INTRODUCTION

☆ AI

- Artificial Intelligence (man-made thinking power) is a branch of computer science by which we can create intelligent machines that can behave like humans, think like humans and able to make decisions.
- Multimodal AI systems can handle multiple types of inputs and outputs.

— Goals

- Replicate human intelligence to solve knowledge-intensive tasks
- An intelligent connection of perception and action.
- Creating a system which can exhibit intelligent behaviour, learn new things itself, demonstrate, explain and advise user.

— Risks

- Potential to displace large number of workers due to capability of automation of human-bound tasks
- Bias and discrimination, since it relies on data to learn and make decisions
- Could create autonomous weapons that ~~can~~ could select and target w/o human involvement, obviously a threat to humans.

— Benefits

- Assist in making wise decisions, make predictions about future using past data.
- Automate repetitive tasks to focus on important things

— Types (based on Technology)

① Artificial Narrow Intelligence (weak, narrow AI)
(goal-oriented, designed to perform singular tasks)

Ex: IBM Watson, Siri, self-driving cars

② Artificial General Intelligence (strong, deep AI)
(ability to learn and apply intelligence to solve any problem)

③ Artificial Super Intelligence (hypothetical AI)
(machines become self-aware and surpass capacity of human intelligence and ability)

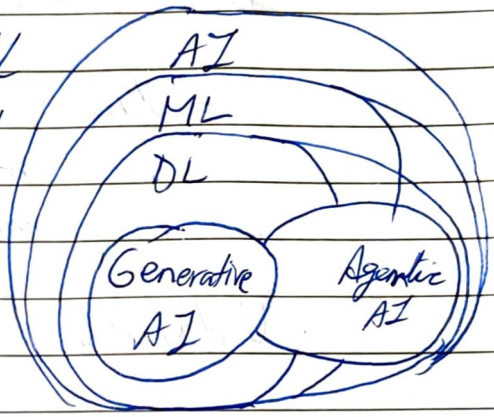
☆ Machine Learning

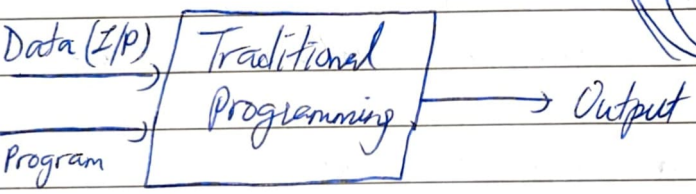
- Branch of AI concerned with construction and study of systems that learn from past data, w/o being explicitly programmed
- Computer program that learns from experience E wrt some class of task T and performance measure P, if performance at task T measured by P improves with experience E.

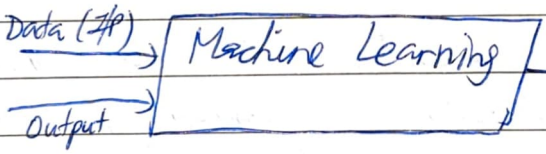
__/__/__

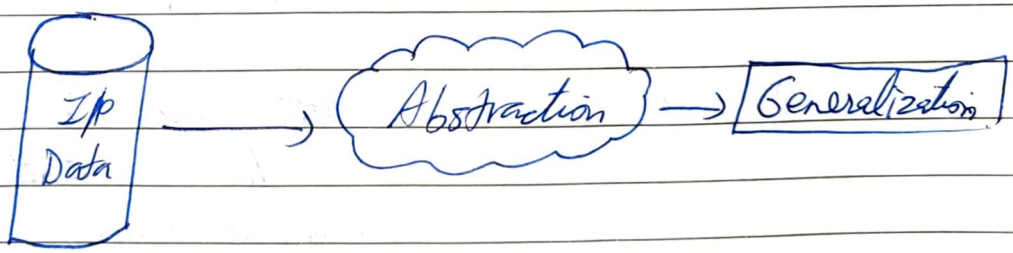
Ex: Handwriting recognition learning problem
T → recognizing and classifying handwritten words within images.
P → Percent of words correctly classified
E → dataset of handwritten words with given classifications

- Deep learning is a subset of ML in which artificial neural networks adapt and learn from vast amounts of data. (mimics the brain's neuron functionality)









- Data Input: Model fed with past data, which acts as the foundation for learning patterns & relationships.
- Abstraction: Processes input data using algorithms to transform into meaningful representation (identifies underlying patterns / features)
- Generalization: Uses abstracted patterns to make predictions on unseen test data.

__/__/____

- Challenges

- Data Quality (biased data can lead to unfair or discriminatory outcomes)
- Data Quantity (insufficient data for effective training)
- Overfitting (model performs well on training data but not on unseen test data, since it learns noise that do not generalize well)
- Underfitting (model is too simple to capture underlying patterns in data)
- Computational resources (hardware)
- Optimal algorithm selection (each have their own strengths and weaknesses, performance varies based on nature of data)
- Security concerns (vulnerable to cyber attacks which can mislead model)
- Lack of Standardization (vary based on model architectures, evaluation metrics, datasets)
- Model interpretability (understanding decision-making)

✯ Multivariate Calculus

– Scalar Function

- Maps one or more variables to a single real number.
- Single-variable: $f(x) = x^2 + 3$
  $f(2) = 7$
- Multi-variable: $f(x, y) = x^2 + y^2$
  $f(2, 3) = 13$

– Vector-Valued Function

- Maps one or more variables to a vector of values.

$$F(x, y) = \begin{bmatrix} x \\ y \\ x^2 + y^2 \end{bmatrix}$$

– Gradient (∇f)

- Scalar function that contains all partial derivatives.

$$\nabla f = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n} \right]^T$$

- Points in direction of steepest ascent.
- In gradient descent optimization algo, gradient helps find direction to adjust model parameters (weights) to minimize loss function.
- Useful for training NN and linear regression models

Ex:- $f(x,y) = x^2 + y^2$

$$\nabla f(x,y) = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right] = [2x \;\; 2y]^T$$

At point (1,2), $\nabla f(1,2) = [2 \;\; 4]^T$

Function increases fastest at (2,4)

— Hessian

- Square matrix of second-order partial derivatives of scalar-valued function.

$$H = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

- Captures curvature information of a function.

Ex:- $f(x,y) = x^2 + xy + y^2$

$$\frac{\partial f}{\partial x} = 2x + y, \quad \frac{\partial f}{\partial y} = x + 2y$$

$$\frac{\partial^2 f}{\partial x^2} = 2, \quad \frac{\partial^2 f}{\partial y^2} = 2, \quad \frac{\partial^2 f}{\partial x \partial y} = 1$$

$$H_f(x,y) = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix} = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

Jacobian

- Matrix of all first-order partial derivatives of vector-valued function.

Eqn

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

- Gives linear transformation of the function around a point.

Eqn

$$f(x,y) = \begin{bmatrix} f_1(x,y) = x+y \\ f_2(x,y) = x^2+y^2 \end{bmatrix}$$

$$\frac{\partial f_1}{\partial x} = 1, \quad \frac{\partial f_1}{\partial y} = 1, \quad \frac{\partial f_2}{\partial x} = 2x, \quad \frac{\partial f_2}{\partial y} = 2y$$

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 2x & 2y \end{bmatrix}$$

- First row tells how first output $f_1 = x+y$ changes with x and y, same for second row.

Determinant

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}, \quad \det(A) = ad - bc$$

If $\det(A) = 0$, A is not invertible

if $\det(A) \neq 0$, A is invertible

Eigenvalues and Eigenvectors

- For matrix A, $A\vec{v} = \lambda\vec{v}$ $\quad |A - \lambda I| = 0$

(λ: eigen value, $\vec{v}$: eigen vector)

Ex: $A = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix} \quad \vec{v} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

$$A\vec{v} = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \end{bmatrix} = 2\begin{bmatrix} 1 \\ 0 \end{bmatrix} = 2\vec{v}$$

Here, $\lambda = 2, \quad \vec{v} = [1 \;\; 0]^T$

Singular Value Decomposition (SVD)

- Factorize a matrix A into:

$$A = U\Sigma V^T$$

where U = left singular vectors
Σ = diagonal matrix with singular values
V = right singular vectors

- Used in noise reduction, image compression, dimensionality reduction (Principal Component Analysis)

Ex: $A = \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix}$

U = eigen vectors of AA^T
V = eigen vectors of A^TA

$$AA^T = \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix}\begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix} = \begin{bmatrix} 10 & 6 \\ 6 & 10 \end{bmatrix}$$

$$A^TA = \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix}\begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix} = \begin{bmatrix} 10 & 6 \\ 6 & 10 \end{bmatrix}$$

Eigen values of AA^T and $A^TA = |(AA^T) - \lambda I| = 0$

Note I = identity matrix $= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

$$\begin{vmatrix} 10-\lambda & 6 \\ 6 & 10-\lambda \end{vmatrix} = (10-\lambda)^2 - 36 = 0$$

$$100 + \lambda^2 - 20\lambda = 36$$

$$\lambda^2 - 20\lambda + 64 = 0$$

$$\lambda^2 - 16\lambda - 4\lambda + 64 = 0$$

$$\lambda(\lambda - 16) - 4(\lambda - 16) = 0$$

$$\lambda = 4, 16$$

$\lambda_1 = 16$, $\lambda_2 = 4$

$\sigma_1 = \sqrt{\lambda_1} = 4$, $\sigma_2 = \sqrt{\lambda_2} = 2$ (singular values)

$$\Sigma = \begin{bmatrix} 4 & 0 \\ 0 & 2 \end{bmatrix}$$

For $\lambda = 16$, $(A^TA - 16I)v = 0$

$$\left(\begin{bmatrix} 10 & 6 \\ 6 & 10 \end{bmatrix} - \begin{bmatrix} 16 & 0 \\ 0 & 16 \end{bmatrix}\right) v = 0$$

$\left(v = \begin{bmatrix} x \\ y \end{bmatrix}\right)$

$$\begin{bmatrix} -6 & 6 \\ 6 & -6 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = 0$$

$$-6x + 6y = 0$$

$$\therefore x = y$$

$\therefore$ Eigen vectors $= \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

For $\lambda = 4$, $\left(\begin{bmatrix} 10 & 6 \\ 6 & 10 \end{bmatrix} - \begin{bmatrix} 4 & 0 \\ 0 & 4 \end{bmatrix}\right)\begin{bmatrix} x \\ y \end{bmatrix} = 0$

$$\begin{bmatrix} 6 & 6 \\ 6 & 6 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = 0$$

$$-x = y$$

Eigen vectors $= \begin{bmatrix} 1 \\ -1 \end{bmatrix}$

After normalizing, denominator $= \sqrt{1^2 + 1^2} = \sqrt{2}$

For $\lambda = 16$, $\vec{v} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$, for $\lambda = 4$, $\vec{v} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{-1}{\sqrt{2}} \end{bmatrix}$

__/__/__

(descending order)

U = combine both eigen vectors for $\lambda = 6$ and $\lambda = 4$

$$U = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{bmatrix} \qquad V = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{bmatrix} \leftarrow (AA^T = A^TA)$$

To verify, $U\Sigma V^T = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} \sqrt{6} & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{bmatrix}$

$$= \begin{bmatrix} 2\sqrt{2} & \sqrt{2} \\ 2\sqrt{2} & -\sqrt{2} \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{bmatrix}$$

$$= \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix} = A$$

Ex: $A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ -1 & 1 \end{bmatrix}$ $\quad AA^T = \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -1 \\ 1 & 1 & 1 \end{bmatrix}$

$$= \begin{bmatrix} 2 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix}$$

$$A^TA = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix}$$

To find eigen values of A^TA.

$$|(A^TA) - \lambda I| = 0$$

$$\begin{vmatrix} 2-\lambda & 0 \\ 0 & 3-\lambda \end{vmatrix} = \lambda^2 - 5\lambda + 6 = 0$$

$$\lambda = 2, 3$$

$$\sigma_1 = \sqrt{3}, \quad \sigma_2 = \sqrt{2}$$

$$(A^TA - 2I)v = 0$$

$$\left(\begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix} - \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}\right)\begin{bmatrix} x \\ y \end{bmatrix} = 0$$

$$\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = 0 \qquad v_2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$y = 0, \quad x = 1$$

$$(A^TA - 3I)v = 0$$

$$\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = 0 \qquad v_1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$-x = 0, \quad x = 0, \quad y = 1 \qquad \text{(Normalized)}$$

$$V = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Since matrix A has dimension 3×2.

$$\Sigma = \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} \sqrt{3} & 0 \\ 0 & \sqrt{2} \\ 0 & 0 \end{bmatrix}$$

To find eigen vectors of AA^T

$$(AA^T - 3I)v = 0$$

$$\begin{bmatrix} 2-3 & 1 & 0 \\ 1 & 1-3 & 1 \\ 0 & 1 & 2-3 \end{bmatrix}\begin{bmatrix} x \\ y \\ z \end{bmatrix} = 0 \qquad u_1 = \frac{1}{\sqrt{3}}\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

$$-x + y = 0, \quad x = y$$

$$x - 2y + z = 0, \quad x + z = 2y$$

$$x = z$$

$$y = z$$

$$(AA^T - 2I)v = 0$$

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -1 & 1 \\ 0 & 1 & 0 \end{bmatrix}\begin{bmatrix} x \\ y \\ z \end{bmatrix} = 0 \qquad u_2 = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}$$

$$y = 0$$

$$x + z = y = 0$$

$$x = -z$$

Since $A = (3\times 2) = U\Sigma V^T = (3\times 3)\times(3\times 2)\times(2\times 2)$
↓ extra vects.

For $\lambda = 0$

$$(AA^T)v = \lambda v$$

$$\begin{bmatrix} 2 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix}\begin{bmatrix} x \\ y \\ z \end{bmatrix} = 0$$

$$2x + y = 0, \quad y = -2x$$

~~$x + 2z =$~~

$$y + 2z = 0, \quad y = -2z$$

$$x + y + z = 0$$

$$u_3 = \frac{1}{\sqrt{6}}\begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix}$$

$$\therefore U = \begin{bmatrix} \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{6}} \\ \frac{1}{\sqrt{3}} & 0 & \frac{-2}{\sqrt{6}} \\ \frac{1}{\sqrt{3}} & \frac{-1}{\sqrt{2}} & \frac{1}{\sqrt{6}} \end{bmatrix} \quad \Sigma = \begin{bmatrix} \sqrt{3} & 0 \\ 0 & \sqrt{2} \\ 0 & 0 \end{bmatrix}$$

$$V = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$\therefore \text{To verify, } A = \begin{bmatrix} \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{6}} \\ \frac{1}{\sqrt{3}} & 0 & \frac{-2}{\sqrt{6}} \\ \frac{1}{\sqrt{3}} & \frac{-1}{\sqrt{2}} & \frac{1}{\sqrt{6}} \end{bmatrix}\begin{bmatrix} \sqrt{3} & 0 \\ 0 & \sqrt{2} \\ 0 & 0 \end{bmatrix}\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ -1 & 1 \end{bmatrix}$$

Probability

- A random variable assigns a numerical value to the outcome of an uncertain event.
- Discrete RV takes specific / separate value each having distinct probability.
- Continuous RV can take any value in a range and probabilities are computed over intervals (area under curve)
- Probability Mass Function (PMF) (Discrete) sums to 1. $P(X = x)$
- Probability Density Function (PDF) (Cont.) integrates to 1 $p(x)$
- Cumulative distribution function (cdf):
  $F(x) = P(X \leq x)$: monotone
  - $p(x) = \frac{d}{dx} F(x)$ (if differentiable)
- $P(A|B) = \frac{P(A \cap B)}{P(B)}$ (conditional probability)
  (B: given)
- $P(A, B) = P(A|B)P(B) = P(B|A)P(A)$ (Product rule)
- Bayes Theorem: $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$

  $\text{Posterior} = \frac{\text{Likelihood} \times \text{Prior}}{\text{Evidence}}$

__/__/__

Ex: Bag A : 3R + 1B
Bag B : 1R + 3B

$P(\text{Bag A} / \text{Red ball}) = ?$

$P(\text{Bag A}) = P(\text{Bag B}) = \frac{1}{2}$

$P(\text{Red} / \text{Bag A}) = \frac{3}{4}, \quad P(\text{Red} / \text{Bag B}) = \frac{1}{4}$

$P(\text{Red}) = P(\text{Red} / \text{Bag A}) \cdot P(\text{Bag A}) + P(\text{Red} / \text{Bag B}) P(\text{Bag B})$

$= \frac{3}{4} \times \frac{1}{2} + \frac{1}{4} \times \frac{1}{2}$

$= \frac{1}{2}$

$P(\text{Bag A} / \text{Red ball}) = \frac{\frac{3}{4} \times \frac{1}{2}}{1/2} = \frac{3}{4}$

* Types of ML

– Supervised Learning

- Machines are trained using well "labelled" training data using which model predicts the output.
- Here labelled data mean data already tagged with correct output.
- Main aim is to find a mapping function that maps input variable X to output variable Y.

