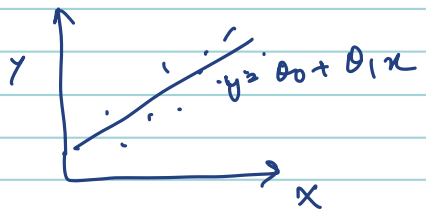
Advanced Machine Learning

▼ **Advanced Machine Learning (guarantor Christina Bauer)**

- Linear and polynomial regression: Explain how the squared error cost function is applied to linear/ polynomial regression and how the gradient descent algorithm is used to solve the optimization objective.
- Logistic regression: Explain the difference of logistic regression and linear regression in terms of the used hypothesis, cost function, and optimization objective.
- Neural networks: Describe the model structure of a neural network and how forward and backward propagation is used to solve the optimization objective.
- Support vector machines: Explain the difference of SVM to logistic regression and how kernels are used in this context.
- Evaluating learning algorithms: Describe the bias and variance problem and how regularization, cross-validation, and other methods can be used to solve for these problems

① Linear Regression

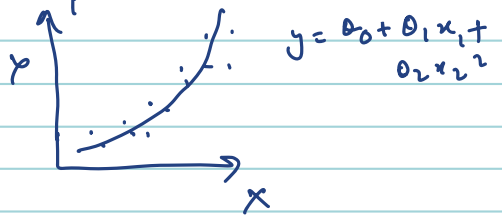
- Statistical method to model rel. b/w dependent variable and one or more independent variables by fitting a linear equation
- $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_2 \dots \theta_n x_n$ hypothesis function:
- suitable for linear trends
- few parameters simpler & less prone to overfitting
- not suitable for non linear data.



- eg: • Predicting house prices
  size ↑, price ↑

Polynomial Regression

- Extension of linear Regression when the relation b/w dependent and independent variable is modeled as polynomial function.
- $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1 x_2 + \theta_4 x_4 + \theta_5 x_4^2 x_2 \dots$
- Suitable for non linear trends
- More parameters, complex, More prone to overfitting
- Suitable for non linear complex data



- eg: • Crop yield based on temperate, soil, rainfall
  • Growth of bacteria over time.

- Objective is same for both, to find optimal θ values (params) to minimize the error b/w predicted and target variable.

MSE → used in Regression to quantify error b/w predicted and target values/outputs.

hypothesis function: $h_\theta(x) = \theta_0 + \theta_1 x$

cost function: $J(\theta_0, \theta_1) \quad \frac{1}{2m} \sum_{i=1}^{m} \underbrace{(h_\theta(x) - y^i)^2}_{\text{predicted error}}$ ← actual target

squared error ensures that both +ve & -ve errors are treated equally.

Gradient Descent: optimization algorithm to find the optimal value for θ (params) to minimize the cost function.
It iteratively updates the parameters in the direction of the negative gradient.

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

Initialize with random params
↓
Compute $h_\theta(x^i)$
compute error
calculate gradient
Update params
Repeat steps

GD update rule:

$$\theta_j := \theta_j - \alpha \frac{\partial J(\theta)}{\partial(\theta_j)}$$

Gradient for param θ_j

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^i) - y^i) x_j^i$$

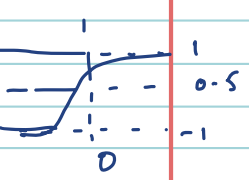
for polynomial include ↑ power $x^2, x^3, x^4 \ldots$

In linear, cost function is convex
∴ ensures convergence to global minima.

In polynomial, cost function is non convex, can get stuck at local minima

Logistic Regression: Statistical model used for binary classification problems.
Model estimates the probability that a given input belongs to certain class.

hypothesis function: Sigmoid function (logistic) to map input features to probability value of 0 & 1.



