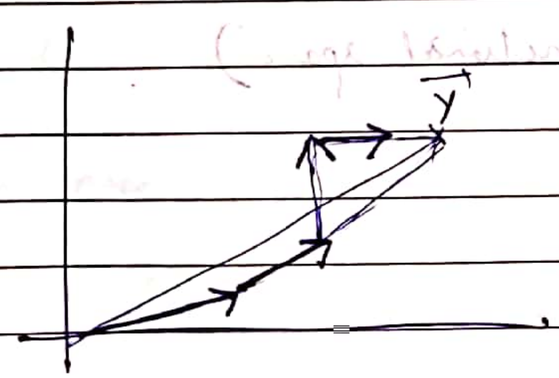


$$\underset{h \in \mathcal{H}}{\operatorname{argmin}} \sum h(x_i)^2 - 2 t_i \, h(x_i) + t_i^2$$

CART (d = 4,6)

$$\underset{h \in \mathcal{H}}{\operatorname{argmin}} \sum \left(h(x_i)^2 - t_i \right)^2$$

$\longrightarrow$ CART minimizes sq. loss.



PSEUDO code :

$H = \vec{0}$

For $t = 1 : T$

$t_i = Y_i - H(x_i)$

$h \leftarrow \underset{h \in \mathcal{H}}{\operatorname{argmin}} \sum (h(x_i) - t_i)^2$

$H \leftarrow H + \alpha h$

END

hyperparameter : - α.

- tree depth

Lec 35 : Neural Networks / Deep Learning (1963)
{Multi-layer perceptron}

Kernel SVM :

$$h(x) = w^T \phi(x)$$

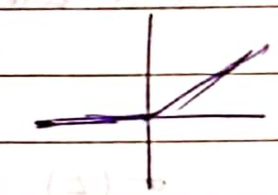
In dep learning :

$$\phi(x) = \sigma(\overbrace{Ax+b}^{\text{Some function transform}})$$

σ → Non-linear f^n.

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

Rectifier fn. → $\sigma(z) = \max(z, 0)$ →



Without σ function, it behaves like a linear classifier.

$$\begin{aligned} h(x) &= W^T \phi(x) \\ &= W^T (Ax+b) \\ &= W^T A x + W^T b \\ &= \hat{W} x + \hat{b} \end{aligned}$$

→ When you learn it, you learn W^T, A & b at the same time.

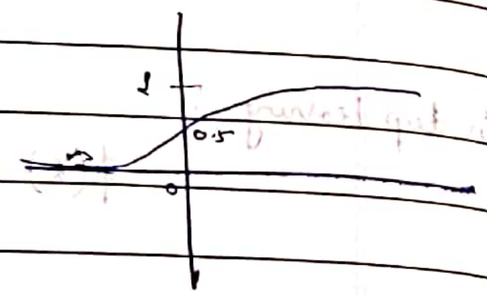
$$\begin{cases} x \rightarrow \phi(x) \\ h(x) = \phi(x)^T W + b \end{cases}$$

Scanned with CamScanner

$$\phi(x) = \sigma(Ax + c)$$

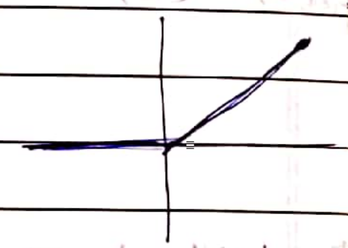
- Sigmoid function

$$\sigma(z) = \frac{1}{1+e^{-z}}$$



- Rectifier linear unit.

$$\sigma(z) = \max(z, 0)$$



$\longrightarrow$ $h(x) = W^T \phi(x) + b$ $\qquad \phi(x) = \sigma(Ux + \vec{c})$

$$\mathcal{L} = \sum_{i=1}^{n} l(h(x_i), y_i)$$

minimize it using gradient descent w.r.t. $\vec{W}$, $\vec{U}$ & $\vec{c}$.