Ex:- Fraud detection, spam filtering, image classification

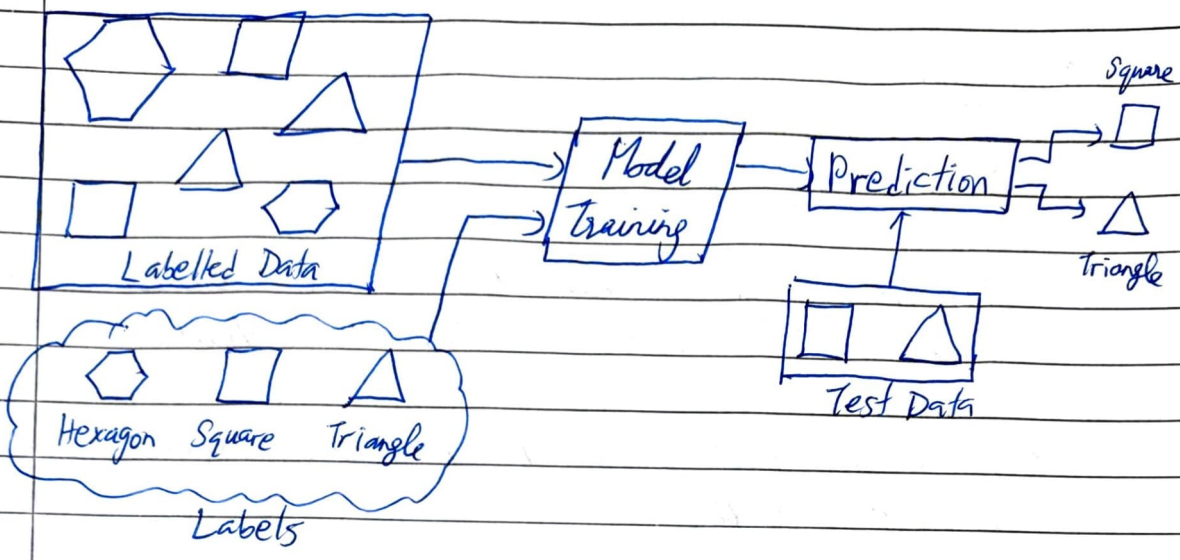
- Classification: Algorithms used when output variable is categorical

Ex:- Random Forest, Logistic Regression, Decision Trees, Support Vector Machines (SVM), KNN

- Regression: Algorithms used for prediction of continuous variables based on relationship between input and output variable.

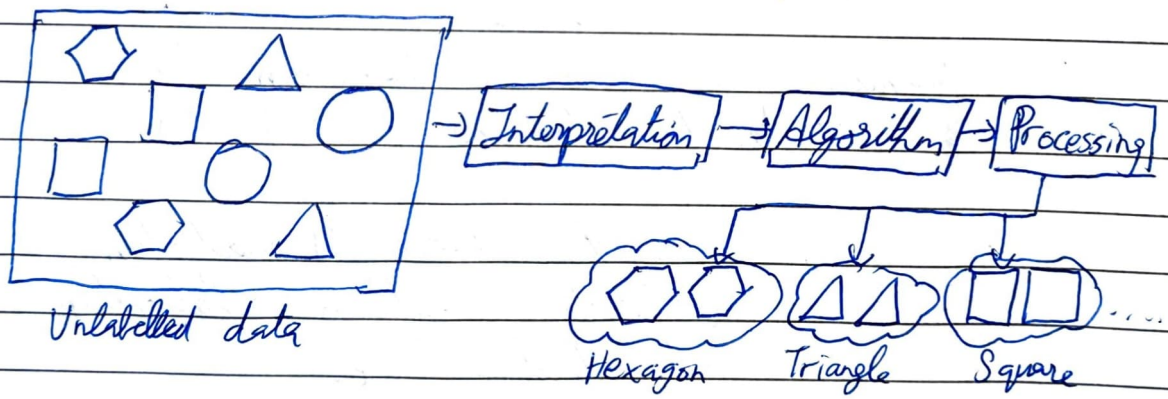
Ex:- Linear Regression, Non-linear regression, Regression Trees, Polynomial Regression.

- Advantages:
  - Model can predict based on prior experience
  - We have exact idea about classes of objects
  - Helps solve real world problems
- Disadvantages:
  - Not suitable for complex tasks
  - High computation time
  - Cannot predict correct if test ≠ train data type.



— Unsupervised Learning

- Model is trained using unlabeled data (w/o target variable), to find the underlying structure of dataset, group that data according to similarities and represent that data in compressed format.



- Clustering: Groups similar data points together

Ex: K-Mean, DBSCAN, Hierachial Clustering (used for customer segmentation by behaviour, document grouping)

- Association: Discover relationships between items that frequently occur together.

Ex: Apriori, FP growth.

Ex: If a customer buys bread, they are also likely to buy butter / jam with it.

- Advantages:
  - Can be used for more complex tasks
  - It is easier to get unlabelled data than labelled data.
- Disadvantages:
  - May be less accurate
  - More difficult

— Semi-Supervised Learning (Google Photos)

- Combination of Supervised & Unsupervised Learning algorithms, that can deal with partially labelled data
- Can be helpful since labelling data is costly and time-consuming

— Reinforcement Learning (Robots) (Feedback-based)

- Action is taken by observing the environment and getting rewarded in return.
- Agent (learning system) learns by itself, which is the best policy to get most reward over time.
- For good action, agent gets positive feedback and for each bad action, it recieves negative feedback/penalty

★ Model Selection

- Process of choosing the best model based on performance on validation dataset

① Selecting the right algorithm
② Tuning hyperparameter
③ Balancing bias and variance.

- Generalization is the ability of a model to predict well on unseen data. It ensures that the model captures the underlying patterns rather than memorizing the training data.

- Technique to improve generalization include L1 and L2 regularization, dropout (NN), early stopping and data augmentation.

☆ VC Dimension

- Measure of how powerful a model is in terms of classification, based on the hypothesis

- If the model can seperate every possible labelling of those points, it can be shattered.

- Largest no. of points that can be shattered → VC dimension

Eg: For linear classifiers in 2D (straight lines)

1 point → label as (+) or (−)
line can seperate it from empty space.

2 points → possible labels = $2^2 = 4$
(+, +), (+, −), (−, +), (−, −)
line can seperate them.

3 points → possible labels = $2^3 = 8$
every label can seperated by some line
if not collinear.

4 points → possible labellings = $2^4 = 16$

For some cases like $\begin{bmatrix} + & - \\ - & + \end{bmatrix}$

no single can seperate (+) from (−), hence ✗

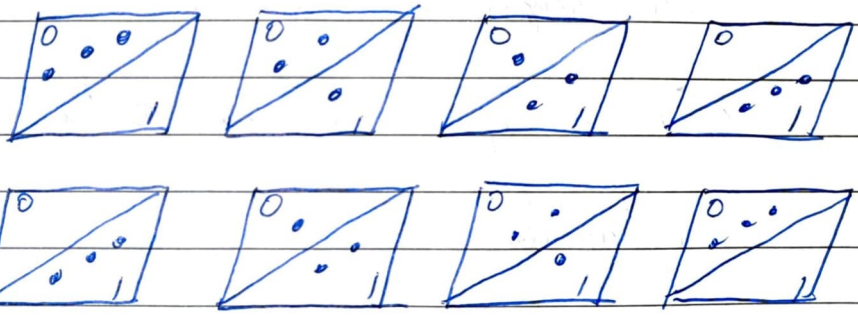
∴ VC dimension for linear classifiers in 2D = 3.

—x—

For 1 point,

For 2 points,

For 3 points



—x—

☆ Training, Validation and Test Data Splits

— Training Set

- Used to fit / train the model, to learn the parameters of the model
- In supervised, features + labels
  In unsupervised, features only.
- 70% of the original dataset usually.

— Testing Set

- Once we have the model trained with training set and hyperparameters tuned using validation set, we need to test the model for generalization.
- Same characteristics as train set, and large enough to yield statistically significant results. (10%)
- Test accuracies help to ensure model is not overfitting / underfitting.

— Validation Set

- Used to tune hyperparameters and prevent overfitting. Helps us choose which model gives best result. (20% of original data usually)

— Validation techniques

- Methods used to assess performance of ML model on data it has not seen during training.

① Holdout Method

- Split data in train, ~~validation~~ validation and test set, then train the model on the training set, then tune the model on validation set (optional) then evaluate the model on the test set to see how well it generalizes on new data.
- May not be suitable for small datasets (unstable results)

② K-fold Cross Validation

- Shuffle dataset randomly.
- Split the dataset into k-groups → $\frac{1}{k}$ of data
- For each group; take one group as test dataset and $\frac{k-1}{k}$ → remaining as training set and fit into model and evaluate
- Repeat k-times, each time taking unique test dataset

	1	2	3	4	5
	1	2	3	5	4
Train →	1	2	4	5	3
	1	3	4	5	2 ← Test
	2	3	4	5	1

(LOOCV)

③ Stratified K-fold CV

- Each fold maintains the same class-distribution as

the original dataset. Useful when dealing with imbalanced datasets. (prevents bias)

Ex: $X = [A, B, C, D, E, F, G, H, I, J]$
$Y = [0, 0, 0, 1, 0, 1, 1, 1, 1, 1]$

Class 0: 4 samples
Class 1: 6 samples

Fold 1: $[A, B, D, F, G, J] \rightarrow [C, E, H, I, J]$
$[C, E, H, I, J] \rightarrow [A, B, D, G, F]$
(2:3) ratio

(4) Leave-One-Out CV

- k = no. of instances in the dataset
Each iteration, one instance used for testing while $(k-1)$ used for training. (changes a lot)
- Provides high-variance estimate but computationally expensive.

(5) Shuffle-Split CV

- Used when k-fold CV is impractical.
- Randomly shuffle the dataset then split (done multiple times, average performance calculated)

(6) Nested CV

- Outer Loop: Split dataset in distinct K-folds (outer test set)

and remaining (k-1) folds passed into inner loop.

- Inner loop: On (k-1) inner folds, perform another cross-validation and try different hyperparameters. Choose best combination with best average performance.

- Evaluate: Train the model with the selected best hyperparameters (from inner loop) on the entire (k-1) folds and evaluate on outer test fold

- Repeat for each outer fold and average out results.

⑦ Holdout with Multiple Validation Sets

- Split validation set into multiple subsets and train model on each seperately.

★ Regression Evaluation Metrics

- $f: X \to Y$; $f(x) = y$
- Data is available as samples (x, y)
  Our objective is to find model $h(x)$ that explains the underlying data, $h(x) \simeq y \quad \forall (x, y)$
- Learn $h(x, w) \simeq y$ having fixed parameter/weight w_j of weight vector w.

— **Mean Squared Error (MSE)**

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

where $N \to$ no. of samples
$y_i \to$ actual value
$\hat{y}_i \to$ predicted value

— **Root Mean Squared Error (RMSE)**

$$RMSE = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2}$$

- Same dimensions as predicted values (more interpretable)

— **Sum-of-Error Squares**

$$\text{Sum-of-Error Squares} = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

__/__/____

Note Expected MSE = $E\left[\sum_{i=1}^{N}(y_i - \hat{y}_i)^2\right]$

Used in probabilistic settings

– Mean Absolute Error (MAE)

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

Note Criterion	MAE (Error ↗)	MSE (Error ↗)
• Penalty on errors	• Linear penalty	• Quadratic penalty
• Sensitivity to outliers	• Not sensitive (robust)	• Very sensitive (outliers dominate)
• Error variability	• Not captured	• Captures error variance
• Interpretability	• Equal weight to all errors	• Large errors have greater influence

☆ Classification Evaluation

• Misclassification Error = $\frac{\text{No. of data points where } (y_i - \hat{y}_i) \neq 0 \text{ (zero error)}}{N}$

• Suitable when class distribution is balanced

• Confusion Matrix

Actual (observation) \ Hypothesis (prediction)	T (+)	F (–)
T (+)	TP	FN
F (–)	FP	TN

- Handles imbalanced datasets, key in real-world decision-making.

TP → Correctly predicted positives
FN → Wrongly predicted negatives (Type-II error)
FP → Wrongly predicted positives (Type-I error)
TN → Correctly predicted negatives.

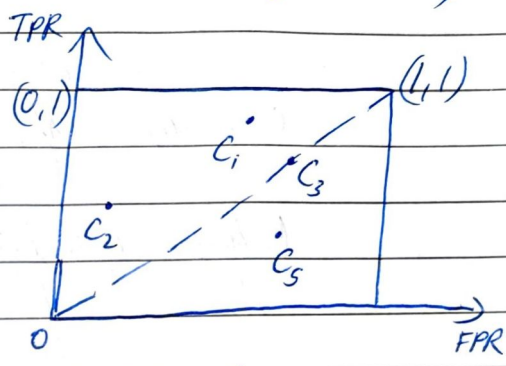
- Accuracy (Success rate) = $\frac{TP + TN}{TP + TN + FP + FN}$
- Misclassification (Failure rate) = $\frac{FP + FN}{TP + TN + FP + FN}$
- Precision = $\frac{TP}{TP + FP}$ (Of all predicted positives, how many were correct?)
- Recall / TPR = $\frac{TP}{TP + FN}$ (Of all actual positives, how many were correct?)
(Sensitivity)
(tells how sensitive our model is upon detection of abnormal event)
- Specificity / TNR = $\frac{TN}{FP + TN}$ (tells how specific our model is in detecting an abnormal event)
- FPR = 1 − Specificity = $\frac{FP}{FP + TN}$ (negatives incorrectly classified out of all negatives)

Note: Good decision-making model would be having high sensitivity and high specificity.

- $F1\ Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$

(Harmonic mean of Precision and Recall)
(F1 = 1 → perfect precision and recall)
(F1 = 0 → either is zero)

— **ROC Curves** (TPR - FPR graph) (comparing classifiers)



- (0,1) → Perfect classifier
- (0,0) → no positives or negatives detected
(1,1) → classifies everything as positive.

C_3 → (on diagonal) random guessing / C_2 → cautious but not comprehensive
C_5 → worse than random
C_1 → eager to classify as positive

— **ROC Curves for Evaluating Classifiers**

- A single classifier yields one confusion matrix for one ROC point.
- To form an ROC curve, vary the decision threshold (minimum probability to classify as class 1), each setting a new point in the ROC space
- In information retrieval (IR) procedures, precision-recall curves are preferred.

Area Under Curve (AUC)

- Quantifies overall ability of model to distinguish between classes.

- AUC = 1 (perfect classifier)
  AUC = 0.5 (random guess)
  AUC < 0.5 (worse than random)

- While accuracy depends on specific threshold, AUC evaluates performance over all thresholds. (robust)

Ex:

Patient	True Label	Predicted Score	T=1	T=0.9	T=0.8
A	1	0.95	0	1	1
B	1	0.85	0	0	1
C	1	0.60	0	0	0
D	0	0.70	0	0	0
E	0	0.30	0	0	0
F	0	0.10	0	0	0

For $T=1$, $TP=0$, $FP=0$, $FN=3$, $TN=3$

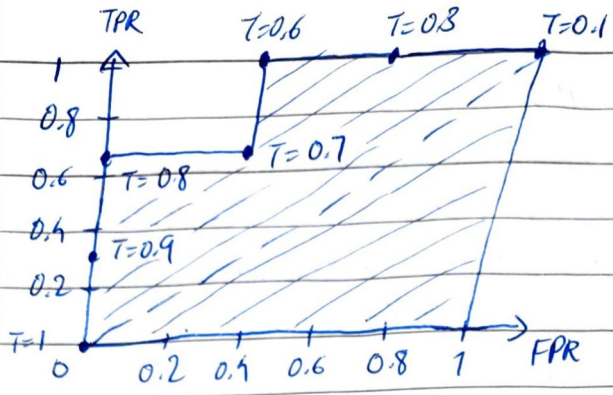
$$TPR = \frac{TP}{TP+FN} = \frac{0}{0+3} = 0, \quad FPR = \frac{FP}{FP+TN} = 0$$

$\therefore$ $T=1 \Rightarrow (0,0)$ on ROC curve

For $T=0.9$, $TP=1$, $FP=0$, $FN=2$, $TN=3$

$$TPR = \frac{1}{1+2} \approx 0.33, \quad FPR = \frac{0}{0+3} = 0$$

$T=0.9 \Rightarrow (0.33, 0)$



⟶ Random classifier

⟶ AUC Area

⟶ ROC Curve.