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

$$0 < h_\theta(x) < 1$$

o/p dependent feature $y = \begin{cases} 0 & \text{if class 1} \\ 1 & \text{if class 2} \end{cases}$

Independent features $\begin{bmatrix} x_{11} & x_{21} & \cdots & x_{m1} \\ x_{12} & x_{22} & \cdots & x_{m2} \\ \vdots & \vdots & \cdots & \vdots \\ x_{1n} & x_{2n} & \cdots & x_{mn} \end{bmatrix}$ $m \times n$

for linear Regression: $h_\theta(x) = \beta_0 + \beta_1 x$

for logistic Regression: $\sigma(t) = \sigma(\beta_0 + \beta_1 x)$

t = linear combination of inputs & model params.

t = input to sigmoid function

Decision boundary:

Setting up a threshold (common 0.5) is set to classify

if $h_\theta(x) > 0.5$ classify 1

$h_\theta(x) < 0.5$ classify 0

$$t = \beta_0 + \beta_1 x$$

$h_\theta(x) =$ sigmoid (t)

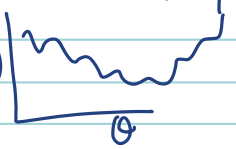
$$h_\theta(x) = \frac{1}{1+e^{-t}}$$

$$h_\theta(x) = \frac{1}{1+e^{-(\beta_0+\beta_1 x)}}$$

$$h_\theta(x) = \sigma(w^T x + b)$$

w^T = transp feature matrix

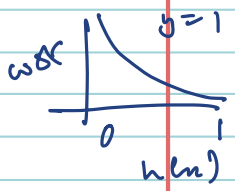
The linear regression MSE cost function doesn't work in logistic regression, as it results in non convex graph

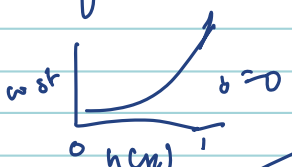


Cost function in Logistic Regression

↳ **Log loss function** (Binary cross entropy loss)

tailored for Binary classification





$$\text{Cost function } (h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y=1 \\ -\log(1-h_\theta(x)) & \text{if } y=0 \end{cases}$$

two equation can be compressed into 1

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[\underbrace{\log(h_\theta(x_i))\, y_i}_{\text{cross entropy +ve class}} + \underbrace{(1-y_i)\log(1-h_\theta(x_i))}_{\text{-ve class}}\right]$$

y = actual label

$h_\theta(x)$ = predicted prob of i^{th} class

Gradient Descent: Reduce cost value

cost function

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y_i \log(h_\theta(x_i)) + (1-y_i)\log(1-h_\theta(x_i))\right]$$

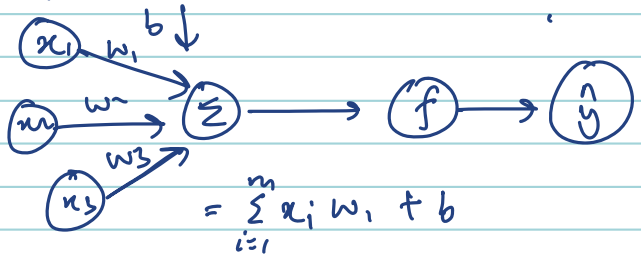
$h_\theta(x_i) = \hat{y} = \sigma(w^T x + b)$ $\sigma(x) = \frac{1}{1+e^{-x}}$

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

Aspect	Linear Regression	Logistic Regression
Hypothesis Function	$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$	$h_\theta(x) = \frac{1}{1+e^{-(\theta_0+\theta_1 x_1+\cdots+\theta_n x_n)}}$
Cost Function	Mean Squared Error (MSE): $J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2$	Log-Loss (Binary Cross-Entropy): $J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y_i \log(h_\theta(x_i)) + (1-y_i)\log(1-h_\theta(x_i))]$
Gradient of Cost Function	$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)x_{ij}$	$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)\,x_{ij}$
Gradient Descent Update Rule	$\theta_j := \theta_j - \alpha\frac{\partial J(\theta)}{\partial \theta_j}$ MSE min	$\theta_j := \theta_j - \alpha\frac{\partial J(\theta)}{\partial \theta_j}$ Log loss min
Prediction Output	Continuous value (e.g., price, height)	Probability (0 to 1)
Convergence Behavior	Generally straightforward due to convex MSE cost function	Convex Log-Loss cost function, but sigmoid function introduces non-linearity
Feature Scaling Importance	Helps speed up convergence, especially with diverse feature ranges	Crucial for efficient convergence due to sigmoid scaling issues

(3) Neural Network

Class of ML models inspired from the structure & function of biological neuron in brain.