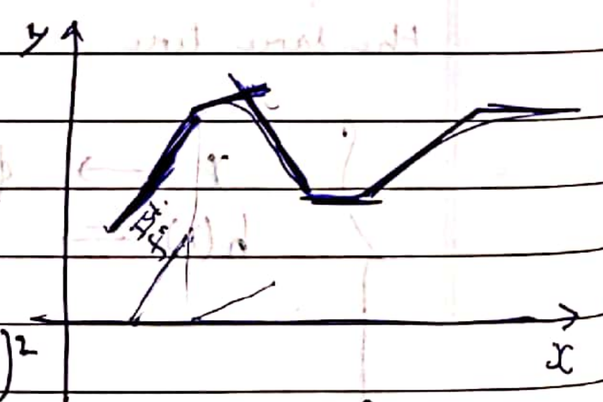
<u>Inside Neural Network</u>

$$\mathcal{L} = \sum_{i=1}^{n} (h(x_i) - y_i)^2$$

$$= \sum_{i=1}^{n} (W^T \sigma(Ux_i + \vec{c}) - y_i)^2$$

$$= \sum_{i=1}^{n} (W^T \max(Ux_i + \vec{c}, 0) - y_i)^2$$

$$= \sum_{d} W_d \max(\underbrace{U_d^T x + c_d}_{\text{Hyper plane (in 2D - line)}}, 0) + b$$



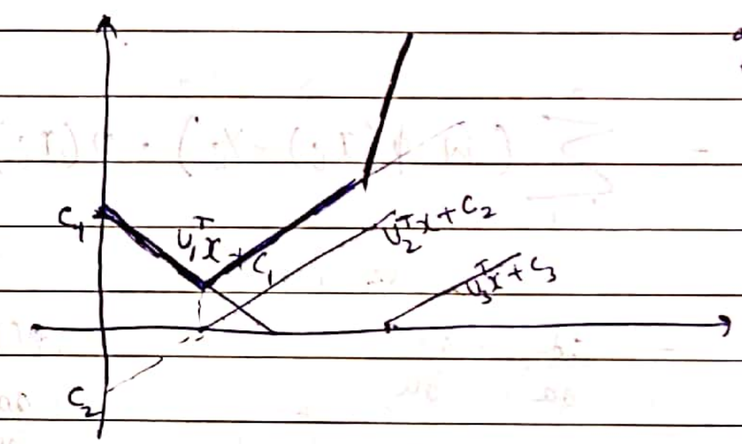
$$U = \begin{bmatrix} \text{---} & u_1 & \text{---} \\ \text{---} & u_2 & \text{---} \\ & \vdots & \\ \text{---} & u_h & \text{---} \end{bmatrix} \qquad \begin{bmatrix} c_1 \\ \vdots \\ c_h \end{bmatrix}$$

$$Ux = \begin{bmatrix} U_1^T x \\ \vdots \\ U_h^T x \end{bmatrix}$$

$U \in \mathbb{R}^{h \times d}$

$h \rightarrow$ we choose

$\hookrightarrow$ how many piecewise fn you allow to estimate the function.



- It approximates functions by piece-wise linear components.
- If we increase 'h', we are approximating the function very accurately.

<u>Layers</u>: $h(x) = W^T \phi(x)$

$$\text{layers} \leftarrow \begin{cases} \phi(x) = \sigma(U \phi'(x) + \vec{c}) \\ \phi'(x) = \sigma(U' \phi''(x) + \vec{c}') \\ \phi''(x) = \sigma(U'' x + \vec{c}'') \\ \vdots \end{cases}$$

Too many layers : Deep.

Scanned with CamScanner

$$\mathcal{L}(h) = \sum_{i=1}^{n} \ell(h(x_i), y_i)$$

$$= \frac{1}{2}\sum_{i=1}^{n} (h(x_i) - y_i)^2$$

$$h(x) = W^T \phi(x), \quad a(x)$$

$$\phi(x) = \sigma(U\phi'(x))$$

$$\phi'(x) = \sigma(\underbrace{U'\phi''(x)}_{a'(x)})$$

$$\phi''(x) = \sigma(\underbrace{U''x}_{a''(x)})$$

<u>optimisation</u>:

$$\frac{\partial \mathcal{L}}{\partial W} = \sum_{i=1}^{n} (W^T\phi(x_i) - y_i) \cdot \phi(x_i)$$

$$\frac{\partial \mathcal{L}}{\partial U} = \frac{\partial \mathcal{L}}{\partial a} \cdot \frac{\partial a}{\partial U} \quad \text{stored}$$

$$a(x) = U\phi(x)$$

$$\frac{\partial a}{\partial U} = \phi(x)$$

$$\frac{\partial \mathcal{L}}{\partial U'} = \frac{\partial \mathcal{L}}{\partial a'} \cdot \frac{\partial a'}{\partial a'} \cdot \frac{\partial a'}{\partial U'}$$

$$\cancel{\frac{\partial \mathcal{L}}{\partial a''} = \left(\frac{\partial \mathcal{L}}{\partial a''}\right) \cdot \frac{\partial a''}{\partial U''}}$$

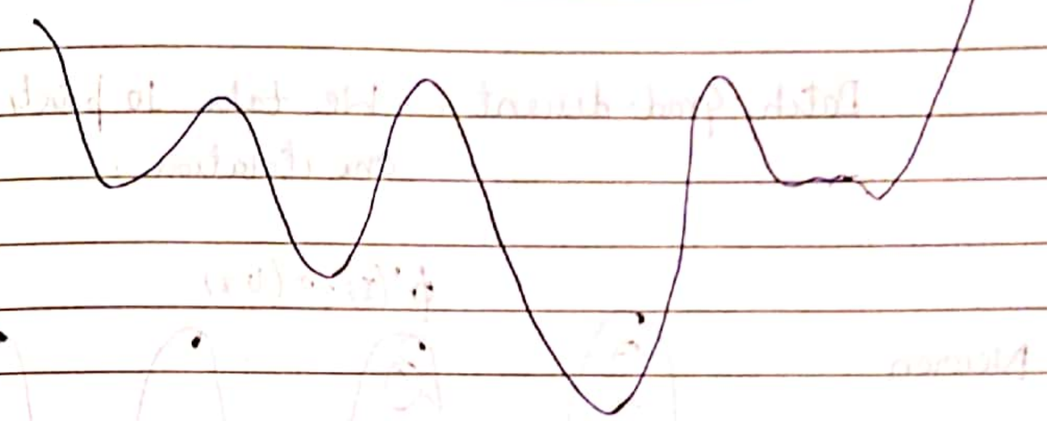
$$W \leftarrow W - \alpha \frac{\partial \mathcal{L}}{\partial W}$$

$$U \leftarrow U - \alpha \frac{\partial \mathcal{L}}{\partial U}$$

problem: Loss function is not convex; because of $\phi(x_i)$ {which is a non linear transformation}

Convex w.r.t. W, but not U, U', U'', ...

So, we are minimising a function like this;



Earlier, initialisation was irrelevant, but now it is no longer irrelevant.

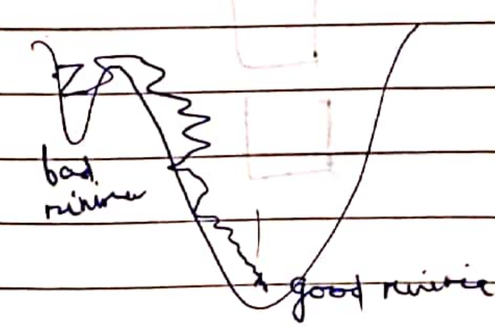
Stochastic Gradient descent:

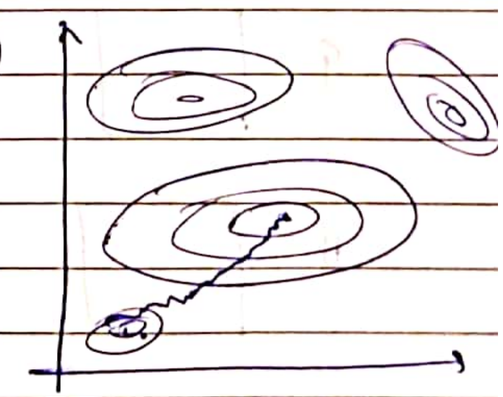
$$\mathcal{L}(h) = \sum_{i=1}^{n} \mathcal{L}(h(x_i), y_i) \quad \text{eg. } \frac{1}{2}\sum_{i=1}^{n} (h(x_i) - y_i)^2$$

