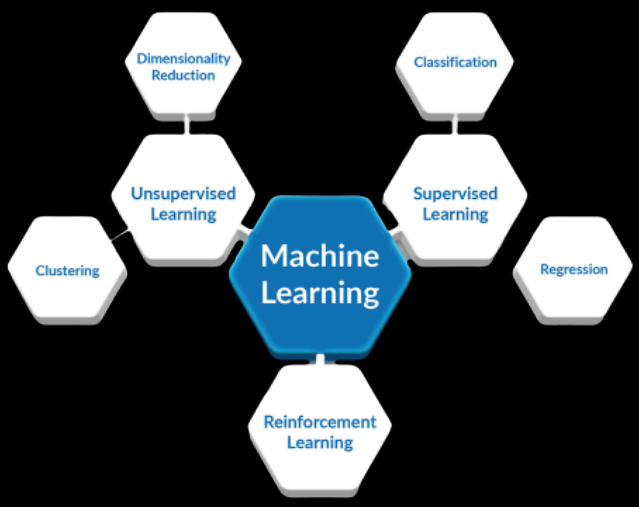
Machine Learning



Types of Machine Learning

Supervised	🔷 Regression	- Linear Regression - Ridge, Lasso Regression - Polynomial Regression - Support Vector Regression (SVR)
	🔷 Classification	- Logistic Regression - K-Nearest Neighbors (KNN) - Decision Tree - Support Vector Machine (SVM) - Naive Bayes
	🔷 Ensemble (Bagging)	- Random Forest - Bagging Classifier
	🔷 Ensemble (Boosting)	- AdaBoost - Gradient Boosting - XGBoost - LightGBM - CatBoost
Unsupervised	🔷 Clustering	- K-Means - DBSCAN - Hierarchical Clustering - Gaussian Mixture Models
	🔷 Dimensionality Reduction	- PCA (Principal Component Analysis) - t-SNE - UMAP - Truncated SVD

ML PROJECT CYCLE

1. Define the problem
   - Predict or Classify
   - Supervised, Unsupervised / Reinforced
2. Gather and Understand Data
   - Collect Data
   - Understand Data, distributions etc.
3. Data Preprocessing
   - Data Cleaning
     - missing Data
     - Duplicates
     - Data Types etc.
   - Handling Outliers (IQR or Z-score)
   - Feature Engineering (create features)
   - Feature Selection
     - Remove Redundant Columns
     - Use Lasso, VIF
   - Encode Categorical Variables
     - One hot Encoder (Nominal)
     - Label Encoding (Ordered)
   - Splitting Dataset (Train Test)
   - Feature Scaling (Standard, Minmax)
     Models = SVM, KNN, Logistic etc.
   - Balancing Dataset (for classification)
     Stratified Sampling, Class weights
4. Model
   - Model Selection
     - Regression → Linear Regression, Random Forest Regressor, XGBoost
     - Classification → Logistic Regression, Random Forest, SVM, etc.
     - Clustering → KMeans, DBSCAN, Hierarchical
     - Use `sklearn`, `xgboost`, `lightgbm`, or deep learning frameworks if needed (TensorFlow, PyTorch)
   - Model Training
   - Model Evaluation
     - Classification: Accuracy, Precision, Recall, F1, Confusion Matrix, ROC AUC
     - Regression: RMSE, MAE, R^2
     - Use cross-validation if needed (`cross_val_score`, `GridSearchCV`)
   - Model Tuning
     (GridSearchCV, RandomizedSearchCV)
5. Model Deployment (App)
   - Flask, Rest API, Streamlit

Target & predictor

SUPERVISED LEARNING

with Labels

1. LINEAR REGRESSION

predict the continuos number

Linear Relation b/w x and y

Theta Notation

Standard

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

$y = m x + c$

Simple equation

Independent variables

Slope

Coefficient

Dependent variables

θ_0 → Intercept (if $x = 0$) (Base price)

Visual Understanding:

For the equation:

$$y = mx + c$$

- The **intercept (c)** shifts the line **up or down.**
- The **slope (m)** tilts the line:
  - A higher m → steeper incline
  - A lower m → flatter line
  - m = 0 → flat line (no relationship)

Problem Statement

We want to predict the **price of a house** based on its **size in square feet.**

Let's say the trained linear regression model gives us this equation:

$$\text{Price} = 50{,}000 + 300 \times \text{Size}$$

This is in the form:

$$y = mx + c$$

Where:

- y = predicted house price
- x = size in square feet
- $m = 300$ = **slope**
- $c = 50{,}000$ = **intercept**

Example Prediction

If a house is **1,000 sq ft**, then:

$$\text{Price} = 50{,}000 + 300 \times 1000 = 50{,}000 + 300{,}000 = ₹350{,}000$$

Summary Table

Term	Value	Meaning
Intercept	₹50,000	Base price with 0 sq ft (fixed cost, land, fees, etc.)
Slope	₹300	Additional cost per square foot
Input (x)	1000 sq ft	Size of the house
Prediction	₹350,000	Total estimated house price for a 1000 sq ft house

AIM; To find the best fit line, the difference between predicted and actual points should be minimum

Multi Linear Equation

$$H\theta(x) = y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots \theta_n x_n$$

or $\quad y = c + m_1 x_1 + m_2 x_2$

COST FUNCTION; is a mathematical formula that measures how far off a model's prediction are from actual Results in entire Dataset

Measure of INACCURACY

$J\theta$ = Mean Squared Error (MSE) is most commonly used cost function in LR

MSE = The average of squared differences between predicted and actual values

(Cost function) $J\theta = \frac{1}{2m} \sum_{i=1}^{m} (y_i - \hat{y}_i)^2$

Number of Data points; y_i Actual; $\hat{y}_i$ Predictions

Loss vs Cost Function;

Definitions

- **Loss Function**: Measures the error between the predicted output and the actual target value for a **single data point**.
- **Cost Function**: Quantifies the **average loss** across the entire dataset, providing an overall measure of the model's performance.

Key Differences

Aspect	Loss Function	Cost Function
Scope	Single data point	Entire dataset
Purpose	Evaluates individual prediction error	Assesses overall model performance
Usage	Guides model updates per example	Used in optimization to minimize total error
Example	Squared error for one prediction	Mean Squared Error (MSE) over all predictions

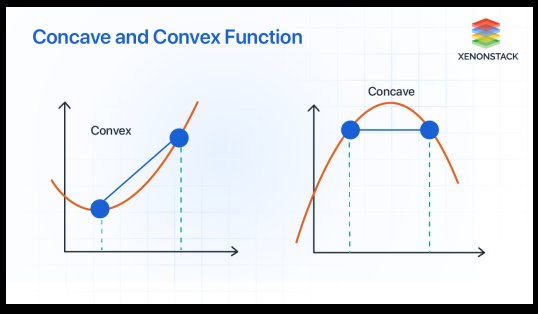
convex for

Gradient Descent; is an optimization Algorithm, to minimize (cost function) or error by iteratively adjusting the parameters like slope and intercept

Aim; of gradient Descent is to minimize the cost function across entire Dataset.

Use; In Linear / Logistic Regression & Neural Nets

Imp; The aim is to reach global minima, but in actual, especially with non convex functions, it may converge to local minimum or saddle point



Gradient Descent Process

I. choose initial values for parameters usually 0 or 1

II. Compute the predictions $y = mx + c$

III. Calculate Partial Derivatives cost function (Gradients) for each parameter $\partial J/\partial m$; $\partial J/\partial c$