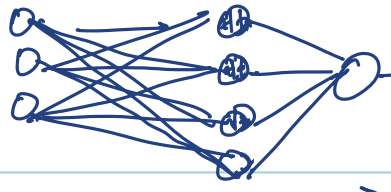
- Input layer
- hidden layer
- Output layer.

w = Represent strength of connection btw neuron

b = shifts activation function, improves models flexibility.

→ Forward Propogation:
process by which input data passes through the network and generate output.

Input → Hidden layer → Output



→ Predicted Value — Target value

↓

Loss function

I/p → I/p feature fed to network

Hidden → weight × Input + bias and Act fn is applied

O/p → computes output based on prob statement

Reg : continuous (single neuron)

class : neuron = no. of classes

↔ Example with linear Regression

Forward

compute output of n/w from given input

$h_\theta(x) = \theta_0 + \theta_1 x$

Backward

minimize the error by updating params

[involves finding gradient ∇ of cost (J) wrt to params]

Input: x – input
y – true value

hidden layer: compute weight
$z = \theta_0 + \theta_1 x$

Output: $h(\theta) = z$

Calculate cost: $\frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2$

Compute error:

Error = $h_\theta(x) - y$

Error = $(\theta_0 + \theta_1 x) - y$

Compute gradient for each θ_0 & θ_1

$\frac{\partial J(\theta)}{\partial \theta_0} = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)$ — bias (intercept)

$\frac{\partial J(\theta)}{\partial \theta_1} = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)(x_i)$ — slope (wt)

process iteratively continues until model converges to an optimal solution (multiple epochs)

update weight & Bias

$\theta_0 := \theta_0 - \alpha \frac{\partial J(\theta)}{\partial J\theta_0}$

$\theta_1 := \theta_1 - \alpha \frac{\partial J(\theta)}{\partial J(\theta_1)}$

α = learning rate

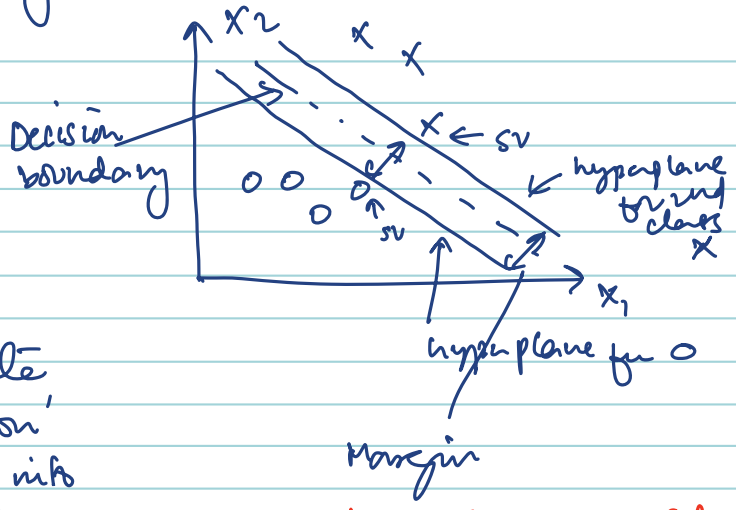
(ii) **SVM** → Support Vector Machines

Supervised machine learning algo. primarily for classification tasks.