Gradient $\frac{\partial \mathcal{L}}{\partial W} = \sum_{i=1}^{n} \frac{\partial \mathcal{L}(h(x_i), y_i)}{\partial W}$

$$\frac{\partial \mathcal{L}}{\partial U} = \sum_{i=1}^{n} \frac{\partial \mathcal{L}}{\partial U}(h(x_i), y_i)$$

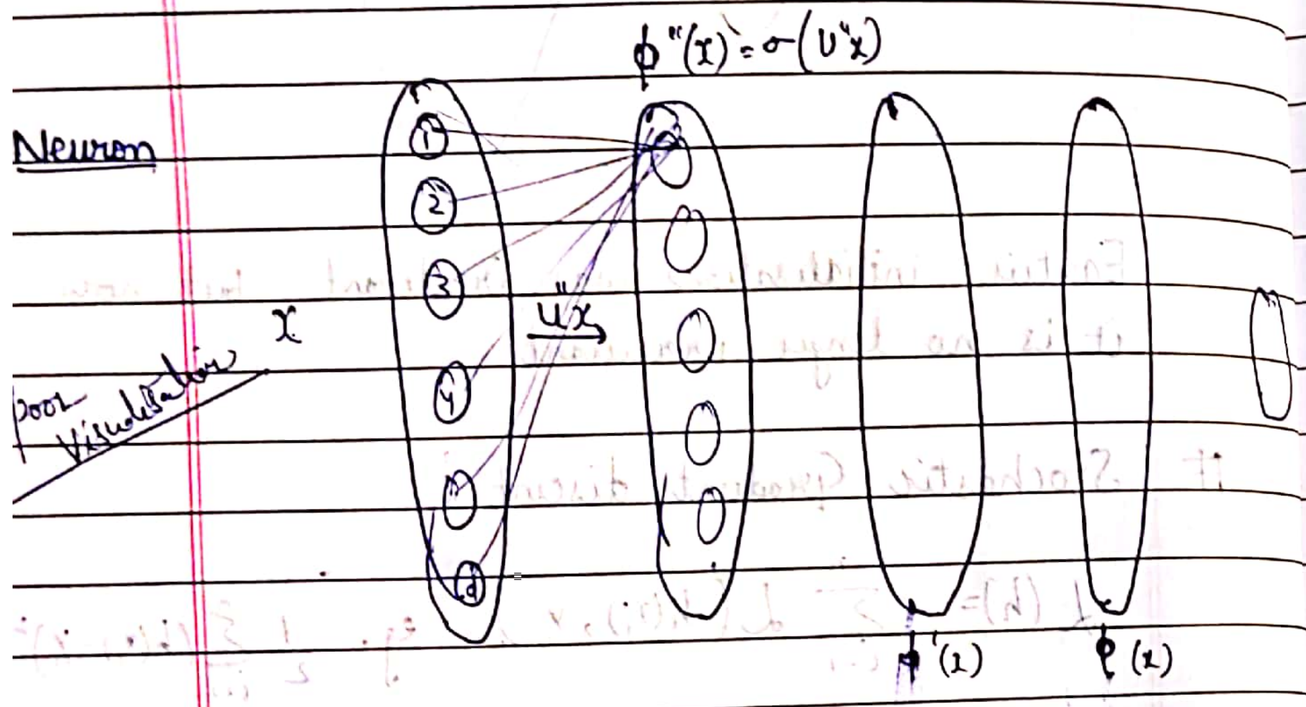
$$\frac{\partial \mathcal{L}}{\partial U} \simeq \frac{\partial \mathcal{L}}{\partial U}(h(x_i), y_i)$$