AUC = 0.8911 unid^2

—x—

Types of Learning

— Eager Learning (Model-based)

- Learns a model in advance from training data and generalizes before recieving queries.
- May overfit / underfit if model assumptions are wrong

— Lazy Learning (Memory-based)

- Stores training data, does not learn a model
  Avoids generalization until query time (when asked)
  Make predictions by comparing query with stored instances
  Flexible and simple, but sensitive to irrelevant features

Feature	Eager-Learning	Lazy Learning
Model Building	Yes	No
Training Time	High	None
Prediction Time	Fast	Slow
Memory Usage	Low	High
Example	SVM, Decision Trees	k-NN

☆ K - Nearest Neighbours

- Non-parametric, instance-based classification method that does not assume any underlying data distribution but rather relies on similarity (distance-metric) between instances for classification.

① To classify new sample x, compute distances from x to all training points
② Identify the k-nearest neighbours. (k → odd to avoid ties)
③ Assign the class that is most frequent among them.

- Use validation set to find the best k

- Small k → sensitive to noise (high variance)
  Large k → more stable but less local
  (considers more no. of neighbours that belong to different classes) (high bias)

- Some distance metrics include

— Euclidean (L2 Norm)

- Straight-line distance between 2 points.

$$d(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

- Works well with continuous and equally scaled values

Ex: $A(1,2)$, $B(4,6)$

$$d = \sqrt{(4-1)^2 + (6-2)^2} = \sqrt{9+16} = \sqrt{25} = 5$$

Manhattan Distance (L1 Norm)

- Sum of absolute differences along each dimension (City - Block distance)

$$d(x,y) = \sum_{i=1}^{n} |x_i - y_i|$$

- Useful when movements are constrained to grid paths like in robotics or city-planning.
- Less sensitive to outliers.

Ex: $A(1,2)$, $B(4,6)$, $d = |4-1| + |6-2| = 7$

Minkowski Distance

- Generalized form of previous two.

$$d(x,y) = \left(\sum_{i=1}^{N} |x_i - y_i|^p \right)^{\frac{1}{p}}$$

If $p = 1$ → Manhattan distance
$p = 2$ → Euclidean distance.

Ex:

Plant	Sunlight (hrs/day)	Soil Moisture	Type
A	6.0	32	Cactus
B	8.0	28	Cactus
C	4.0	68	Fern
D	3.2	82	Fern
E	3.8	72	Fern
F	6.8	22	Cactus
G	5.5	40	Cactus

For new plant $X(5, 60)$, $k = 3$

$$d_{AX} = \sqrt{(5-6)^2 + (60-32)^2} = \sqrt{1^2 + (28)^2} = \sqrt{785} \approx 28$$

$$d_{BX} = \sqrt{(5-8)^2 + (60-28)^2} = \sqrt{3^2 + 32^2} = 32.1$$

$$d_{CX} = \sqrt{(5-4)^2 + (60-68)^2} = \sqrt{1^2 + 8^2} = \sqrt{65} = 8.06$$

$$d_{DX} = \sqrt{(5-3.2)^2 + (60-82)^2} = 22.06$$

$$d_{EX} = \sqrt{(5-3.8)^2 + (60-72)^2} = 12.06$$

$$d_{FX} = \sqrt{(5-6.8)^2 + (60-22)^2} = 38$$

$$d_{GX} = \sqrt{(5-5.5)^2 + (60-40)^2} = 20$$

∴

	Plant	Distance to $X(5, 60)$	Type
→	C	8.06	Fern
→	E	12.06	Fern
→	G	20	Cactus
	D	22.06	Fern
	A	28	Cactus
	B	32.1	Cactus
	F	38	Cactus
	X	0	(Fern)

- **Advantages:**
  - Simple, intuitive
  - No training needed (lazy learner)
  - Non-parametric (no assumptions)
  - Supports incremental learning (data ++)
  - Naturally extensible to Multi-Class
  - Works well with sufficient data.

- Limitations:
  - Slow at prediction
  - High memory usage
  - Sensitive to irrelevant features
  - Dimensionality affects accuracy
  - Requires careful tuning of k
  - Imbalanced dataset → high bias

— Condensed Nearest Neighbours (CNN)

- Data reduction technique for (KNN)
- Objective is to keep only most informative samples to reduce memory/storage and prediction time w/o major loss in accuracy.
- T: full training set
  S: condensed subset
  T|S: set of all points in T not yet in S. (set diff)

① Initialize subset S with one random (first) sample from each class
② For each $x \in T|S$, classify x using 1-NN on S and if misclassified, add x to ~~T~~ S, and update 1-NN model.
③ Repeat until no new points are added.
④ Finally train the final-KNN model using final condensed subset S to classify new unseen data.

Exa- Using the previous example,
$T = \{A, B, C, D, E, F, G\}$
$S = \{A, C\}$ (first cat in and first dem)
$T|S = \{B, D, E, F, G\}$

For B(8, 28), $d_{BA} = 31.11$ ✓
$d_{BC} = 34.07$

For D(3, 85)
$d_{DA} = 26.11$
$d_{DC} = 28.01$ ✓

For E(6.5, 21)
$d_{EA} = 38.03$ ✓
$d_{EC} = 41.04$

For F(3.8, 64)
$d_{FA} = 5.25$
$d_{FC} = 2.16$ ✓

For G(5.2, 58)
$d_{GA} = 1.18$ ✓
$d_{GC} = 4.04$

Since no misclassifications, no additions → CNN stops.
$S = \{A, C\}$
Using 1-NN, for point X(5, 60)
$d_{XA} = 1.08$ ✓ (Cactus)
$d_{XC} = 2.04$

- Advantages:
  - Faster prediction, storage-efficient
  - Remove outliers and noise
  - Useful for large datasets where k-NN is slow
- Disadvantages:
  - Still sensitive to class imbalance
  - May lose accuracy
  - Order dependent during initialization

Linear Regression

- Linear Regression is a statistical technique used to find relationship between variables (features and labels)
- In linear algebra, $y = mx + b$ (m: slop, b: y-intercept)
- $\hat{y} = b + w_i x_i$ (prediction, bias, weight, feature values)
- During training, model calculates the optimal value of weight of each feature and bias to produce the best model

Ex: $\hat{y} = b + w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4$ (x_1: pounds, x_2: displacement, x_3: acceleration, x_4: horsepower)

(To predict car's mileage)

— Loss

- Numerical metric that describes how wrong the models predictions are.
- Goal of training a model is to minimize the loss (Distance between model's predictions and actual labels)
- Models trained with MAE will be much farther from outliers compared to MSE.

Cost Function

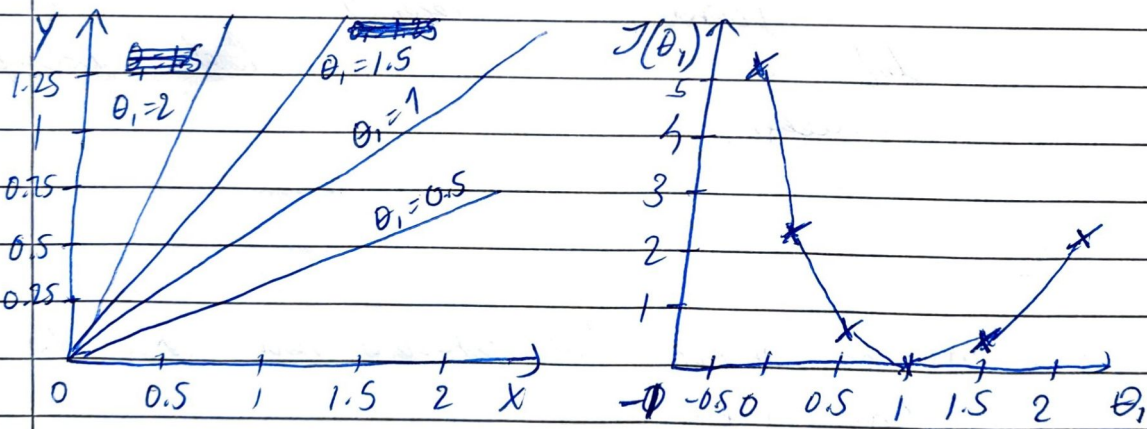
- While loss functions refer to errors for single data point predictions, cost functions aggregate these individual losses (average errors across entire training dataset)
- Measures how close predictions are to the actual y-values; average over all m-training instances.

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

($h_\theta(x^{(i)})$: prediction, $y^{(i)}$: actual)

for $\hat{y} = \theta_0 + \theta_1 x_1 = h_\theta(x)$

$h_\theta(x)$ (for fixed $\theta_0 = 0$) $\qquad$ $(J(\theta_1))$



Gradient Descent

- For any arbitrary function $J(\theta_0, \theta_1)$, we want to find minimum of it.

$$\theta_j \Rightarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$ (repeat until convergence)

- **Learning Rate (α)**: controls step size in weight updates during gradient descent.

- Too small α → very slow convergence, may get stuck at flat regions
  Very large α → overshoots the minimum, could possibly cause oscillations or divergence.
- Hence α should be small enough to maintain stability and large enough to converge quickly

Note: Convergence refers to the process where the algorithm successfully approaches and settles very close to minimum of cost function.
Divergence is when model tends to move further away from the minimum rather than towards it and jumps past it.

Convex Error Functions	Non-Convex Error Functions
• Any local minimum is also the global minimum	• Can have multiple local minima, maxima, and saddle points (slope = 0)
• Bowl-shaped in 2D, convex surface in higher-D	• Wavy / irregular
Ex: Linear regression with MSE, Cross entropy in logistic regression	Ex: Neural Networks, Decision-Tree splitting criteria.

Note: For $\hat{y} = b_1 x + b_0$

$$b_1 = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sum (x_i - \bar{x})^2}$$

where $b_0 = \bar{y} - b_1 \bar{x}$

$\bar{y}$ → mean of dependent variable | $\bar{x}$ → mean of independent variable

__/__/__

☆ Ridge Regression (L2 regularization)

— Multicollinearity

- Situation where predictor variables (independent variables) are highly correlated with each other.

Ex- $Height_{(cm)} = 2.54 \times Height_{(inch)}$

$Model = \theta_0 + \theta_1 \cdot Height_{(cm)} + \theta_2 \cdot Height_{(inch)}$

Since they contain the same information, regression model cannot uniquely assign weights.

Model will try to balance the coefficients, often giving high (+) value to one feature and high (−) to other.

Even though prediction still works, coefficients become unstable and meaningless. (Least squares estimates become unstable) (Causing overfitting / noise)

- Fixes :
  - Collect more diversified data
  - Increase sample size
  - Reduce no. of independent variables (dimensions)
  - Deploy a different model.

— Model

- Ridge Regression specially corrects for multicollinearity in regression analysis, by adding a penalty term to the loss function to shrink the coefficients, thus improving stability and generalization.

__/__/____

- **Ordinary Least Squares (OLS) Form:**

$$Y = \theta_0 + \sum_{j=1}^{n} \theta_j X_j$$

where θ_0 → intercept
θ_j → coefficients
X_j → predicts variables (independent variables)
Y → dependent variable

- **Matrix Form:** (Compact)

$$Y = X\theta$$

$$= \begin{bmatrix} 1 & x_1^{(1)} & \cdots & x_n^{(1)} \\ 1 & x_1^{(2)} & \cdots & \vdots \\ \vdots & \vdots & \cdots & \vdots \\ 1 & x_1^{(m)} & \cdots & x_n^{(m)} \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_m \end{bmatrix}$$

(Vector of predicted values)

(To estimate θ, $\theta = (X^T X)^{-1} X^T Y$)

- **Cost Function:** $J(\theta) = \sum_{i=1}^{m} (Y_i - \hat{Y}_i)^2$ ~~(OLS)~~

(Residual Sum of Squares (RSS))

- MSE is RSS normalized by no. of samples. (average error per observation)

- OLS is the method used to find best-fitting line in linear regression by minimizing RSS.

- **L2 Penalty**

- Represented by: $\lambda \sum_{j=1}^{n} \theta_j^2$ → λ → regularization parameter.

- Inserted at the end of RSS function, resulting in ridge regression estimator.

$$\text{New RSS} = \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2 + \lambda \sum \theta_j^2$$

$$= \text{~~RSS~~} + \text{~~λ~~}$$

(As $\lambda \uparrow$, high-value coeff shrinks at a greater rate than low-value coeff)

Logistic Regression

- Supervised learning algorithm used for classification problems
- Models the probability that an input belongs to a certain class using the logistic function (sigmoid) which maps any real-valued number into a value between 0 and 1.
- Decision made by applying a threshold to the predicted probability. If probability is above the threshold, the model assigns one class, if below, the other.

$$z = w_0 + w_1 x_1 + w_2 x_2 + \dots + w_n x_n$$

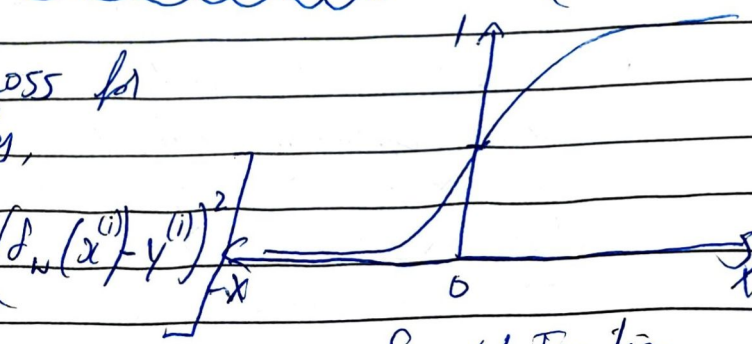
$$\hat{y} = f_w(x) = \sigma(z) = \frac{1}{1+e^{-z}} \quad \text{(Sigmoid)}$$

- **Odds:** $\text{Odds} = \frac{\hat{y}}{1-\hat{y}} \quad \in [0, \infty)$

Ex: $P(\text{win}) = (0.8)$, $P(\text{loss}) = 0.2$, $\text{Odds} = \frac{0.8}{0.2} = 4$

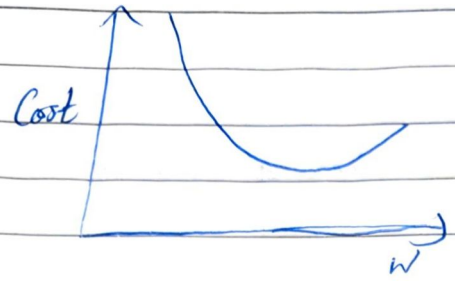
- **Logit:** $\log(\text{Odds}) = \log\left(\frac{\hat{y}}{1-\hat{y}}\right) \quad \in (-\infty, \infty)$
- Squared Error Loss for m training samples,

$$J(w) = \frac{1}{m} \sum_{i=1}^{m} \frac{1}{2} \left(f_w(x^{(i)}) - y^{(i)} \right)^2$$

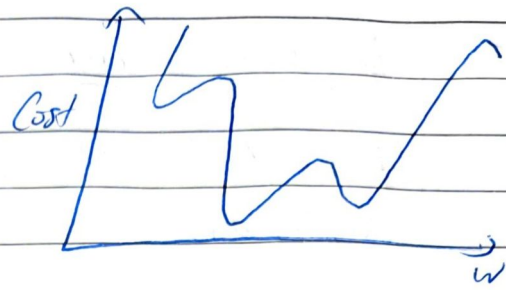


Sigmoid Function

__/__/____



(Linear Regression)
(convex curve)
(gradient descent always finds global minima)



(Logistic Regression)
(non-convex curve)
(multiple local minima)
(gradient-descent gets stuck)

- To avoid this instability issue, Cross-Entropy Loss is used. (aka Log Loss)

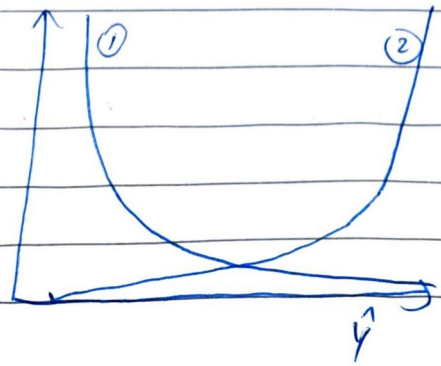
- It measures the dissimilarity between predicted probabilities and actual class labels.

$$L\left(f_w(x^{(i)}), y^{(i)}\right) = -[y \log(f_w(x^{(i)})) + (1-y) \log(1-f_w(x^{(i)}))]$$

$$L(\hat{y}, y) = -[y \log(\hat{y}) + (1-y) \log(1-\hat{y})]$$

- For m training samples,

$$J(w) = \frac{-1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log(f_w(x^{(i)})) + (1-y^{(i)}) \log(1-f_w(x^{(i)})) \right]$$



① → (y = 1) $L = -\log(\hat{y})$

② → (y = 0) $L = -\log(1-\hat{y})$

✱ Decision Tree

- Flowchart-like structure used for classification or regression.
- Here, Nodes → features (attributes)
  Branches → decisions
  Leaf Nodes → outcomes (class labels or values)
- Decision Trees split the data recursively based on feature values.