Objective: Finds an optimal hyperplane that best separates the data points of different class in a high dimensional space.

maximizes the margin b/w the classes

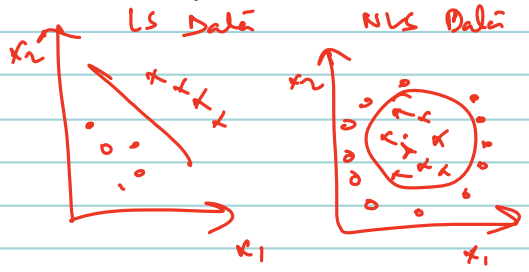
hyperplane – decision boundary
2D – line
3D – plane
higher dim: hyperplane

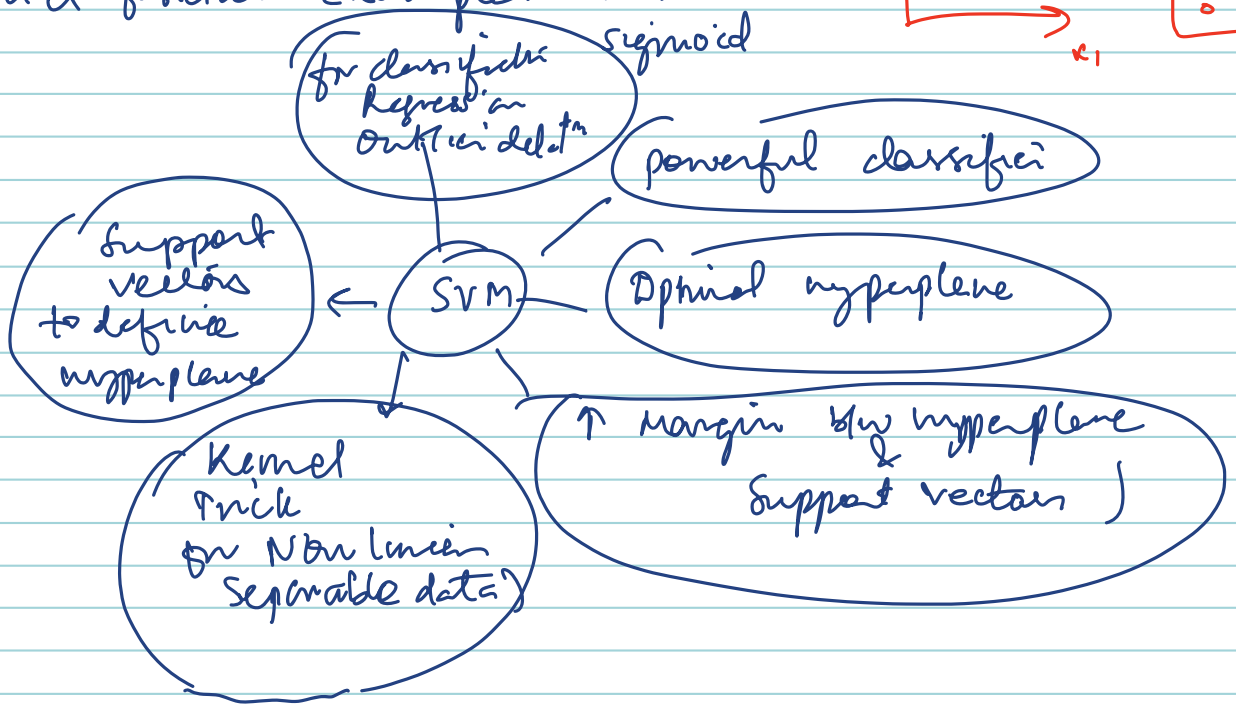


Kernel trick:

For non linearly separable data, SVM uses "Kernel function" to map the input features into a higher dimensional space where, linear hyperplane can separate the data

Kernel function example: RBF, sigmoid





SVM	Logistic Regression
can handle non linear & linear decision boundary (finds hyperplane)	① Linear decision boundary
② More robust at handling outliers	② Sensitive to outliers
③ Maximizes margin → optimization objective for better generalization	③ minimizes cost() [log loss] using algo like GD.
④ Computationally expensive	④ Simple implementation & less expensive