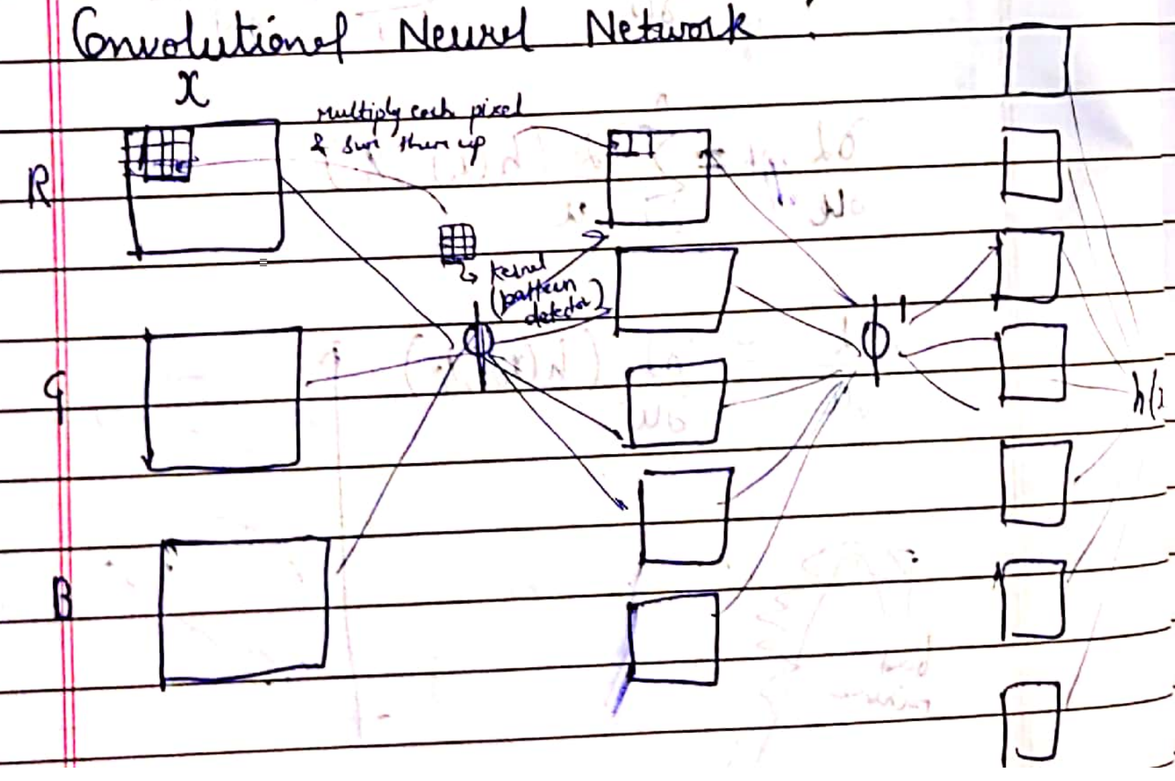


SGD is too noisy, It is too imprecise to hit small narrow (bad) local minima.

Batch Grad. descent : We take 10 points in one iteration.



Convolutional Neural Network :



Stanford University (CS 224N)

NATURAL LANGAUGE PROCESSING
(with deep learning)

Word Embeddings

> Word representation in terms of Vector.

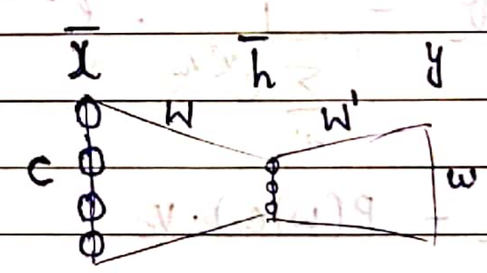
Frequency based	Prediction based
- Count Vector	- CBOW
- TF-IDF Vector	- Skip-gram
- Co-Occurence Vector	

→ Prediction based embeddings (Word2Vec)