These derivatives will tell us how to change m and c to reduce the error ↓ (Directions)

$$\frac{\partial J}{\partial m} = -\frac{2}{n}\sum_{i=1}^{n} x_i \cdot (y_i - \hat{y}_i)$$

$$\frac{\partial J}{\partial c} = -\frac{2}{n}\sum_{i=1}^{n} (y_i - \hat{y}_i)$$

IV. Update Parameters;

$$m = m - \alpha \cdot \frac{\partial J}{\partial m}$$

$$c = c - \alpha \cdot \frac{\partial J}{\partial c}$$

Step Size
learning Rate x slope

New values

where α = learning Rate

why using (−)

Gradient by default shows steepest increase so that's why we use (−) for opp. Direction

📉 If Learning Rate is:

Alpha Value	What Happens
Too Small (e.g., 0.0001)	Very slow learning, takes many steps to reach the minimum
Just Right (e.g., 0.01)	Smooth and stable convergence to the minimum
Too Large (e.g., 1.0)	Overshoots the minimum, may diverge or oscillate

V; Repeat steps until we almost Reach Global Minima

Interview Question;

we can't have a local minima in linear Regression, because LR is a convex function and in such functions there are only Global minima, may be in Neural Nets we will have local minima as well

Types of Gradient Descent;

Stochastic means Random

↳ Large Datasets (use)

🔍 Gradient Descent vs. Stochastic vs. Mini-Batch

Feature	Gradient Descent (GD)	Stochastic Gradient Descent (SGD)	Mini-Batch Gradient Descent
Data per Step	Entire dataset	Single random data point	Small batch (e.g., 32, 64)
Speed	Slow	Fast	Moderate
Convergence Path	Smooth and stable	Noisy and unstable	Balanced and less noisy
Memory Usage	High	Very Low	Moderate
Best For	Small datasets	Very large datasets, online learning	Most real-world ML applications

Q. How to Know if Gradient Descent has Converged (should stop)

"We stop gradient descent when the cost function's change between iterations is very small, the gradients are near zero, or a maximum number of iterations is reached. In practice, we also monitor validation performance to avoid overfitting."

Q. 🌍 Global Minima vs. ☁️ Local Minima

Feature	Global Minima	Local Minima
Definition	Lowest possible point of the cost function across the **entire** curve or surface	A point lower than its neighbors, but **not the absolute lowest**
Cost Value	Smallest of all possible cost values	Small, but not the smallest overall
Uniqueness	Unique (for convex functions)	Can be multiple
Desirability	Ideal solution	May lead to suboptimal results
Occurs in	Convex functions (like Linear Regression)	Non-convex functions (like Neural Networks)

🎯 **Example (Intuition):**

Think of a mountain range:

- **Global Minima** is the **deepest valley**.
- **Local Minima** are **smaller dips** that are lower than nearby peaks but not the lowest overall.

Performance Metrics in Linear Regression

1. R^2 or Goodness of Fit — Coefficient of Determination

basically tells how much of variance in target variable is explained by our model

R^2 — Prediction Accuracy

Let's say actual marks of 5 students are:
[40, 60, 80, 100, 120] — they vary a lot (high variance)

Now your model predicts:
[42, 62, 82, 102, 122] — very close to actual marks!

✅ The model **explained almost all the variance** → R^2 close to 1

But if your model predicted:
[70, 70, 70, 70, 70] — same value for all students

❌ The model explained **none of the variance** → R^2 close to 0

$$R_2 = 1 - \frac{\text{Residual sum of Squares}}{\text{Total sum of squares}} = 1 - \frac{SS_{res}}{SS_{total}}$$

$$R_2 = 1 - \frac{(y - \hat{y})^2}{(y - \bar{y})^2}$$

(Small) (Large) — to our model.

🧠 **Intuition:**

R^2 Value	Interpretation
1	Perfect fit — model explains all the variance
0.9	90% of the variation in Y is explained by the model
0	Model does not explain any of the variation
< 0	Model fits worse than predicting the mean

$\hat{y_i}$ = predictions y_i = Actual

overfitting

2. Adjusted R^2 — Takes Number of features into Consideration

is a modified version of R^2 that penalizes us for adding more features or too many variables

🤔 **Why Adjust R^2?**

- R^2 **always increases** when you add more predictors — even if they're not useful.
- **Adjusted R^2 only increases** if the new predictor **genuinely improves the model**.
- It prevents **overfitting** by adjusting for the number of features.