(5) effective in ↑ dimensional space

(6) hypothesis function: Uses sigmoid to output probabilities

(6) ↓ dim spaces (lines)

(7) Uses hyperplane to separate class

---

(5) Evaluating Learning Algorithm:

Bias: Difference b/w predicted and actual values.
→ ↑ Bias model pays little attention to training data & oversimplifies the model

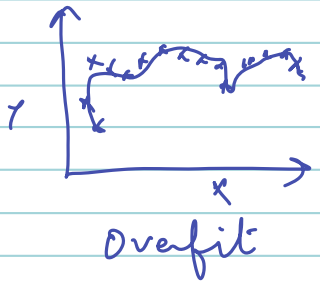
↑ Bias model too simple

Variance: Model's sensitivity to fluctuations in training data
↑ variance model → pays lot of attention to training data and learns noise & randomness in data leading to not able to generalize on test data (unseen)

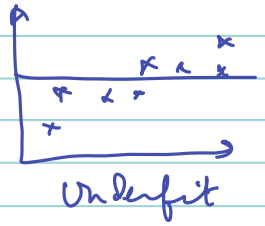
↑ variance model too complex

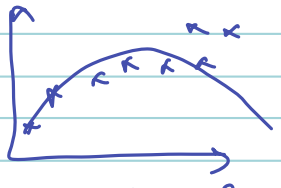
↑ Bias, ↓ variance → Underfitting — Performs bad on train & test data

↑ variance, ↓ Bias → Overfitting — Performs good on train & bad on test (unseen)



Overfit


Underfit


Good Balance.

Bias variance Tradeoff:

↑ model → simple and few params → ↑ Bias, ↓ variance (underfit)

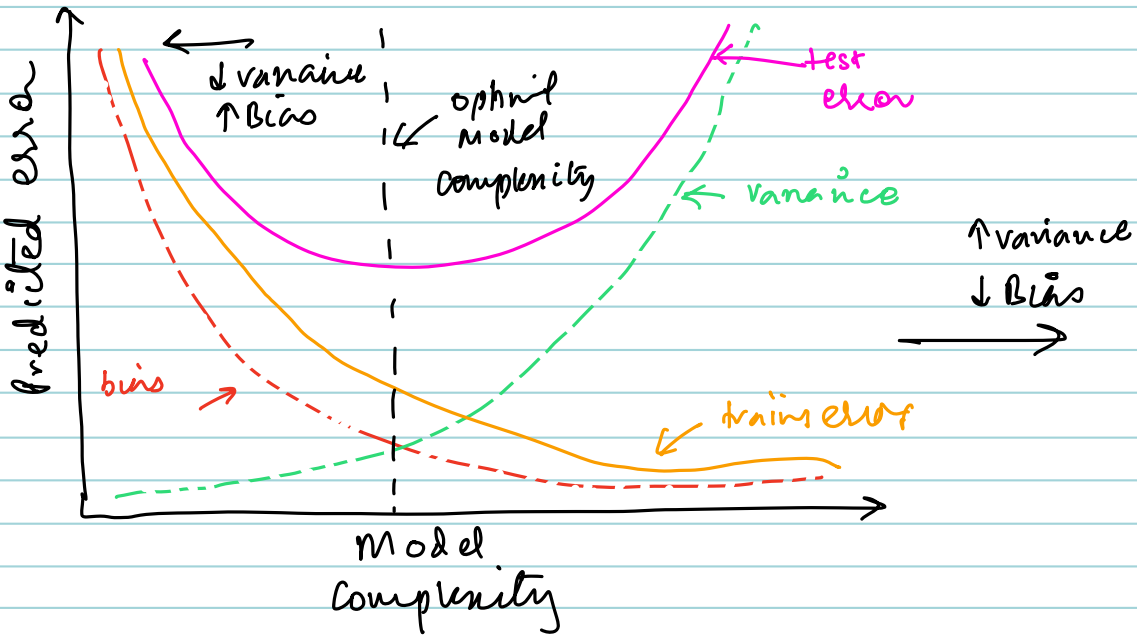
↑ model → complex and more params → ↓ Bias, ↑ variance (overfit)