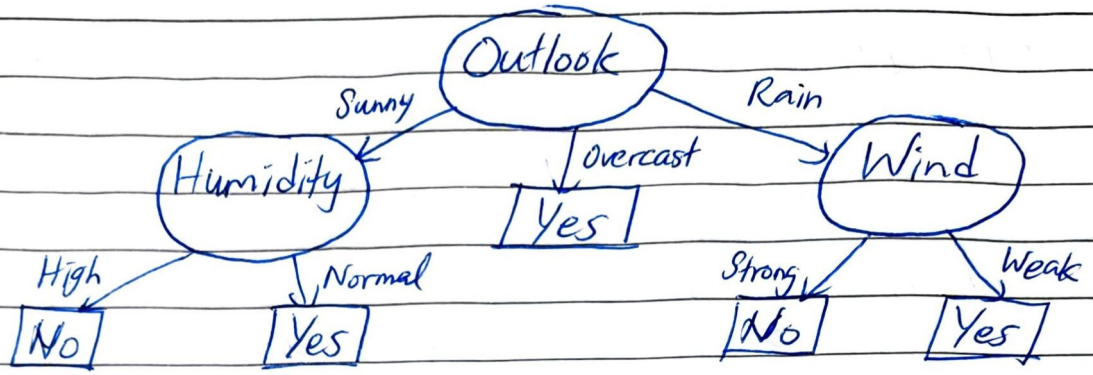
Feature	Univariate Tree	Bivariate Tree
• Split Criterion	• Single feature at a time	• Combination of multiple features
• Node Decision	• One feature per node (Best feature/threshold)	• Linear combination possible
• Complexity	• Simpler	• More complex
• Interpretability	• High	• Lower

Ex: CART, ID3, C4.5

✱ Univariate Trees

- At each internal node, only one feature is used to split the data.
- The algorithm searches for the best feature and threshold to split on

__/__/__



- Goal of induction: Build a tree that works well on on unseen data, not just training samples (generalization) One that captures the real patterns / relationships between class labels and attributes.

- Occam's Razor Simplicity principle advises to choose the simplest tree that can capture the true structure and classify unseen data better.

- Overfitting: Tree fits training data perfectly but performs worse on testing data, due to noise (high bias)

— ID3 Algorithm

- Divide-and-conquer process where attribute with the highest impact on classification goes at the root and splits data.

- Repeat recursively until all samples at a node belong to the same class.

- This can be determined with the help of split criterion such as Information Gain / Entropy Reduction, Gini Ratio, etc.

- **Entropy**: Measures level of impurity / disorder in a dataset.

$$\text{Entropy} = -P_1 \log_2(P_1) - P_2 \log_2(P_2)$$

where P_1, P_2 are proportions / probabilities of Class 1 and Class 2 samples in dataset.

- Entropy = 0 ⇒ Node is pure
  Entropy = 1 ⇒ Maximum impurity (equal class distribution (50-50))
- **Information Gain (IG)**: Reduction of entropy after split.

$$IG = \text{Entropy}(D) - \sum_{i=1}^{d} \frac{|D_i|}{|D|} \text{Entropy}(D_i)$$

IG = Overall Entropy − Weighted Entropy (of each attribute)
(from entire dataset) (of that category)

Here D → entire data, D_i → subset of D (each attribute)

- Choose the category with the highest IG as the root.

Ex: Dataset D: 14 samples (9 Yes, 5 No) ~~Outlook~~

$$\text{Entropy}(D) = \frac{-9}{14}\log_2\left(\frac{9}{14}\right) - \frac{5}{14}\log_2\left(\frac{5}{14}\right) = 0.941$$

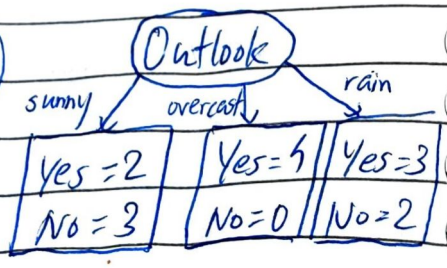
(highly impure)

If we split the dataset by Outlook

$$\text{Entropy}(\text{Sunny}) = \frac{-2}{5}\log_2\left(\frac{2}{5}\right) - \frac{3}{5}\log_2\left(\frac{3}{5}\right)$$

$$= 0.971$$

$$\text{Entropy}(\text{Overcast}) = \frac{-4}{4}\log_2\left(\frac{4}{4}\right) = 0$$



$$\text{Entropy (Rain)} = -\frac{3}{5}\log_2\left(\frac{3}{5}\right) - \frac{2}{5}\log_2\left(\frac{2}{5}\right) = 0.971$$

$$\text{Weighted Entropy (Outlook)} = \frac{5}{14}(0.971) + \frac{4}{14}(0) + \frac{5}{14}(0.971)$$

$$= 0.693$$

$$IG(\text{Outlook}) = \text{Entropy}(O) - \text{Weighted E}$$

$$= 0.971 - 0.693 = 0.248$$

Similarly find for other attributes and choose the one with the highest IG.

- CART Algorithm

- Gini Impurity: Probability of incorrectly classifying a randomly chosen element from the dataset if it was labelled according to class distribution in that node

- It tells how mixed the classes are within a single node of a decision tree.

$$\text{Gini}(D) = 1 - \sum_{i=1}^{m} p_i^2$$

(m classes)

$p_i \Rightarrow$ proportion of class i in the node.

- Pure node $\Rightarrow$ Gini = 0

For m classes, maximum Gini $= 1 - \frac{1}{m}$ (equal class distribution)

Ex: $m = 2$, maximum impurity/gini $= 1 - \frac{1}{2} = 0.5$

- For Binary Splitting on Continuous Attributes,

① Sort values for each continuous variable in ascending order.
② Generate split points for each pair of candidates (row) (midpoint between consecutive sorted values
③ For each candidate split, split the dataset into:
Left subset : values ≤ split point
Right subset : values > split point
And for each subset, count the no. of each output class in that subset and generate gini for that subset.

④ Compute weighted gini after the split.

$$gini_{split} = \frac{|D_L|}{|D|} gini(D_L) + \frac{|D_R|}{|D|} gini(D_R)$$

⑤ Compute gini gain (reduction)

$$GiniGain = Gini(D) - Gini_{split}$$

Larger the GiniGain the better the split.
Pick the split with the highest ~~Gini~~ GiniGain.

⑥ Repeat recursively until Gini = 0 (pure node) or stopping criterion is met (min samples per node)

Feature	CART	ID3
• Full Name	Classification & Regression Trees	Iterative Dichotomiser 3 (classification only)
• Splitting Criterion	Gini Index (C) Variance Reduction (R)	Information Gain (based on Entropy)
Produces	Binary Tree	Multiway Tree

Note: When it comes to handling numerical values, CART can directly handle continuous variables by finding optimal split points, while ID3 needs to categorize those numerical values before splitting.

— Advantages (of Univariate Decision Trees)

- Simple and interpretable
- Fast Training and Low Complexity
- No feature scaling needed
- Feature importance (insight)

— Disadvantages

- Overfitting Risk (can create deep trees that fit noise unless pruning / regularized)
- Instability (upon small change in dataset)
- Not always optimal
- Limited Expressiveness (since splits based on one feature, they cannot capture feature interaction)
- Bias towards features with more levels (features with many unique values may dominate splits may seem impure)

★ Pruning

- Fully grown trees can perfectly classify training data, but may ~~overfit~~ capture noise (overfitting)
- Pruning helps to improve generalization by reducing random coincidental splits for better interpretability

__/__/__

(Types)

- Pre-pruning (Early Stopping)
  - Stop tree growth early using thresholds on purity gain, depth, min samples, etc. However it is hard to know the right stopping point.
- Post pruning
  - Grow the full tree, then remove branches that add little predictive power.

(Criteria)

- Error-based Pruning
  - Avoid overfitting by balancing performance on training and testing data.
  - If pruning reduces test error, even if training error increases, we keep this pruned version.
- Adjusted Error-Rate
  - Adjusted Error = Misclassification Error + Penality (Tree Size)
  - Thus encouraging simpler trees that generalize better.
- Cost Complexity (CART)
  - Cost Complexity, $CC(T)$ = Misclassification Error / Impurity $+ \alpha$ (no. of leaves (tree size))
  - Here α → complexity parameter (controls penalty strength)
    Small α → large trees allowed
    Large α → smaller trees preferred.

— Pessimistic Pruning

- When we don't have a seperate validation / test set (training error = 0, overfitting, too optimistic)
- By adding a small penalty (error per leaf node) to account for bias, on the training error, we compare the adjusted error of the subtree (vs) as a leaf. If the latter is much less, prune the subtree.

Note Regression Trees are built just like classification trees, just impurity is replaced by regression error (MSE). Here, output is discrete, each leaf gives constant value.

Aspect	Parametric	Decision Tree (non-parametric)
Function	Assumes global function over input space	No assumption about global form
Parameters	Learns fixed set of parameters from training set	Learns structure (splits) directly from data.
Application	Same parameter set used for all test inputs	Tree adapts to input through branches.

— Advantages of Decision Tree

- Transparent if-then rules
- Easy to interpret and explain
- Works well on non-metric discrete features.
- Good for pattern recognition with mixed features
- Complementary to k-NN and ANN (rely on features & similarity) (rely on distances)

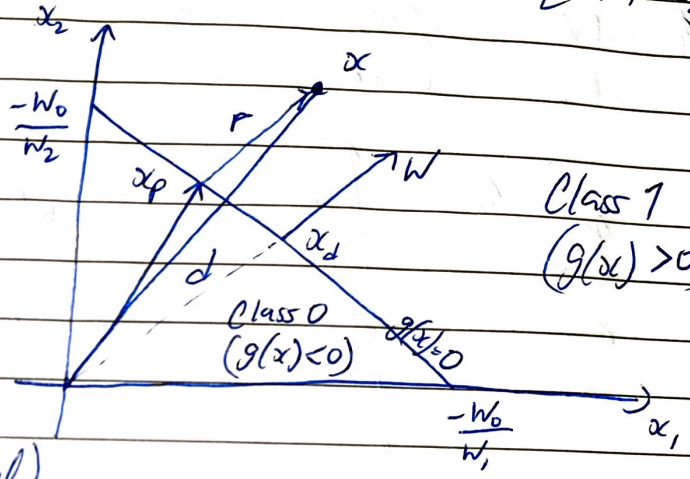
__/__/__

Disadvantages

- Overfitting & Instability
- Missed feature interactions
- Computation cost (sorting splits, pruning)
- Discrete approximation in Regression
- Bias towards features with many values.

BINARY CLASSIFICATION

- Assigns input patterns to two classes by finding a linear decision boundary that separates those two classes in the feature space.
- Each input pattern = feature vector = $[x_1, x_2]^T$
- In 2D,



$(g(x) = W_1 x_1 + W_2 x_2 + W_0 = 0)$

where

$W = [W_1, W_2]^T$

(weight vector, defines slope / normal)

(W is perpendicular / orthogonal to the decision boundary)

(defines the orientation of the ~~type~~ decision boundary)

(which direction the boundary faces, how steep / tilted)

and

W_0 → bias term (controls where boundary intersects axes)

(shifts the boundary away from the origin)

- A linear classifier uses a discriminant function:

$$g(x) = W^T x + W_0$$ (equation of n-D hyperplane)

① Compute weighted sum: $g(x) = \sum_{j=1}^{n} W_j x_j + W_0$

(Aggregation unit)

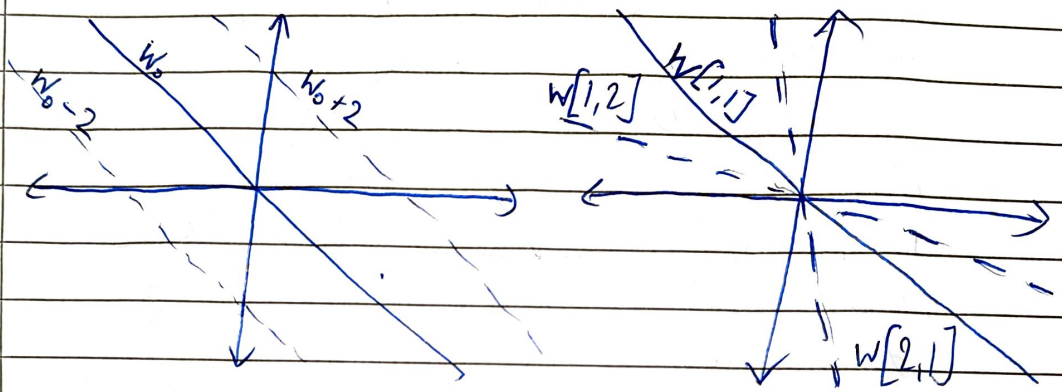
Bias term represented as constant input $x_0 = 1$ with weight W_0

② Output unit: (Step / sign function)

$$\hat{y} = \text{sgn}(g(x)) = \begin{cases} +1, & g(x) > 0 \quad \text{(Class 1)} \\ -1, & g(x) < 0 \quad \text{(Class 0)} \end{cases}$$

- If $g(x) = 0$, point lies on decision boundary. (undecided)

Note: $w \rightarrow$ defines orientation (Up-down, Left-Right)
$w_0 \rightarrow$ defines placement (forward / backward)



- Scaling (kw, kw_0) does not change the hyperplane. only relative ratio of coefficients matters

- Any point on the hyperplane can be written as:

$$x = x_p + \frac{r}{\|w\|} w$$

→ perpendicular offset from hyperplane

where $x_p \rightarrow$ projection of point x onto hyperplane
$r \rightarrow$ signed distance from hyperplane

$$r = \frac{g(x)}{\|w\|} \qquad d = \frac{w_0}{\|w\|} \text{ (distance from origin)}$$

- $w_0 < 0$, origin lies on negative Half-space $(g(x)<0)$ (H^-)
$w_0 > 0$, origin lies on positive Half-space $(g(x)>0)$ (H^+)
$w_0 = 0$, hyperplane passes through the origin. (H)

- w points in the direction where $g(x)$ increases.

★ Perceptron Algorithm

- Main goal of a classifier is to minimize the no. of misclassifications.

- This algorithm iteratively updates weights and moves the hyperplane until it correctly seperates and reduces classification errors.

- Uses misclassified samples to adjust the decision boundary.

① Initialize weights w and w_0
② For each training sample $(x^{(i)}, y^{(i)})$
- Compute output $\hat{y}^{(i)} = \text{sgn}(w^T x^{(i)} + w_0)$ just like simple linear classifier
- If correct, no update
- If incorrect, update weights and bias
③ Repeat until all are correctly classified.