🔑 **Interpretation:**

Metric	Meaning
$R^2 = 0.90$	Model explains 90% of variability
Adjusted $R^2 = 0.82$	Only 82% is valid after adjusting for added features
If Adjusted R^2 drops	You've added **unnecessary** variables

$$\text{Adjusted } R^2 = 1 - \left(\frac{(1 - R^2)(n - 1)}{n - k - 1}\right)$$

n = number of observations
k = number of predictors

Imp. Why is R^2 always greater than adjusted R^2?

Adjusted R^2 will always be less or equal to R^2 because

$$R^2 = 1 - \frac{RSS}{TSS}$$

where as $\text{Adjusted } R^2 = 1 - (1 - R^2) \cdot \frac{n - 1}{n - p - 1}$

n = number of Data points
p = number of predictors

Lets assume $n = 100$ & $p = 5$, $R^2 = 0.80$

$1-(1-0.8) \cdot 100-1 / 100-5-1$

$1-(0.2) \cdot 99/94$

$1-0.21063 = 0.789362$

This makes sure we are'nt going above R^2 either same or low.

MAE Average absolute difference between predicted and observed Data points.

$$MAE = \frac{1}{n} \sum_{i=1}^{n} (y - \hat{y})$$

Non Sensitive to outliers

- No squaring
- Non Sensitive to outliers

MSE VS MAE

Aspect	MSE	MAE
Formula	Mean of squared errors	Mean of absolute errors
Penalizes big errors?	Yes (more)	No (equally)
Sensitive to outliers?	Yes	No
Use when	You care about big errors	You want robustness

✅ 4. Use Case Preference

Use Case	Preferred Metric
When large errors are **very bad**	**MSE** (e.g. finance, risk modeling)
When you want a **robust** error metric	**MAE** (e.g. real-world noisy data)

When an outlier causes a **large error**, squaring it in MSE makes it **even larger**, so MSE "punishes" big errors **a lot more**.

But MAE just takes the absolute value, so **it doesn't blow up as much**.

🔍 **Example:**

- Error = 10
  - MSE: $10^2 = 100$
  - MAE: $|10| = 10$
- Error = 100 (outlier)
  - MSE: $100^2 = 10,000$ ← 🚨 Huge impact
  - MAE: $|100| = 100$ ← Not as dramatic

UNDERFITTING VS OVERFITTING;

1. Underfitting (High Bias)

- **Model is too simple**
- Can't capture patterns in training data
- Both **training** and **test** error are high

Example: Using a straight line to fit curved data

🧠 **Think:** "Model doesn't learn enough"

Training Error (Simple Data)

2. Overfitting (High Variance)

- **Model is too complex**
- Fits training data *too well*, including noise
- **Training error is low**, but **test error is high**

Example: A zig-zag line that perfectly follows training points but fails on new data

🧠 **Think:** "Model learns too much — even the noise"

BIAS VS VARIANCE

Bias = Error due to Simplicity of model { misses patterns
Wrong assumptions

Training Error

Model performs well on Train Data = High Bias

Variance = Error due to Complexity of Model { Memorizes noise
More Sensitivity to training data

Testing Error

🔁 In short:

Term	Think of it as...	Mistake it makes
Bias	Too dumb / Too simple	Misses real patterns
Variance	Too smart / Too sensitive	Gets fooled by noise

Good performance on Test = Low Variance.

BIAS VARIANCE TRADE OFF

✅ 3. What's the Tradeoff?

- If you **reduce bias**, variance usually **increases**.
- If you **reduce variance**, bias usually **increases**.