- Goal is to have low Bias & low variance, fitting

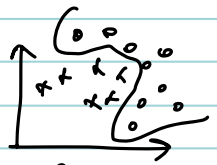
data well w/o being overly complex

For good model, good balance b/w bias & variance such as it minimizes total error

Total Error = $(\text{Bias})^2$ + Variance + Irreducible Error



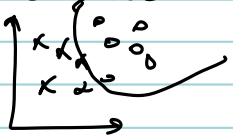
Regularisation :



↑ train accuracy

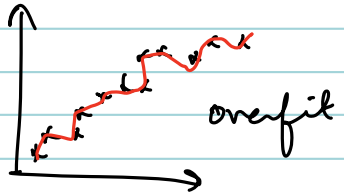
↓ test accuracy → overfitting

to avoid overfitting, we ideally want a smooth curve that can fit well in both train as well as test data set.



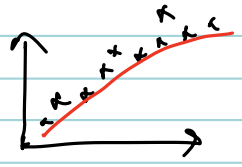
↓ use Regularisation

eg : Regression



$z = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_{500} x_{500}$

500 params

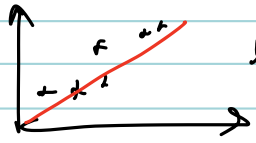


technique → called Regularisation

nullify affect of certain params/term to get smooth

term

$z = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4 \dots \theta_{500} x_{500}$



linear $z = \theta_0 + \theta_1 x$

2 params → straight line, we want more degree

Regularisation Methods: • L1 & L2 → (nullifying the affect of certain params / neurons ↑ linearity on model)
• Dropout
• Early stopping

Regularisation: ← Ignore less significant features
→ lower model complexity
→ computational expense reduce.
by scaling down / shrink regression coefficients
$y = 0.4 + 1.2x_1 + 20x_2 + 3.9x_3$
(x_1: less imp, x_3: imp)

L1 Regularisation
Lasso

Loss = Original loss + $\lambda \|w\|$
($\|w\|$: absolute value of coefficient)
Can lead to some coeff value → 0 for feature selection

L2 Regularisation
Ridge

L2 or Ridge = Loss + $\lambda \|w\|^2$
(Penalty)
$\|w\|^2 = w_1^2 + w_2^2 + w_3^2 + w_4^2 + \cdots + w_n^2$
as $\lambda \uparrow$, coefficient mag ↓
• adds sum of squared coeff to loss
• Reduces mag of coeff, but not = 0

Dropout Regularisation

Randomly dropping out (deactivating) a proportion of neuron during training,

helps in model to avoid relying too heavily on any single neuron and encourages robust feature engineering.

If dropout → 20%, then each neuron has 20% chance of being dropped out at every training step.

Early Stopping to avoid overfitting: (Regularisation technique)

Technique where, training process is halted once the model's performance on validation data starts degrading