CBOW — — $\overset{w}{O}$ — — $P(w|c)$

Skip-Gram — $\underset{c}{O}$ O — O O $P(c|w)$

CBOW (Continous Bag of words)



$$\bar{x} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \in \mathbb{R}^d$$

one-hot encoding of a word in Vocabulary

$$\bar{h} = W^T \bar{x} = \bar{v}_c$$

$$\bar{u} = W'^T \bar{h}$$

$$u_i = \bar{v}_w^T \bar{v}_c$$

$$y_i = \sigma(u_i) = \frac{e^{\bar{v}_c^T \bar{v}_w}}{\sum e^{\bar{v}_c^T \bar{v}_{w'}}}$$

$y_i = P(w|c)$

Using MLE ;

$$\theta = \{ \bar{v}_c, \bar{v}_w \}$$

$$L(\theta) = \prod_w P(w|c;\theta)$$

$$= \prod \frac{e^{\bar{v}_c^T \bar{v}_w}}{\sum_{w'} e^{\bar{v}_c^T \bar{v}_{w'}}}$$

$$l(\theta) = \sum_{w'} \bar{v}_c^T \bar{v}_w - \log \sum_{w'} e^{v_c^T v_w}$$

$$\frac{\partial l}{\partial \bar{v}_w} = \bar{v}_c - \frac{1}{\sum_{w'} e^{v_c^T v_w}} \cdot e^{v_c^T v_w} \cdot v_c$$

$$= v_c - P(w|c) \cdot v_c$$

$$= v_c \left[1 - P(w|c) \right]$$

Apply Gradient descent.

$$v_w^{new} \leftarrow v_w^{old} - \alpha . v_c (1 - P(w|c))$$

If corpus is very large, $P(w|c)$ is expensive to compute in each iteration.

Skip-gram with negative sampling

Unsupervised learning $\longrightarrow$ Supervised Classification problem

D : Set of all pairs (w, c) that are in the text.

D' : Set of all pairs (w, c) that are not in the text.

Z : 1 ; if pair is in set D
0 ; if pair is in set D'

$P(Z=1 | (w,c))$ is the probability that $(w,c) \in D$

$P(Z=0 | w, c)$ is the prob. that $(w,c) \in D'$

$$P(z|x) = \frac{P(x|z) \cdot P(z)}{P(x)}$$

Logistic regression :

$$P(Z=1 | (w,c)) = \frac{1}{1 + e^{v_c^T v_w}}$$

$$P(Z=0 | (w,c)) = 1 - \frac{1}{1 + e^{v_c^T v_w}}$$

$$= \frac{1}{1 + e^{-v_c^T v_w}}$$

$$P(z \mid (w,c); \theta) = \left(\frac{1}{1+e^{v_c^T v_w}}\right)^{z} \left(\frac{1}{1+e^{-v_c^T v_w}}\right)^{1-z}$$

$$\theta = \{ v_c, v_w \}$$

MLE

$$L(\theta) = \prod_{(w,c) \in D \cup D'} \left(\frac{1}{1+e^{v_c^T v_w}}\right)^{z} \left(\frac{1}{1+e^{-v_c^T v_w}}\right)^{1-z}$$

$$l(\theta) = \sum_{(w,c) \in D \cup D'} z \log\left(\frac{1}{1+e^{v_c^T v_w}}\right) + (1-z) \log\left(\frac{1}{1+e^{-v_c^T v_w}}\right)$$

$$= \sum_{(w,c) \in D} \log\left(\frac{1}{1+e^{v_c^T v_w}}\right) + \sum_{(w,c) \in D'} \log\left(\frac{1}{1+e^{-v_c^T v_w}}\right)$$

$\frac{\partial l}{\partial v_c}$ For one pair ;

$$\sigma(v_c^T v_w) + \sum_{i=1}^{k} \sigma(-v_c^T v_w) P_n(w)$$

we will look more than 1 pair (k) which are in D'.
Because, it's imbalance classification problem. Few points in z=0 & large points in z=1 class.

$$P_n(w) = \frac{\# w}{\sum_{w'} \# w'}$$

Select w & c randomly, & assure they will not be in corpus.

$$P_n = U(s)^{3/4} / Z$$

King $\rightarrow (v_c)_{king}$, King $\rightarrow (v_w)_{king}$

$$v_{king} = (v_c)_{king} + (v_w)_{king}$$