- If $y^{(i)} = +1$, $g(x^{(i)}) > 0$ ✓ (+)
  $g(x^{(i)}) \leq 0$ ✗ (−)
  If $y^{(i)} = -1$, $g(x^{(i)}) < 0$ ✓ (−)
  $g(x^{(i)}) \geq 0$ ✗ (+)

- Hence if $y^{(i)} g(x^{(i)}) > 0$ → correct classification
  $y^{(i)} g(x^{(i)}) \leq 0$ → misclassification.

- For each misclassified training point, $(x^{(i)}, y^{(i)})$, the updation formula is as follows:

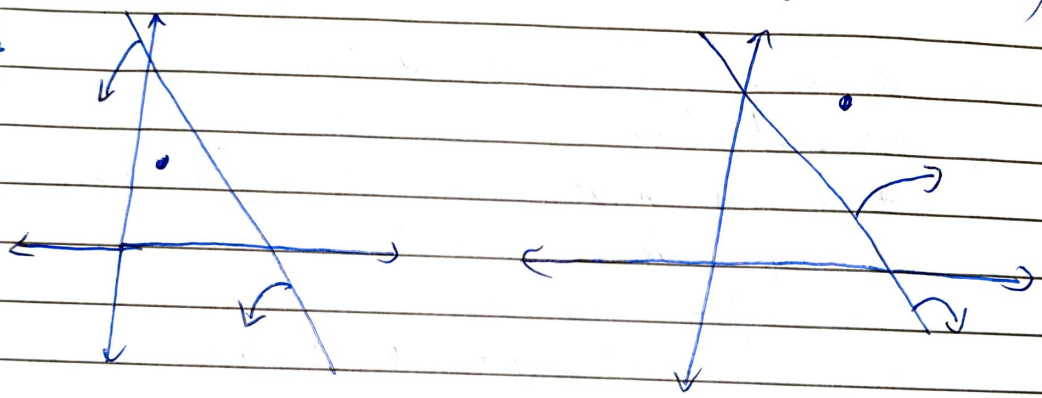
$$w \leftarrow w + \eta\, y^{(i)} x^{(i)}$$

where $y^{(i)} \in \{-1, 1\}$ (true labels)
$x^{(i)} \rightarrow$ feature vectors
$\eta \rightarrow$ learning rate (step size) $(0 < \eta < 1)$

- Geometrically, consides $y^{(i)} = +1$ (supposed to be positive but misclassified)

$$\left(w \leftarrow w + \eta\, x^{(i)}\right)$$

w rotates toward the positive sample (anti-clockwise)



If $y^{(i)} = -1$ (supposed to be negative but misclassified)

$\left(w \leftarrow w - \eta\, x^{(i)}\right)$ (w rotates ~~away from~~ towards the negative sample (clockwise))

$$w_0 \leftarrow w_0 + \eta\, y^{(i)} R^2$$

where R is the farthest distance of any training point from origin (talks about scale of dataset)

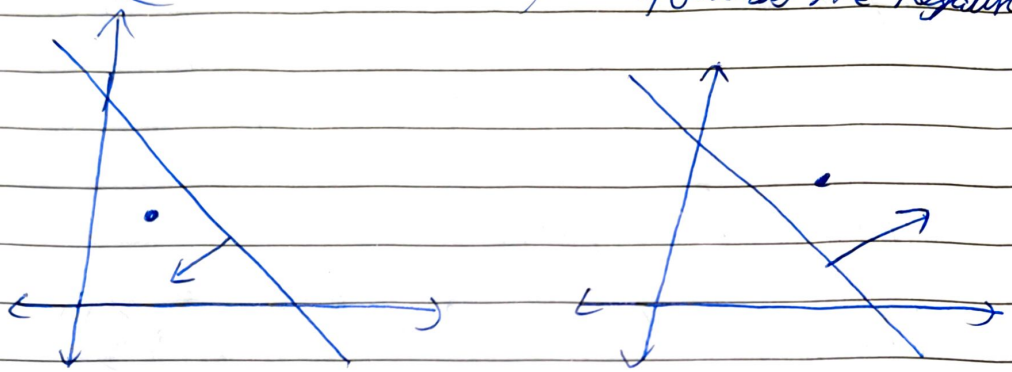
$$\left(R = \max \|x^{(i)}\|\right) \quad \text{(maximum length of training vector)}$$

- Geometrically, if $y^{(i)} = +1$ (supposed to be positive)

$(w_0 \leftarrow w_0 - \eta R^2)$ (translate the hyperplane away from the positive sample)

If $y^{(i)} = -1$ (supposed to be negative)

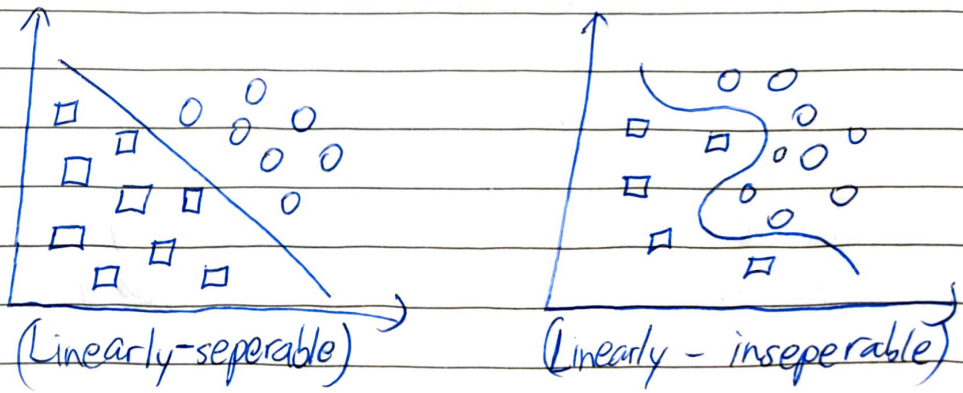
$(w_0 \leftarrow w_0 + \eta R^2)$ (translate the hyperplane towards the negative sample)



- One major limitation of perceptron algorithm is that the hyperplane may lie too close to training samples but fail to classify unseen test data → poor generalization

- Also this algorithm only works with linearly seperable data (which is rare to obtain in real world) else ∞ loop occurs.

- Despite being simple and intuitive, it is sensitive to noise and outliers and cannot provide probability estimates.



★ Winnow Algorithm

- Uses multiplicative weight updates instead of additive like Perceptron.
- Best suited for high-dimensional data ~~to keep~~ Keeps only relevant features, ignores the rest.
- Input vects $x = [x_1, x_2, \ldots, x_n]^T$, $x_i \in \{0, 1\}$

Weights, $w = [w_1, w_2, \ldots, w_n]^T$, $w_i > 0$

$g(x) = \sum w_i x_i$

Decision rule $\Rightarrow$ $\hat{y} = \begin{cases} 1, & g(x) \geq \theta \\ 0, & g(x) < \theta \end{cases}$ (threshold)

① Start with all weights equal ($w_i = 1$)
Choose learning rate multiplier, $\alpha > 1$ and threshold θ

② Compute weighted sum $g(x)$ and decision $\hat{y}$.
- If $\hat{y}$ is correct, no update.
- If $\hat{y}$ is wrong, update only the active features

③ • Promotion (FN) ($y = 1$ but $\hat{y} = 0$)
Boost importance of active features, ($x_i = 1$)

$$w_i \leftarrow w_i \times \alpha$$

Demotion (FP) ($y = 0$ but $\hat{y} = 1$)
Reduce importance of active features ($x_i = 1$)

$$w_i \leftarrow \frac{w_i}{\alpha}$$

- Even though this algorithm is robust and converges fast, it is still sensitive to noise, requires binary input ($x_i = 0$ or 1) and choosing α, θ is critical.

Perceptron	Winnow
• $w \leftarrow w + \eta y^{(i)} x^{(i)}$ (additive correction)	• $w_i \leftarrow w_i \cdot \alpha$ (or) w_i / α (multiplicative correction)
• All features treated equally.	• Promotes only important features suppresses irrelevant ones.
• Requires linearly separable data	• Requires high dimensional, sparse data.
• Output $\{+1, -1\}$	• Output: $\{0, 1\}$
• Slow adaptation	• Fast promotion/demotion of useful features.

Ex: $x = [1, 0, 1, 0]$, $y = 1$, $w = [1, 1, 1, 1]$
$\alpha = 2$, $\theta = 2.5$

$g(x) = 1.1 + 0.1 + 1.1 + 0.1 = 2 < \theta$ $(\hat{y} = 0)$ (promotion)
Thus promote all active features weights on x.

$w_{new} = [2, 1, 2, 1]$, $x = [1, 0, 1, 0]$

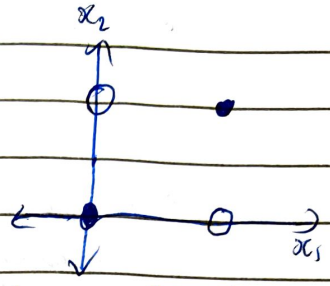
$g(x) = 2 + 0 + 2 + 0 = 4 > \theta$ $(\hat{y} = 1)$ ✓

Kernel Methods

- The problem with Perceptron and Winnow algorithms is that they only work with linearly separable data, but not all data is linearly separable.

Ex- **XOR pattern**

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0



Not separable by a linear hyperplane.

- This is where kernel methods come to play; instead of drawing a curved boundary in the original space, we transform the data into a higher-dimensional space where a hyperplane can separate them (+1 dimension)

- We define a mapping $\phi : R^n \rightarrow R^m$ where $m > n$

which adds an extra (derived) feature to make them linearly separable (squared terms, cross-terms)

$$g(x) = w^T \phi(x) + w_0$$

which in higher-dimension would be a straight line but in the original space, becomes a non-linear curve

Ex- For the same XOR pattern example, add another dimension (x_1, x_2) to make the feature space 3D from 2D

If $x = [x_1, x_2]$, $\phi(x) = [x_1, x_2, x_1 x_2]$ (non-linear mapping)

x_1	x_2	$x_1 x_2$	y
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

(Try plotting in Desmos)

Now we can separate $y=0$ and $y=1$ with a plane.

- Here $\phi(x)$ is some function that transforms x into new features
- Each time we misclassify a sample $(x^{(i)}, y^{(i)})$

$$w \leftarrow w + \eta y^{(i)} \phi(x^{(i)})$$

After multiple updates, $w = \sum \alpha_i y^{(i)} \phi(x^{(i)})$
where α measures how strongly that training point x_i has influenced the final model. (point/sample importance)

Hence, $g(x) = \sum \alpha_i y^{(i)} \langle \phi(x_i), \phi(x) \rangle + w_0$

Here $\phi(x_i)$ → training sample in the feature space
$\phi(x)$ → current/test sample (dot product is computed)

- Dot product tells how similar they both are in the transformed feature space. (thus contributing in the decision making of the classifier)

$$K(x, z) = \langle \phi(x), \phi(z) \rangle$$ (Kernel function)

- Every prediction is built from the similarities between the new point and the training points.

$$g(x) = \sum \alpha_i y_i K(x_i, x) + b$$

where $\alpha \rightarrow$ influence, $K \rightarrow$ similarity.
$y \rightarrow$ label

— **Kernel Functions**

① **Linear** (same as linear classifier)

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

(no mapping, feature space = input space)
$x \in R^N \rightarrow \phi(x) \in R^N$

$$g(z) = \sum \alpha_i y_i x^T z + b$$

$$= \left(\sum \alpha_i y_i x_i\right)^T z + b$$

$$g(x) = w^T z + b$$

② **Polynomial Kernel**

For $c > 0$, $d \in N$, $x, z \in R^N$

$$K(x, z) = (x^T z + c)^d$$

which expands to include powers and cross-terms

③ **Gaussian Kernel**

For $\sigma > 0$, $x, z \in R^N$

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

Also called Radial Basis Function (RBF) kernel
Value depends only on distance $\|x - z\|$, measuring closeness in Euclidean space.

- Widely used in SVM, kernel regression, clustering.

(4) Sigmoid (Hyperbolic Tangent Kernel)
For $a, b \in \mathbb{R}$, $x, z \in \mathbb{R}^N$

$$K(x, z) = \langle \phi(x), \phi(z) \rangle = \tanh(a x^T z + b)$$

Neural-network like activation, often used in SVMs.

Ex: Using 2D, 2-degree polynomial mapping for the XOR problem, for $x = [x_1, x_2]$, $z = [z_1, z_2]$, $d = 2$

$$K(x, z) = (x_1 z_1 + x_2 z_2 + c)^2$$
$$= x_1^2 z_1^2 + x_2^2 z_2^2 + c^2 + 2 x_1 x_2 z_1 z_2 + 2c(x_1 z_1 + x_2 z_2)$$

2D → 6D

$$\phi(x) = [x_1^2, x_2^2, \sqrt{2}\, x_1 x_2, \sqrt{2c}\, x_1, \sqrt{2c}\, x_2, c]$$
$$\phi(z) = [z_1^2, z_2^2, \sqrt{2}\, z_1 z_2, \sqrt{2c}\, z_1, \sqrt{2c}\, z_2, c]$$

For $c = 1$, $\phi(x) = [x_1^2, x_2^2, \sqrt{2}\, x_1 x_2, \sqrt{2}\, x_1, \sqrt{2}\, x_2, 1]$

$\phi(0,0) = [0, 0, 0, 0, 0, 1]$

$\phi(0,1) = [0, 1, \sqrt{2}, 0, \sqrt{2}, 1]$

$\phi(1,0) = [1, 0, \sqrt{2}, \sqrt{2}, 0, 1]$

$\phi(1,1) = [1, 1, \sqrt{2}, \sqrt{2}, \sqrt{2}, 1]$

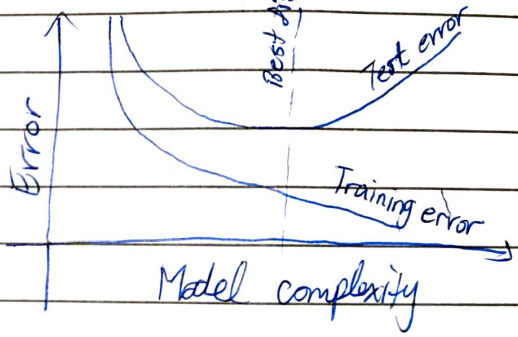
PRACTICAL ASPECTS OF ML

- Overfitting occurs when the model learns noise and random fluctuations in the training data instead of the underlying pattern, leading to poor performance on new unseen data despite performing well on the training data. (Mugging up)

- Generalization is the models ability to perform well on new unseen data

- The ultimate goal of Machine Learning is to balance ~~fetting~~ fitting training data while maintaining flexibility to adapt.

- This divergence in test error tells that the model has begun memorizing than learning meaningful patterns.



— Causes

- Small datasets with limited examples
- Noisy / inconsistent training data
- Insufficient regularization techniques
- Overly complex models, too many parameters.

— Solutions

- Implement simple model architectures
- Apply regularization techniques (L1/L2)
- Collect more diverse training data

- Use dropout layers in neural networks.
- Cross-validation (robust train-test split)
- Ensemble multiple models
- Data Augmentation (transformations, synthetic data)
- Early Stopping (stop training before overfitting occurs)

— Black Swan Paradox (coined by Nassim Nicholas Taleb)

- Rare, unpredictable, high-impact events that defy our expectations.

Ex: 2008 Financial Crisis, COVID-19 pandemic.

- These events lie outside training experience and challenge model assumptions.
- ML models that rely on historic data, perform catastrophically worse on unseen/rare cases
- Bayesian models handle uncertainity better than frequentist ones (by assigning probabilities to rare events)
- Such models must handle distribution shifts and real-world diversity with ease.
- Strategies to mitigate Black Swan Risk include:

(i) Diversify Data Sources (use federated learning) ✍
(ii) Embrace Uncertainity
(iii) Continuous Adaptation (evaluate and adapt models to detect and respond to distribution shifts in real-time.)