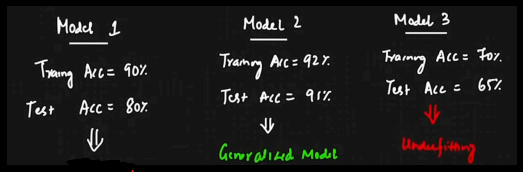
📈 So you need to **balance** them:

Not too simple(*lowbias*), **Not too complex**(*lowvariance*)

🎯 **Final Goal:**

Just complex enough to learn patterns, but simple enough to generalize well!

EXAMPLE;



🔁 Summary Table:

Situation	Bias	Variance	Train Error	Test Error
Underfitting	High	Low	High	High
Overfitting	Low	High	Low	High
Good Fit	Low	Low	Low	Low

RIDGE & LASSO REGRESSION

LASSO REGRESSION ; L_1 Regularization

is a type of Linear Regression that includes regularization term to prevent

- ✓ · Overfitting
- ✓ · Multicollinearity
- ✓ · Feature Selection

Penalizes large coefficients (slope)

$$J\theta = \frac{1}{2m}\sum_{i=1}^{m}(y-\hat{y})^2 + \lambda \cdot |m|$$

λ · mode of m

$\lambda \cdot |m|$ → Regularization Term

Why Use L1 Regularization (Lasso)?

Benefit	Explanation
Feature selection	Unimportant features get zero weight (i.e., are dropped)
Reduces overfitting	Simpler models generalize better to new data
Sparse solutions	Many coefficients = 0 → compact, interpretable models

RIDGE REGRESSION ; L_2 Regularization

is a type of Linear Regression that includes regularization term to prevent

- ✓ · Overfitting
- ✓ · Multicollinearity
- X · Feature Selection (keeps all features)

Shrinks coefficients

$$J\theta = \frac{1}{2m}\sum_{i=1}^{m}(y-\hat{y})^2 + \lambda \cdot (m)^2 \longrightarrow \lambda \cdot (\text{slope})^2$$

REGULARIZATION

technique to avoid overfitting by penalizing large co-efficients.

λ = Regularization Parameter (Hyper parameter)

↳ can be found through (Cross Validation tech)

- If $\lambda = 0$ ⟶ Standard Linear Regression
- If $\lambda = \infty$ ⟶ shrinks towards 0
- If λ = too high ⟶ Underfitting

Lasso vs Ridge Regression

L_1 Lasso $|m|$

Shrinks close to zero (Removes)

(feature selection)

L_2 Ridge m^2

Shrinks all (but keeps) coefficients

(No feature selection)

Realtime use both and choose whichever performs well.

Assumptions of Linear Regression LINNM

1. **Linearity:**
   The relationship between the input features and the output (target) is a straight line. This means that if the features change, the target changes proportionally.
2. **Independence:**
   Each data point is independent of others. One observation should not influence or be related to another. This is important to avoid biased results.
3. **Homoscedasticity:** Spread of errors is consistent
   The errors (differences between actual and predicted values) have the same amount of spread or variance across all levels of the features. So, the model's mistakes are consistent throughout.
4. **Normality of errors:**
   The errors should be roughly normally distributed (like a bell curve). This helps in making reliable confidence intervals and hypothesis tests.
5. **No multicollinearity:**
   The input features should not be highly correlated with each other. When features are very similar, it's hard to tell which one is actually affecting the target, and it can confuse the model."

Linear Regression Practicals

Train Test split

- train_size = 0.8 means 80% of the data will be used for training
- random_state = sets the seed for reproducibility; commonly used value is 42

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,train_size=0.8,random_state=42)
```

→ Simple Linear Regression

```
from sklearn.linear_model import LinearRegression
linear_reg=LinearRegression()
linear_reg.fit(X_train, y_train)

#To get predictions
y_pred=linear_reg.predict(X_test)
```

Model Evaluation

```
# Model Evaluation   from sklearn.metrics import *
linear_reg_mse = mean_squared_error(y_test, y_pred)
linear_reg_rmse = mean_absolute_error(y_test, y_pred)
linear_reg_r2_score = r2_score(y_test, y_pred)
```

→ With Cross Val Score