CV (80.4, 78.5, 79.5, 82.4)
Robust
generalised
consistent

CV (68.2, 81.2, 72.1, diverse 52.2)
: not robust
. overfitting

Cross Validation

↓

To evaluate the model's performance to detect overfitting & underfitting

Dataset ÷ into 3 parts

↙ ↓ ↘

Training	Validate	Test
60%	20%	20%

Methods :- K fold Cross validation

All data → All data

→ Test fold → Training | Test

K folds ↓

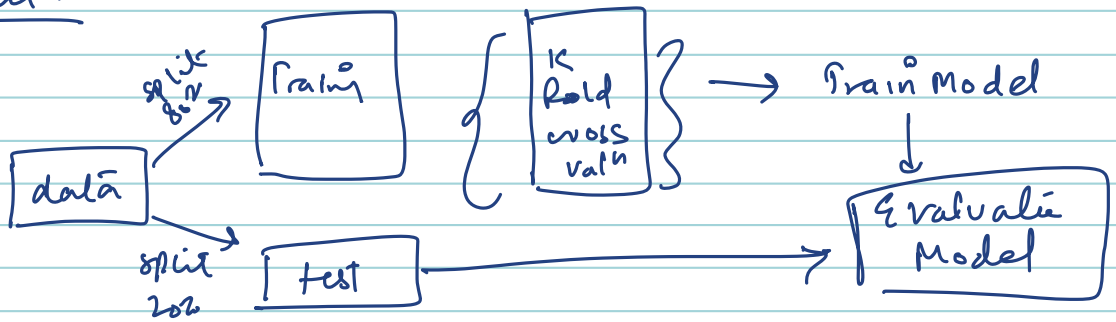
eg: for K = 5

Split 1	1	2	3	4	5	→ accuracy of Iter 1
split 2	1	2	3	4	5	→ Acc in 2
split 3	1	2	3	4	5	→ Acc in 3
split 4	1	2	3	4	5	→ Acc Iter 4
split 5	1	2	3	4	5	→ Acc Iter 5

average accuracy / score → Compare Acc

Test data → accuracy

Flowchart:



for K, one fold is test & remaining K-1 → training

Can be used for model selection

- Gives more robust estimate of models performance on unseen data & hence prevents overfitting

data efficient: every data point is used for training, validation, making most of available data.

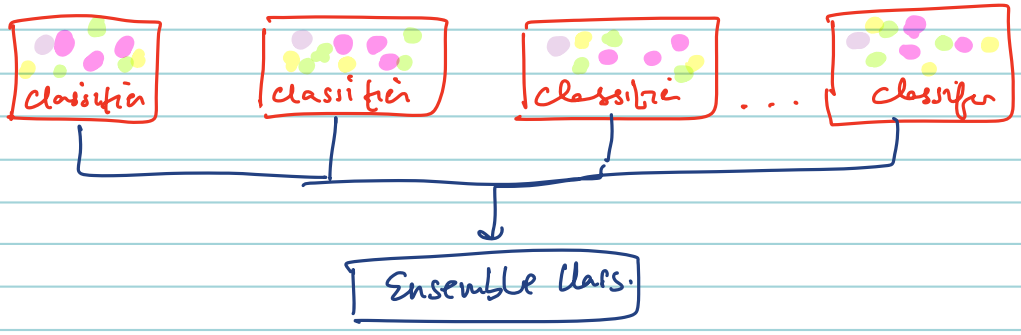
Ensemble Methods: Address Bias-Variance tradeoff by combining multiple models to improve overall performance.

to reduce variance

Bagging: Bootstrap + Aggregating

parallel

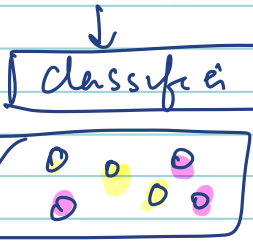
dataset



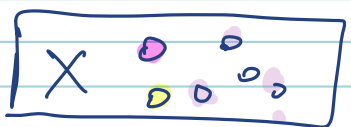
Create ← sub-dataset of equal size (Sampling with replacement) (Bootstrap)

Boosting

Sequential

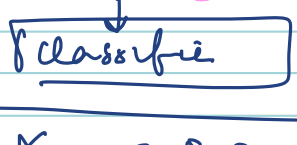


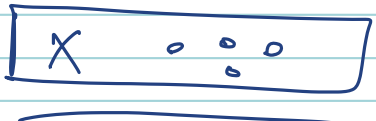
correctly classified



incorrectly classified

weightup the incorrectly → classified samples





. . . . . Ensemble classifier

until entire dataset is accurately predicted

Both are good at reducing variance as they combine multiple models

But for bias, Bagging has close to zero bias as no weight of data is change.

Boosting : weighting added ∴ Reduces Bias.
but ↑ prone to overfitting