Bias - Variance Tradeoff

Bias

- Error from overly simplistic assumptions in the learning algorithm, model fails to capture underlying patterns and relationships in the data.
- High bias → Underfitting (model too simple)

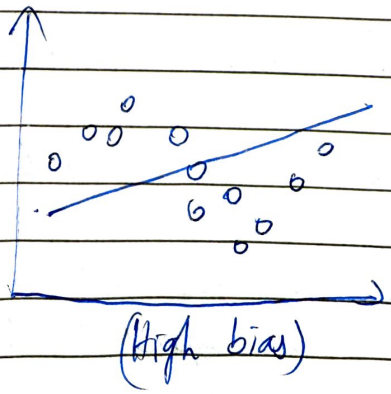
Ex: Using a linear model to fit non-linear data.
(Poor predictions on training data)

Variance

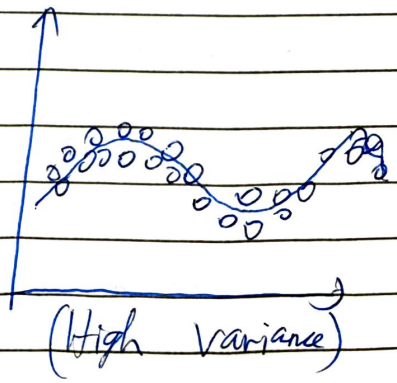
- Error caused by the model being too sensitive to small fluctuations in the training data. Model memorizes noise instead of learning generalizable patterns
- High variance → Overfitting (model too complex)
(poor generalization)

Ex: Mugging up training data

- Overfitting creates instability, leading to poor predictions on new data. (even though less training error)



(High bias)



(High variance)

ML (Contd.)

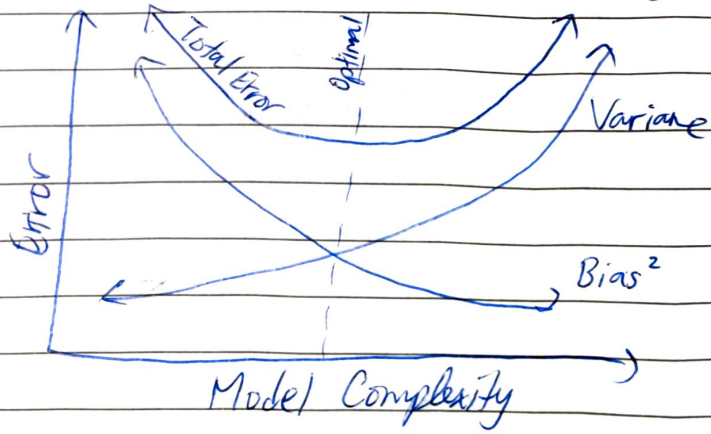
Bias	Variance	Problem	Behaviour
Low	Low	None	Ideal generalization
High	Low	Underfitting	Too simple
Low	High	Overfitting	Too complex
High	High	Poor Model	Neither fits nor generalizes.

Ex: For polynomial regression, mean squared error tells how model complexity affects accuracy across different polynomial degrees.

- Model Complexity ↑, Bias ↓, Variance ↑ (need to balance)

— Strategies

- Find the optimal model complexity by minimizing total error = $Bias^2$ + Variance + Irreducible Error



- Apply L1 (Lasso) or L2 (Ridge) regularization to reduce variance by penalizing complex parameters.
- Feature Engineering (selection and dimensionality reduction to reduce model complexity)
- Cross Validation and Ensemble Learning (bagging, boosting)

★ Optimization Techniques

- To minimize loss functions & improve model predictions through iterative parameter updates

Ex: Mean Squared Error (MSE) for regression
Cross-Entropy Loss for classification
Hinge Loss for SVMs.

- This is done by moving parameters opposite to the gradient direction to minimize loss (in loss-parameter graph) (gradient descent)

— Batch Gradient Descent (computationally expensive)

- Uses all of the training data per gradient calculation and shuffles the dataset.
- For each sample ~~in~~ (in this case, the entire dataset)

$$W \leftarrow W - \eta \frac{\partial L}{\partial W}$$

and repeat until convergence. (large scale optimization)

— Stochastic Gradient Descent

- Uses one sample of the training data per gradient calculation
- For each sample x_i,

$$W \leftarrow W - \eta \left(\frac{\partial L}{\partial W}\right)_{x_i}$$

- Introduces noisy updates but faster convergence.

— Mini-Batch Gradient Descent

- Uses a certain amount of samples per update, balancing computation and memory.
- More stable and efficient compared to SGD while maintaining speed (parallel processing, optimized memory usage)

(i) Must consider tuning Learning Rate (η)
Too large cause divergence, too small slows convergence

(ii) Data should be randomly shuffled each epoch to prevent biased updates and improve convergence stability.

(iii) Advanced optimizers that incorporate momentum (Adam, RMSProp) and adaptive learning rates enhance stability and accelerate training.

— Gibbs Algorithm

- Unlike gradient descent where one solution of best fit is computed, Bayesian model approximate a possible combination of values that fit the data well each with a probability. (we care about uncertainty)
- Part of MCMC (Markov Chain Monte Carlo) technique framework for sampling from complex distributions
- Uses conditional probability to generate new values one variable at a time.

- Suppose we have two variables X and Y, we want to know how they behave together (joint probability distribution $P(X, Y)$)

We know $P(X/Y)$ and $P(Y/X)$
Initialize a value for X and Y

Update X by $P(X/Y)$
Update Y by $P(Y/X)$
Repeat until value of (X, Y) begin to settle into area that represent true joint probability distribution.
→ Markov Chain.

Structured Output Prediction

- Predicting outputs that are related to each other, not independent
- The model must make sure the output is globally consistent (all predictions fit together, logically)
- Simple classification → one label (simple, independent)
  Email → Spam / Not spam
- Structured prediction → multiple related labels / sequence
  Sentence → [Noun, Verb, Noun, ...] (tagged words)
  (Complex, dependent structures predicted)
- Types of Structured prediction include:
  - Multi-label Classification (One input → multiple labels)
  - Sequence tagging (sequence of outputs for sequence inputs)

Multi label Classification

- For a given input x, predict a subset of labels $Y \subseteq L$

If there are k possible labels, there are 2^k label combinations (main challenge)

Ex: Input = Image of man riding a bicycle
Output = {Person ✓, Bicycle ✓, Car ✗, Outdoor ✓}

- Evaluation metrics include Hamming loss (fraction of labels predicted wrong) and Jaccard similarity (F1 Score) (measure overlap between true and predicted label sets.)

- We convert the multi-label problem into several simpler ones:

(a) Binary Relevance (BR): Train one independent classifier per label $f_i(x) = l_i$
(Simple, scalable but ignores label relationships)

(b) Classifier Chains (CC):
Each classifier uses not only one input x but also previous label predictions, $f_i(x, l_1, \ldots, l_{i-1})$
(Models label dependencies but order matters)

(c) Label Powerset (LP): Treat each unique combo of labels as a single class (captures all dependencies but too many classes, sparse data)

- Model existing ML algorithms to handle multiple labels such as a shared deep learning model (CNN, Transformer) followed by k independent sigmoid output units (BR).

Sequence Tagging

- Given a sequence $x = (x_1, x_2, \ldots, x_n)$ predict another sequence $y = (y_1, y_2, \ldots, y_n)$ where y_i is a tag for x_i.

Ex: In Named Entity Recognition (NER)
Sentence = "The CEO of Apple visited London last week"
Tags = [O, O, O, ORG, O, LOC, O, O]

- Markov Assumption: Tag of word mainly depends on current word x_i and previous tags y_{i-1} sometime y_{i-2}
- Evaluation Metrics include per-token accuracy (simply checks each tag individually) and chunk F1 Score (measures full entities correctly tagged)
- Classic Models used for sequence tagging include:

(a) HMM (Hidden Markov Model) that models the joint probability of tags and words. Defined by initial probabilities ($P(y_1)$) transition probabilities $P(y_i | y_{i-1})$ and emission probabilities $P(x_i | y_i)$.

Uses Viterbi algorithm to find the most likely tag sequence.

(b) CRF (Conditional Random Field) is a smarter version of HMM that instead of using fixed probabilities, learns flexible feature weights to ensure entire sequence is globally consistent. Models $P(y|x)$ instead of $P(x, y)$ (Captures dependencies between tags)

(c) Bidirectional LSTM: Reads sequence from L→R and R→L and captures context from both sides, outputs context-aware representation of each word.

(d) BiLSTM + CRF: Enforces valid transitions on tags with generated feature scores.

(e) Transformers (BERT, RoBERTa): Use self-attention to capture long-term dependencies across the sequence.

PROBABILISTIC MODELLING

- Represent knowledge as probability distributions, learn from data $P(x, y)$, $P(y/x)$, $P(x/y)$

★ Discriminative Models (Learn $P(y/x)$)

- They directly learn the conditionally probability distribution $P(y/x)$ (probability of label y given features x) (doesn't care about how x is generated)
- For example, Logistic Regression,
  Probability of class $y=1$, $P(y=1/x) = \sigma(w^T x + b)$
  where $\sigma(z) = \frac{1}{1+e^{-z}}$ (sigmoid)
  Output a probability score between 0 and 1 (threshold = 0.5)

★ Generative Models (Learn $P(x, y)$)

- They aim to learn the joint probability distribution $P(x, y)$ (how labels (y) and features (x) occur together)

— Naive Bayes Classifier

- Build probabilistic models of data distribution that generalize well even with limited training data.
- Uses Bayes rules to predict labels.

$$P(y_k/x) = \frac{P(x/y_k) \cdot P(y_k)}{P(x)}$$

where $P(y_k/x)$ → posterior probability (probability of class y_k after seeing x)

$P(y_k)$ → prior probability (initial probability of class y_k)

$P(x/y_k)$ → likelihood (how likely x is given class y_k)

$P(x)$ → Evidence (probability of observing x across all classes)

$$P(x) = \sum P(x/y_q) \cdot P(y_q)$$

- Goal is to assign x to a class with maximum posterior probability. $P(y_k/x)$

$$\sum P(y_q/x) = 1$$

- To handle continuous features, (for $P(x/y_k)$)

(a) Discretize into bins (dividing continuous numerical values into discrete intervals → categories) (useful for small datasets)

(i) Equal-width binning (bins of equal size)

(1-2), (2-3), (3-4), (4-5), ...

(ii) Equal-frequency binning (each bin has same no. of data points) (useful for uneven distributions)

(5 points) (5 points), (5 points) ...

(iii) Domain-driven / Custom Binning (based on expert knowledge or thresholds)

Age → {Child, Teen, Adult, Senior}

(b) Use probability density function for large datasets.

$$P(x/y_q) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu)^2}{2\sigma^2}} \quad \text{(Gaussian)}$$

- We assume that features are dependent on each other while calculating joint probability using chain rule.

$$P(x_1, x_2, x_3, \ldots / y_q) = P(x_1 / y_q) \cdot P(x_2 / x_1, y_q) \cdot \ldots$$

But due absence of structural knowledge, we assume features are independent given the class. (conditionally)

$$P(x/y_q) = \prod_{j=1}^{n} P(x_j / y_q)$$

$$y_{NB} = \arg\max_{y_q} \left(P(y_q) \cdot \prod_{j=1}^{n} P(x_j / y_q) \right)$$

- All features are equally important and conditionally independent given the class.
- Prior: $P(y_q) = \frac{N_q}{N}$ (No. of training samples of class y_q)

$x_j \rightarrow j^{th}$ feature (attribute) in the dataset.
$V_{x_j} \rightarrow$ set of possible values that x_j can take

Ex: If $x_j \rightarrow$ Weather, $V_{x_j} = \{Sunny, Rainy, Cloudy\}$

- $V_{nx_j} \rightarrow n^{th}$ value in that set
- Likelihood, $P(x_j = V_{nx_j} / y_q) = \frac{N_q(V_{nx_j})}{N_q}$

where $N_q(V_{nx_j})$ → no. of samples in class y_q with attribute $x_j = V_{nx_j}$

- Example

Determine predicted class for $x = \{\text{Gender} = M, \text{Height} = 1.95\text{m}\}$

Gender (x_1)	Height (x_2)	Class
F	1.6	Short
M	2	Tall
F	1.9	Medium
F	1.88	Medium
F	1.7	Short
M	1.85	Medium
F	1.6	Short
M	1.7	Short
M	2.2	Tall
M	2.1	Tall
F	1.8	Medium
M	1.95	Medium
F	1.9	Medium
F	1.8	Medium
F	1.75	Medium

[Short = y_1, Medium = y_2, Tall = y_3]

$N = 15$

No. of classes = 3

$V_{x_1} : \{M, F\} = \{V_{1x_1}, V_{2x_1}\}$, $d_1 = 2$

$V_{x_2} : \{V_{1x_2}, V_{2x_2}, V_{3x_2}, V_{4x_2}, V_{5x_2}, V_{6x_2}\}$

bins $= \{(0, 1.6), (1.6 - 1.7), (1.7 - 1.8), (1.8 - 1.9), (1.9 - 2.0), (2.0 - \infty)\}$ $d_2 = 6$

(Count $N_q(v)$) — 1 1

	Value	Short	Medium	Tall
V_{1x_1}	M	1	2	3
V_{2x_1}	F	3	6	0
V_{1x_2}	0-1.6	2	0	0
V_{2x_2}	1.6-1.7	2	0	0
V_{3x_2}	1.7-1.8	0	3	0
V_{4x_2}	1.8-1.9	0	4	0
V_{5x_2}	1.9-2.0	0	1	1
V_{6x_2}	2.0-∞	0	0	2

Test Data : $x : \{M, 1.95\} = \{x_1, x_2\}$
V_{1x_1}, V_{5x_2}

$P(y_1) = \frac{4}{15} = 0.267$

$P(y_2) = \frac{8}{15} = 0.533$

$P(y_3) = \frac{3}{15} = 0.2$

$P(x_1/y_1) = \frac{N_{1V_{1x_1}}}{N_1} = \frac{1}{4}$ (Short + Male)

$P(x_1/y_2) = \frac{N_{2V_{1x_1}}}{N_2} = \frac{2}{8}$ (Medium + Male)

$P(x_1/y_3) = \frac{3}{3} = 1$ (Tall + Male)

$P(x_2/y_1) = \frac{0}{4} = 0$ (1.9-2.0 → Short)
1.95

$P(x_2/y_2) = \frac{1}{8}$ (1.95 → Medium)

$P(x_2/y_3) = \frac{1}{3}$ (1.95 → Tall)

$\Pi P(x_j/y_1) = P(x_1/y_1) \times P(x_2/y_1) = \frac{1}{4} \times 0 = 0$

$\Pi P(x_j/y_2) = P(x_1/y_2) \times P(x_2/y_2) = \frac{2}{8} \times \frac{1}{8} = \frac{1}{32}$

$\Pi P(x_j/y_3) = P(x_1/y_3) \times P(x_2/y_3) = 1 \times \frac{1}{3} = \frac{1}{3}$

$\Pi P(x_j/y_1) \cdot P(y_1) = 0 \times 0.267 = 0$

$\Pi P(x_j/y_2) \cdot P(y_2) = \frac{1}{32} \times 0.533 = 0.0166$

$\Pi P(x_j/y_3) \cdot P(y_3) = \frac{1}{3} \times \frac{3}{15} = 0.066$

$y_{NB} = \underset{}{\text{argmax}}\,(0, 0.0166, 0.066)$
$= 3$

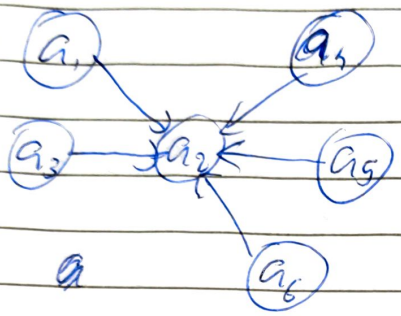
∴ Predicted class is Tall

★ Bayesian Belief Network (BBN)