```
from sklearn.model_selection import cross_val_score
# Use neg_mean_squared_error for MSE scoring (must take negative to get actual MSE)
mse_scores = cross_val_score(linear_reg, X_train, y_train, scoring='neg_mean_squared_error', cv=5)
```

→ Ridge Regression L2 Regularization

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV

# Initialize the Ridge regressor
ridge_reg = Ridge()

# Define the set of alpha values to search
params = {'alpha': [0.0001, 0.001, 0.01, 0.1, 1, 5, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]}

# Use GridSearchCV to find the best alpha
ridge_regressor = GridSearchCV(ridge_reg, params, scoring='neg_mean_squared_error', cv=5)
ridge_regressor.fit(X_train, y_train)

# Output the best alpha and corresponding (positive) MSE
print("Best alpha value:", ridge_regressor.best_params_['alpha'])
print("Best MSE:", -ridge_regressor.best_score_)
```

→ value of λ

Feature Reduction.

→ Lasso Regression L1 Regularization

```
from sklearn.linear_model import Lasso
from sklearn.model_selection import GridSearchCV

# Initialize the Ridge regressor
lasso_reg = Lasso()

# Define the set of alpha values to search
params = {'alpha': [0.0001, 0.001, 0.01, 0.1, 1, 5, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]}

# Use GridSearchCV to find the best alpha
lasso_regressor = GridSearchCV(lasso_reg, params, scoring='neg_mean_squared_error', cv=5)
lasso_regressor.fit(X_train, y_train)

# Output the best alpha and corresponding (positive) MSE
print("Best alpha value:", lasso_regressor.best_params_['alpha'])
print("Best MSE:", -lasso_regressor.best_score_)
```

Feature Reduction.

Note; Finally Model with less MSE will be chosen for predictions.

2. Logistic Regression classification

is used to predict category or class

- when output var is category / class

eg., pass / fail

Imp; Logistic Regression is best suited for Binary classification problems.

Scenarios If $h_\theta(x) < 0.5 = 0$ Fail

$h_\theta(x) \geq 0.5 = 1$ pass

Why can't I use linear Regression in Logistic problem

We can't because Linear isn't built to handle probabilities like [0,1] it can go beyond that like 0.2, 0.3, 0.5 (continuous prediction)

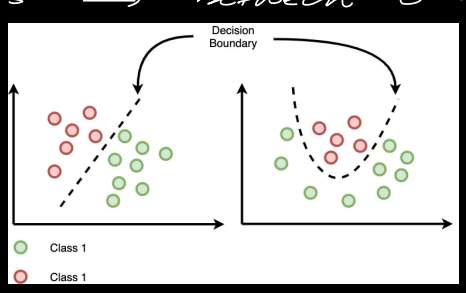
=> Logistic Regression uses Sigmoid function to handle classification problems

Imp Binary classification = Sigmoid function

Multi class classification = Softmax function

DECISION BOUNDARY It is a separing line or separator b/w two classes

e.g., 0.5 ⟶ between 0 and 1



SIGMOID FUNCTION; is used to convert any real number into 0 & 1 (interpreted as probability)

Logit function

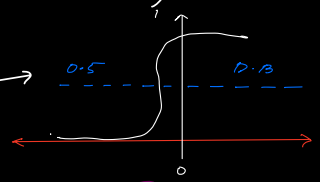
$$f(z) \text{ or } g(z) = \sigma(z) = \frac{1}{1+e^{-z}}$$

$z = mx + c$

- z is any real number (often a linear combination of features and weights).
- e is the base of natural logarithms (approx. 2.718).