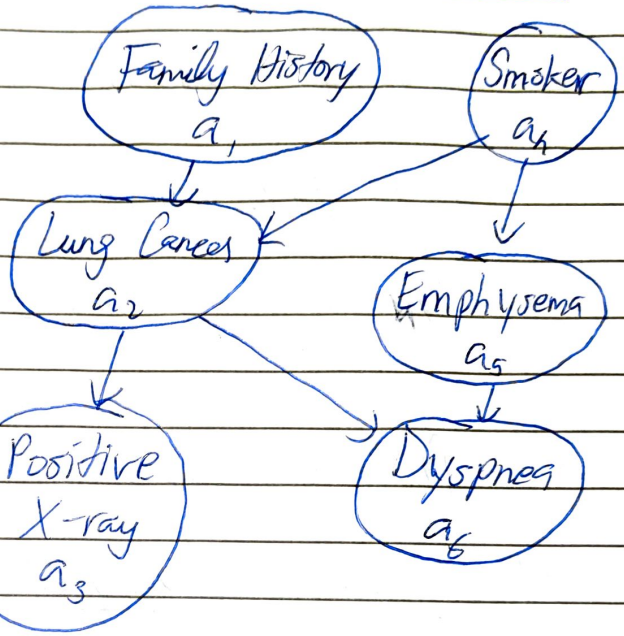
- Represented dependencies and independencies explicitly via a graph.
- Shows which variables depend on which others and uses conditional probability to express those relationships.
- It is a Directed Acyclic Graph containing nodes (random variable) connected by edges (conditional dependency cause → effect, arrows show direction of influence)
- Each node has a Conditional Probability Table (CPT) that tells how strong the relationships are (for root nodes, CPT = prior probability)

- On the other hand, simple Naive Bayes ~~model~~ is a simplified network where each feature depends only on the class, not other features (independent)
- Class variable is a single parent of all feature nodes

Naive Bayes



Bayesian Belief Network



→ P(Family History = T) = 0.20

Family History	P
T	0.20
F	0.80

→ P(Smokes = T) = 0.30

Smokes	P
T	0.80
F	0.70

→ P(Lung Cancer = T | Family History = F, Smoker = T) = 0.15

P(LC = T / FH = T, S = F) = 0.25

P(LC = F / FH = F, S = F) = 0.98

P(LC = F / FH = T, S = T) = 0.62

Family History	Smokes	P(Lung Cancer = T)	P(Lung Cancer = F)
F	F	0.02	0.98
F	T	0.15	0.85
T	F	0.25	0.75
T	T	0.38	0.62

→ $P(Emphysema = T / Smokes = T) = 0.20$
$P(Emphysema = F / Smokes = F) = 1$

Smokes	P(E = T)	P(E = F)
F	0	1.00
T	0.20	0.80

→ $P(Dyspnea = T / LC = T \; E = F) = 0.30$
$P(Dyspnea = T / LC = F, E = T) = 0.50$
$P(Dyspnea = F / LC = F, E = F) = 0.95$
$P(Dyspnea = F / LC = T, E = T) = 0.40$

Lung Cancer	Emphysema	P(Dyspnea = T)	P(D = F)
F	F	0.05	0.95
F	T	0.50	0.50
T	F	0.30	0.70
T	T	0.60	0.40

→ $P(PositiveXRay = T / LungCancer = T) = 0.85$
$P(PositiveXRay = T / LungCancer = F) = 0.10$

Lung Cancer	P(PositiveXRay = T)	P(PositiveXRay = F)
F	0.10	0.90
T	0.85	0.15

- Goal is to find total probability of $P(PositiveXRay = T)$

$P(PositiveXRay = T) = P(PositiveXRay = T / LungCancer = T)$
$P(LungCancer = T) +$
$P(PositiveXRay = T / LungCancer = F)$
$P(LungCancer = F)$

(Marginal Probability of Lung Cancer + CPT for Positive X Ray)

$P(\text{Lung Cancer} = T) = P(LC = T / FH, S) \cdot P(FH) \cdot P(S)$

$= 0.02(0.8)(0.7) + 0.15(0.8)(0.3) + 0.28(0.2)(0.7)$
$+ 0.38(0.2)(0.3)$
$= 0.105$

$P(\text{Lung Cancer} = F) = 1 - P(\text{Lung Cancer} = T)$
$= 0.895$

$P(\text{Positive X Ray} = T) = (0.85)(0.105) + (0.10)(0.895)$
$= 0.17875$

ENSEMBLE LEARNING

- Combines multiple individual models to create a stronger more accurate predictive model.
- The individual models that we combine are known as weak learners which either have high bias or high variance, cannot learn efficiently and perform poorly.

Max Voting

- Generally used for classification problems where each model's prediction is considered as a vote.
- The predictions we get from majority of models is considered as the final prediction

Averaging

- We take the average of predictions from all models and use it to make the final prediction.
- Can be used in both regression and classification (probabilities) problems

Weighted Average

- All models are assigned a weight based on the importance of each model in making the prediction.
- All these approaches help in reducing bias and variance of weak learners thus improving the overall accuracy of the final strong learner

- Instead of modifying / combining the predictions itself, there are certain advanced ensemble methods that combine multiple base models to create a new model & how to improve them iteratively.

★ Bagging (Bootstrap Aggregating)

- Designed to improve stability and accuracy of ML algorithms by combining multiple homogeneous weak learners that learn from each other independently in parallel to determine an aggregated result / prediction.

- Decreases variance, thus helps avoid overfitting. Usually applied to decision tree methods.

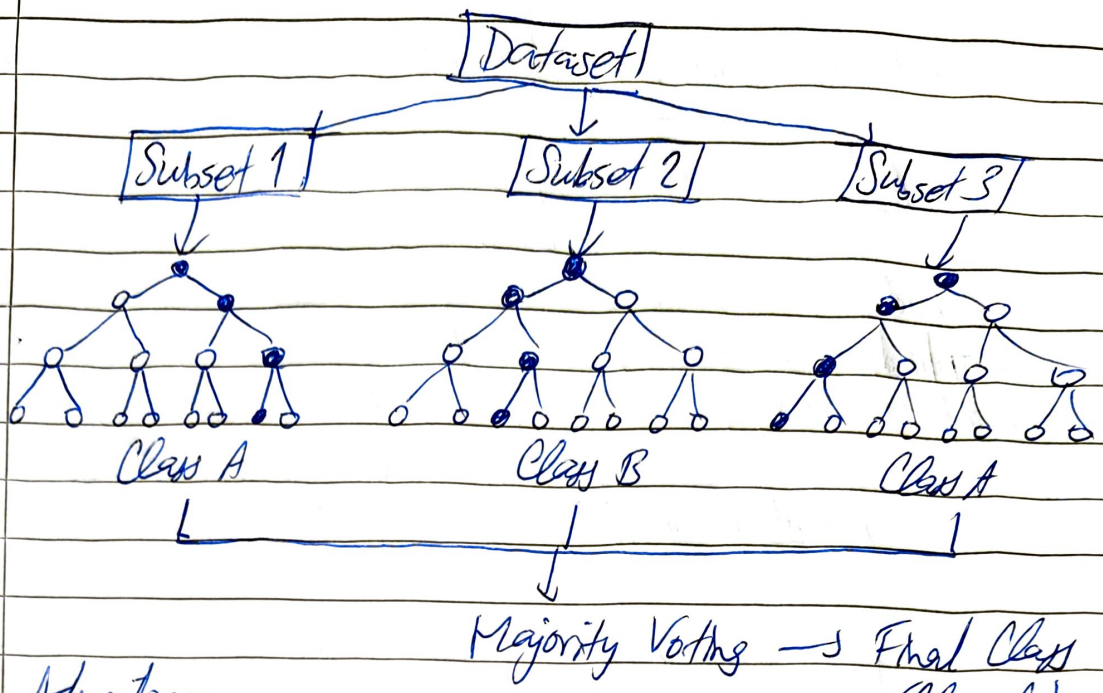
① Multiple subsets created from original dataset by selecting observations with replacement.

② Base models created on each of these subsets.

③ Each model is learning in parallel with each training set, independent of each other.

④ Final predictions are determined by combining the prediction from all models. (majority voting for classification or averaging for regression)

- Random Forest models use Bagging where decision tree models with higher variance are present. Several random trees make a Random Forest



- Advantages

- Reduces overfitting through random sampling of data.
- Performs well on large datasets with high-dimensions.
- Works for both classification and regression tasks.
- Handles missing values and noisy data efficiently

- Limitations

- Difficult to interpret due to ensemble complexity
- Slow training and prediction compared to a single decision tree
- May overfit if too many trees / irrelevant features.
- Requires more memory and computational resources.

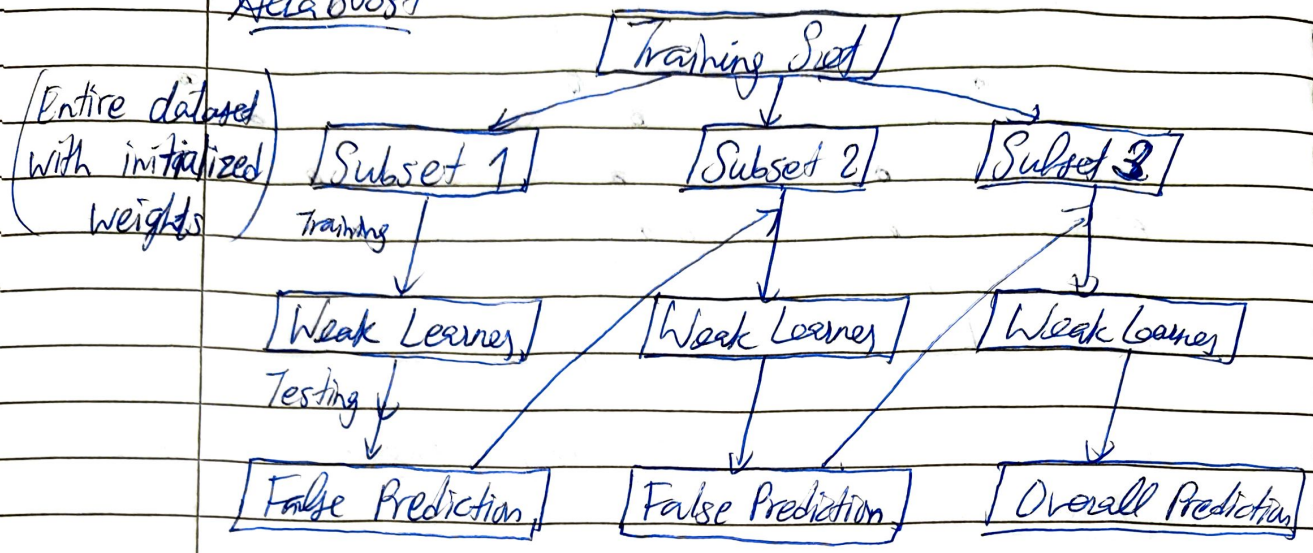
Boosting

- Attempts to build a strong classifier by using multiple weak classifiers in series.

- You train weak models one after another such that each one fixes the errors of the previous one

- Here models are trained sequentially to reduce bias

— AdaBoost



① Initialize sample weights for every training sample equally, $w_i = \frac{1}{N}$ for N samples.

② Train a weak learner on all datapoints and note the predictions made for each sample.

③ If a sample is classified correctly, decrease its weight (nothing to fix here)
If a sample is classified wrong, increase its weight (new model should care about this more)
Normalize the weights, $\sum w_i = 1$

④ Each weak learner gets a model weight based on how accurate it was (Mistakes $\propto \frac{1}{\alpha}$)

$$\alpha = \frac{1}{2} \ln\left(\frac{1-\text{error}}{\text{error}}\right)$$ (how trustworthy that weak model is)

This way by combining, better ones have more say in the final prediction

⑤ Each new models learns from the updated sample weights

Final output = $\text{sign}\left(\sum_t \alpha_t h_t(x)\right)$ → prediction of weak learner t.

— XGBoost

- XGBoost = Gradient Boosting + Second-Order Optimization (Hessian) + Regularization.

① Initialize the model with a basic prediction (for regression, average of all target values)

② Compute the residuals (error) between actual and predicted values, these represent what the next new tree should learn.

③ Fit the decision tree to the residuals, this predicts how to correct the previous errors.

④ Calculate the leaf (correction value) weights for the decision tree that group similar residuals together.

$$\text{Leaf weights} = -\frac{\text{Sum (gradients)}}{\text{Sum (hessians)}}$$

Based on the leaf weights, increase / decrease predictions by those weights * learning rate for those ~~input features~~ leaves. (to avoid overfitting)

⑤ Update the predictions and repeat until error is minimal

- Hence, each tree is just adding a small correction to the previous prediction, to finally build an overall robust model.

(Leaves get one correction value that represent how much to fix those residuals)

Note: Gradients tell which direction and how strongly we should change the prediction (advanced residual)
Hessian tells how confident/steep the loss changes if we move in that direction (curvature)

Note: For squared errors, residuals = actual − predicted
gradient = pred − actual
Hessian = 1

$$\text{Residual} = y - \hat{y}$$

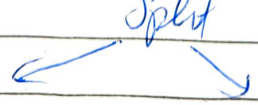
$$\text{Gradient} = \frac{\partial L}{\partial \hat{y}} = \frac{\partial}{\partial \hat{y}}\left(\frac{1}{2}(\hat{y}-y)^2\right) = \hat{y} - y$$

$$\text{Hessian} = \frac{\partial^2 L}{\partial \hat{y}^2} = 1$$

Now XGBoost measures gain for each split made in the tree, if the split groups samples with similar gradients, the model can correct them more effectively (high gain)

$$\text{Gain} = \frac{1}{2}\left(\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} + \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}\right) - \gamma$$

where G_L, G_R → sum of gradients in left, right
H_L, H_R → sum of hessians.
λ → regularization term to prevent overfitting
γ → minimum gain threshold for pruning.
Then, for the ideal split, leaf weights are computed and correction value obtained (by multiplying with η)



Leaf L
[+10, −30]
weight = −10
correction = −1

Leaf R
[−10, +30]
weight = +10
correction = +1

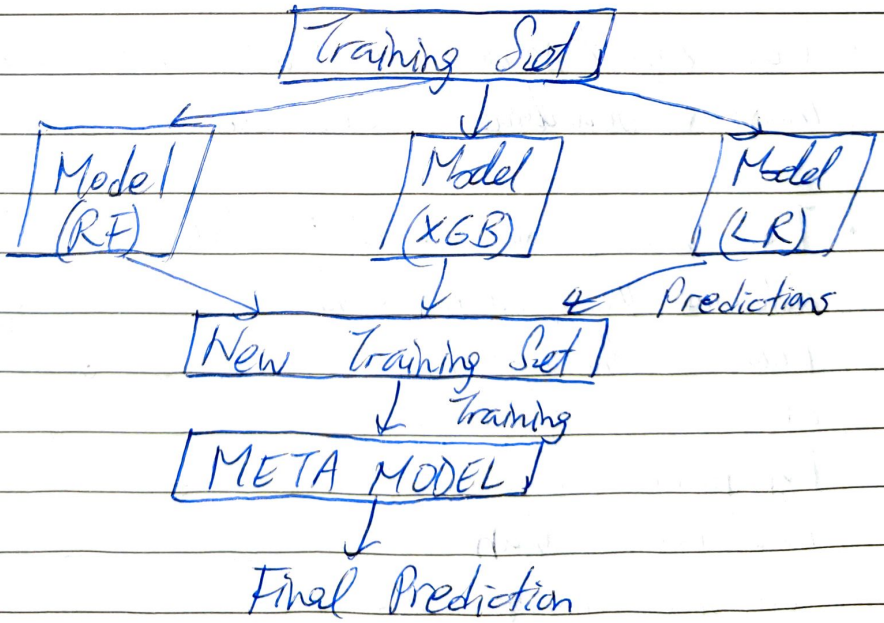
Stacking

- To improve prediction accuracy of strong learners (heterogeneous) to create a single robust model

- Bagging → many weak, similar models (vote)
  Boosting → many weak, sequential models correct each other
  Stacking → multiple strong, different models working together

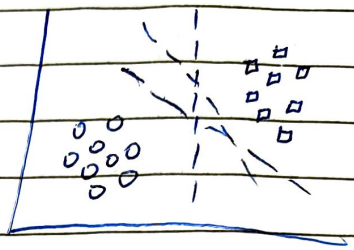
- Stacking does not combine predictions directly, it learns how to combine them through another meta model.

1. Train multiple models on the same training data
2. Use their predictions to create a new dataset where input features = predictions from each model and target = actual value from original dataset.
3. Train the meta model on this new dataset, which learns which model to trust more by assigning weights to each model.
4. Finally, the meta-model outputs the final prediction.

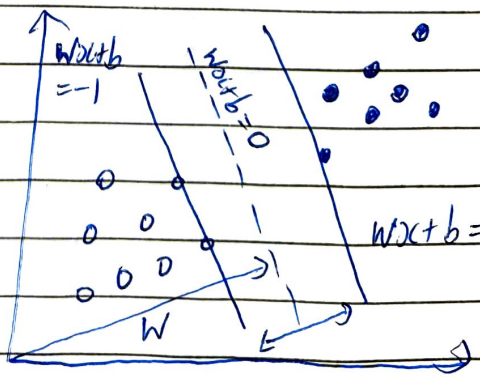


SUPPORT VECTOR MACHINES

- Main goal of Logistic Regression is to find the optimal hyperplane that best seperates different classes
- However there is a confusion of which is the best classifier (multiple solutions exist)



- SVM seeks a unique solution by maximizing the margin (minimum distance between hyperplane & samples of both classes closest to the hyperplane) (support vectors)
- Works well for both linearly seperable and ~~inseperable~~ inseperable datasets, thus ~~providing~~ promising good generalization to unseen data.
- Margin is the gap between the hyperplane and the support vectors (samples closest to hyperplane)
- These support vectors are points that touch or cross the margin boundaries. (critical for defining the boundary)
- These few points are what determine the position, angle, width of the margin, rest all do not affect the hyperplane, making it more robust than logistic regression that considers all points in determine the boundary (including outliers)



$\|w\| \rightarrow \sqrt{w_1^2 + w_2^2 + \dots}$

If class 1 corresponds to 1
class 2 corresponds to -1

$wx_i + b \geq 1$, $\forall x_i$ with $y_i = 1$ (Class 1)
$wx_i + b \leq -1$, $\forall x_i$ with $y_i = -1$ (Class 2)
$wx_i + b = 0 \rightarrow$ decision boundary / hyperplane
$wx + b = 1$, $wx + b = -1 \rightarrow$ support vector boundaries

$y_i(wx_i + b) \geq 1$, $\forall x_i$ (constraint for perfect classification)

- We want to either

maximize $\frac{2}{\|w\|}$ or minimize $\frac{1}{2}\|w\|^2$

- Hard-Margin SVM aims to find w, b that solves $\min \frac{1}{2}\|w\|^2$ such that $y_i(w \cdot x_i + b) \geq 1$, $\forall x_i$

- It's problem is convex, there is a global unique min value

- Soft-Margin SVM allows some instances to fall within the margin (slack variables) but penalizes them.

$y_i(w \cdot x_i + b) \geq 1 - \xi_i$ $\forall x_i$ $\xi_i \geq 0$

(Minimize $\frac{1}{2}\|w\|^2$) + $C \sum \xi_i$

- Algorithm tries to maintain $\xi \rightarrow 0$ while maximizing margin but does not minimize no. of misclassifications

- $\xi \rightarrow$ distance into the margin
If point is correctly classified, $\xi = 0$
If point is inside the margin, $0 < \xi < 1$

If point is beyond the margin, misclassified, $\xi > 1$

- SVM tries to keep $\xi = 0$ (no violations), keep $||w||$ small (wider margin) and balance between them using C (higher the stricter, more penalty)

- Large $||w|| \rightarrow$ small margin
  Small $||w|| \rightarrow$ larger margin
  Margin $= \frac{2}{||w||} \rightarrow$ distance between margin boundaries (maximize)

- $\xi_i = \max(0, 1 - y_i(w \cdot x_i + b))$

- Sometimes data is not linearly separable in the original space, hence it is better to map them to higher-dimension to be able to linearly classify them.

- Hence we find function $\phi(x)$ to map to a different space, then SVM formulation becomes:

  $\min \frac{1}{2}||w||^2 + C\sum \xi_i$ such that

  $(y_i(w\phi(x) + b) \geq 1 - \xi_i) \; \forall x_i, \quad \xi_i \geq 0$

- We use the kernel trick to avoid explicitly mapping data into higher-dimensional space through dot product.

- One-vs-All SVMs: For n classes, train n binary classifiers each distinguishing one class from the rest. The classifier with the highest distance from the hyperplane (margin width) determines the predicted class.

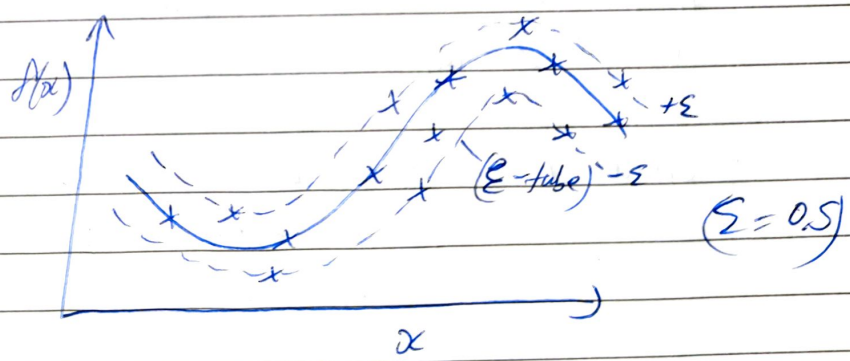
- One-vs-One SVM: Train $n(n-1)/2$ classifiers each discriminating between a pair of classes. Each classifier casts a vote and the class with the most votes wins. (binary SVMs)

☆ Support Vector Regression (SVR)

- In SVC, outputs were discrete $(-1, 1)$, goal was to find the maximum margin hyperplane that seperates classes. Errors were treated as misclassifications if points fall on the wrong side of the margin.

- In SVR, outputs are continuous, real value, here errors of approximation replaces the margin.

- Here, we define an ε-insensitive tube around the regression function $\hat{y} = f(x)$ such the predictions within this tube are considered good-enough (no penality)



- $\varepsilon \rightarrow$ width of tube / tolerance band

- Points inside the tube → no penality
  Points outside the tube → penalized by slack variable ξ.

- Large ε → wider tube, small deviations ignored
  → less sensitive to fluctuations
  → low accuracy on training data
  → better generalization to unseen data.
  → reduces risk of overfitting
  → may underfit if ε too large.

- Small ε → narrow tube, models fits very close to training data.
  → high accuracy on training data
  → sensitive to noise, overfitting
  → $f(x)$ bends frequently to stay close to points.

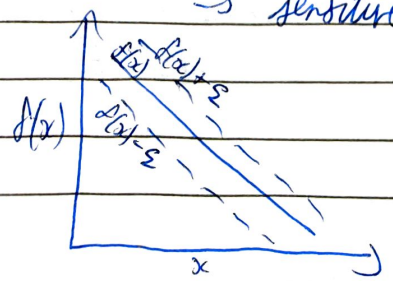
- Loss = $\begin{cases} 0 & \text{if } |y - f(x)| \leq \varepsilon \text{ (within tube)} \\ |y - f(x)| - \varepsilon & \text{otherwise (linear penalty)} \end{cases}$

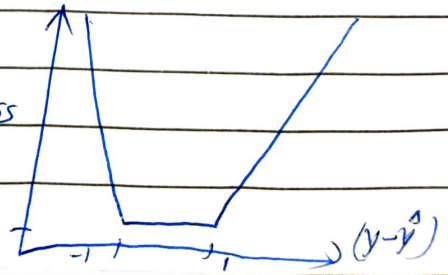
- Objective → $\min \frac{1}{2}\|w\|^2 + C \sum$ losses outside tube

- $\|w\|^2$ measure model complexity
  C → tradeoff between model complexity and variance (cost of violation)

- Small C → model does not punish deviations beyond ε much (wider tolerance for deviation)
  → better generalization, may underfit.

- Large C → very strict with violations, strong pressure to fit training data, higher $\|w\|$
  → high accuracy on training data, overfitting
  → sensitive to noise





★ Ranking

- Task of reordering a list of objects based on their predicted relevance to a given query or context.

- Classification → predicts a label (relevant or not)
  Regression → predicts a score (0.85% relevant)
  Ranking → predicts a permutation.

- Learning to Rank (L2R) frameworks (how search engines and recommendation systems learn to sort items in best order) rank stuff in 3 different ways:

(a) Point-wise: Sort all items by their individual relevance score (classification / regression)

(b) Pair-wise: Binary classification on pairs, learning focuses on relative order. (RankBoost)

(c) List-wise: Model sees whole ranked list and optimizes a ranking metric directly

- Evaluation metrics for ranking include:

(a) Non-Position-Aware Metrics

$$\text{Precision @ K} = \frac{|\{\text{relevant items in top K}\}|}{K}$$

(fraction of items in top K positions that are relevant
(does not care about order

(b) Position-Aware Metrics

rel_i → relevance score of item at position i

Discounted Cumulative Gain @ K, $DCG@K = \sum_{i=1}^{K} \frac{2^{rel_i} - 1}{\log_2(i+1)}$

(rewards relevant items, penalizes them more they appear down the ranking)

Normalized $DCG@K = \frac{DCG@K}{IDCG@K}$ (ideal sorting in decreasing relevance)

(Score of ranking perfection)

- RankBoost is a classic L2R algorithm based on AdaBoost framework.

- Instead of a simple classification problem, RankBoost converts ranking into a preference problem between 2 classes (pairs of items) (binary classification on pairs)

$$\hat{y} = \begin{cases} +1 & \text{if item } i \text{ ranked before item } j \\ -1 & \text{if item } j \text{ ranked before item } i \end{cases}$$

(pairwise - comparison)

__/__/__

RECOMMENDATION SYSTEMS

- Information - filtering systems to predict what the user likes (estimate preference / utility)
- Increase revenue of companies, boost engagement and improves retention of users. (stays loyal)
- Let U = Users, I = Items
  We define a utility function : $U \times I \rightarrow R$
  where $R \rightarrow$ rating (1-5 stars, like/dislike)
- Since real-world data forms a sparse $(U \times I)$ matrix with missing entries, task is to predict them.
- Recommenders depend on :

(i) Explicit feedback (directly given by users)
(ratings, reviews, like/dislike)

(ii) Implicit feedback (observed behaviour)
(clicks / views / search history / watch time)

- Key challenges faced by recommendation systems include:

(i) Sparsity (most users rate less items, matrix mostly empty)

(ii) New users (no history to recommend from)
New item (No ratings, don't know who to show to)

(iii) Scalability (must handle millions of users + items efficiently)

(iv) Lack of Diversity (may recommend same kind of content, reduces discovery)

★ Content-Based Filtering

- Focuses on Item's attributes and User's past tastes.
- User-Independent (specific), don't need data from other users.

① Define an Item Profile Vector V_i, extract relevant item attributes (features.

Ex: Movie : [Genre, Director, Actors]

② • Use TF-IDF for handling Textual features.
- Term Frequency (TF) (how often word appears in an item)
- Inverse Document Frequency (IDF) (penalizes very frequent words)

Together they provide a vector where important distinguishing words get higher weight.
Helps the system understand what the item is truly about.

② Build a single User Profile Vector P_u which represents the sum/average of item profile vectors for all items the user rated highly.

$$P_u = \sum_{\text{(rating)}} r_i \cdot V_i$$

③ Prediction of Similar items using Cosine Similarity (V_j) (un-rated)

$$\text{Similarity}(P_u, V_j) = \frac{P_u \cdot V_j}{\|P_u\| \|V_j\|}$$

(common in high-dimensional spaces)

- Cosine similarity measures the cosine of the angle between two vectors (−1 → opposite (not similar), +1 → identical direction)

- As a result, the system ranks unrated items by this similarity score and recommends top K items.

(Pros)
- Mitigates new user problem
- Handles sparse data well since it focuses on content features, not just ratings.
- System can easily explain why recommendation was made (interpretable)

(Cons)
- Too dependant on metadata (new item problem)
- Requires significant domain knowledge and manual effort to extract and process meaningful features.
- Over-specialization, recommends too similar items, struggles to suggest diverse items.

☆ User-Based Collaborative Filtering

- Uses behaviour of similar users.

① Calculate User-User Similarity (sim(u,v))

- Measure similarity between active user (u) and every other user (v) on items both have rated

- Pearson correlation is used as the metric since it corrects for user rating bias by normalizing the ratings by subtracting the user's average rating.

② Identify the K-Nearest Neighbours (top K- users v most similar to user u)

③ Generate prediction for rating of unrated item by taking weighted average of K-neighbour's ratings on that item (here r_u → user's average rating)

$$\hat{r}_{u,i} = r_u + \frac{\sum sim(u,v) \cdot (r_{v,i} - r_v)}{\sum |sim(u,v)|}$$

- There also exists item-based collaborative filtering, which scales better, saving massive computational effort.

(Pros)
- Diversity introduced
- No manual feature engineering required.

(Cons)
- New user problem
- Scalability of UBCF (computational complexity $O(n^2)$)
- Sparsity.

Citation & Graph-based Methods

- Instead of ratings / features, use citation links
- Used in research-papers, patents, web-search.

Co-Citation Similarity (Static)

- If Document A and Document B both appear in the reference list of Document C then A and B are co-cited (similar)
- Strength of similarity depends on no. of distinct documents that cite both of them.

Bibliographic Coupling

- If Document A and Document B both include Document C (common reference, cited same paper) then they are bibliographically coupled.
- Strength of coupling depends on no. of same papers that both cited together (common).

★ PageRank Algorithm

- Classic link-analysis algorithm that determines the importance or authority of nodes in a graph.
- The web (or collection of documents) is modelled as a directed graph where nodes are webpages / documents / items and edges are hyperlinks / citations / references
- A page's importance is determined by the importance of the pages that link to it.
- PageRank (PR) is the probability that the random surfer (who clicks random links on a page) will be on that page after a certain amount of clicks.

$$PR(A) = (1-d) + d \sum_{T_i \in B_A} \frac{PR(T_i)}{L(T_i)}$$

where $PR(A)$ → Importance score of page A
B_A → set of backlinks T_i that lead to A (A's incoming links)

$L(T_i)$ → no. of outgoing links from T_i.

- d → Damping Factor (probability that user follows the link on the current page, rather than randomly jumping to new page $(1-d)$)

Assuming random surfer keeps clicking links from page to another, there are possibilities of landing on a page with no outgoing links (dangling node) or trapped in a loop. Hence damping factor is introduced to simulate behavior of real web user.

① Every page starts with the same PR score $\left(\frac{1}{N}\right)$ for N pages. (equal probability of being visited initially)

② Iteratively update all pages by redistributing rank scores through the network of links, until convergence (change in pagerank value < threshold)

HITS Algorithm (Hyperlink-Induced Topic Search)

- Used for ranking pages specifically for a given search query, by assigning two scores that are mutually recursive. (until convergence)

① Authority Score (a_p): Measures quality of content of page p

Sum of Hub scores of all pages that link to p.

$$\left(a_p \propto \sum_{q \to p} h_q\right)$$

② Hub Score (h_p): Measures quality of links on page p.

- Sum of authority scores of all pages p links p to.

$$h_p \propto \sum_{p \to q} a_q$$

★ Reverse Search

- A search that finds the source or context of a specific item (image)
- Feature extraction from input → Indexing / Matching.

★ Timed-PageRank

- Standard PR treats all links equally regardless of when they were created or when link structure was updated.
- Introduces time decay factor where newer links or recently updated pages are more weighted:

$$PR(p,t) = (1-d) + d \sum \frac{P(q,t-1)}{OutDegree(q)} \times F(Age)$$

- Captures current importance and freshness of content well.

★ Model-Based Collaborative Filtering

- User-based and Item-based CF were memory-based

— Using Association Rules (AR)

- Finding relationships between items in large datasets

(Metrics)
- Support: How often items A and B occur together.

- Confidence: ~~How much more likely~~ P(buying B | A was bought)
- Lift: P(A, B bought together by chance)

— Using Matrix Factorization (MF)

- Decompose the ~~sparse~~ sparse U×I matrix into two lower-dimensional matrices of latent factors (which are hidden factors that explain observed ratings)

$$U \times I \approx (U \times LF) \times (LF \times I)$$

- Learn values in matrix by minimizing error between predicted rating and actual rating.

Association	Matrix Factorization
• Binary data (output)	• Ratings (continuous output value)
• Finds local relationships (item A → item B)	• Finds global relationships based on latent features.
• Simple, easy to interpret rules.	• Mathematically-complex,
• Computationally expensive for large datasets.	• Highly scalable, better for large datasets
• Sensitive to high sparsity	• Better at handling sparsity