Q. Why is Sigmoid function called Activation function because it converts raw output into the probability (0,1) enabling LR to make binary predictions

SIGMOID FUNCTION



DECISION RULE IN LOGISTIC REGRESSION

✅ **Case 1: If $z > 0$**

- Exponential term becomes small: $e^{-z} < 1$
- So:

$$g(z) = \frac{1}{1 + \text{small number}} > 0.5$$

- ➤ **Meaning:**
  The predicted probability of Class 1 is **more than 50%**
- ➤ So we **predict Class 1**

✅ **Case 2: If $z < 0$**

- Exponential term becomes large: $e^{-z} > 1$
- So:

$$g(z) = \frac{1}{1 + \text{large number}} < 0.5$$

- ➤ **Meaning:**
  The predicted probability of Class 1 is **less than 50%**
- ➤ So we **predict Class 0**

✅ **Case 3: If $z = 0$**

- Then:

$$g(0) = \frac{1}{1 + e^0} = \frac{1}{2} = 0.5$$

- ➤ This is the **exact threshold**
- ➤ It's called the **decision boundary**

🎯 **One-liner for interviews:**

> "If z > 0, probability > 0.5 → Class 1; if z < 0, probability < 0.5 → Class 0; z = 0 gives 0.5 → decision boundary."

Imp. Logit function = Sigmoid = Activation fx.

COST FUNCTION IN LOGISTIC REGRESSION

Q. can we use MSE as cost function in Logistic

> "MSE doesn't work for logistic regression because it creates a **non-convex cost function** due to the sigmoid transformation. This makes it hard for gradient descent to converge to the global minimum. Instead, we use **log loss**, which results in a **convex function**—ensuring stable and efficient optimization for classification problems."

🔁 **Optional Follow-up (if they ask "why non-convex is bad?"):**

> "Non-convex functions can have multiple local minima or flat regions, which can cause gradient descent to get stuck and fail to find the best solution."

V.V.Imp Why log loss over MSE ? Local Minima Problem

We can't use MSE on Logistic Regression because of Sigmoid Transformation, if MSE is used anyway, it will make it Non convex function, resulting in multiple local minima;

Instead, we use log loss as cost function, which results in convex function.

Logistic Cost Function

Log Loss

$$J(\theta) = \frac{1}{n}\left[\left(y_i \cdot \log(\hat{y}_i) + 1 - y_i\left(\log(1-\hat{y}_i)\right)\right)\right]$$

$$y = \frac{1}{1+e^{-(mx+c)}}$$

or

$$H_\theta(x) = \frac{1}{1+e^{-(\theta_1 x+\theta_0)}}$$

$J(\theta)$ ↓ Log Loss

Imp.

🔁 **If you compare to simple linear regression:**

Linear Regression	Logistic Regression	Meaning
$y = mx + c$	$\hat{y} = \frac{1}{1+e^{-(mx+c)}}$	m is **slope**, c is **intercept**
$\theta_1 = m$	$\theta_1 = m$	weight (slope)
$\theta_0 = c$	$\theta_0 = c$	intercept (bias)

Gradient Descent in Logistic Regression

1. Initialize the weights (m)
   θ_j , usually 0
2. Compute predictions (Sigmoid function)
3. Compute Gradients wrt weights θ_j

$$\frac{\partial J(\theta)}{\partial j} = \frac{1}{n}\sum_{i=1}^{n}\left(\hat{y}^i - y^i\right) \cdot x_j^{(i)}$$

4. Update Parameters using Gradient Descent

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

where α = Learning Rate

5. Repeat until Convergence

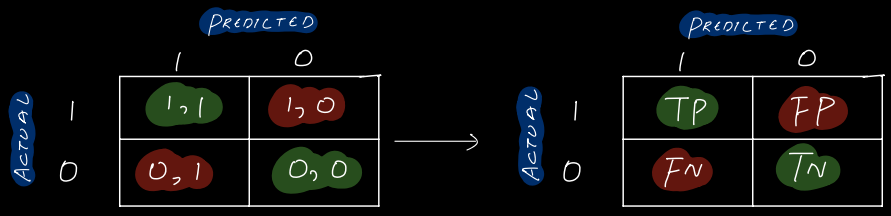
Convergence ⟶ cost doesn't change much iterations are over.

⚠️ **Important Notes:**

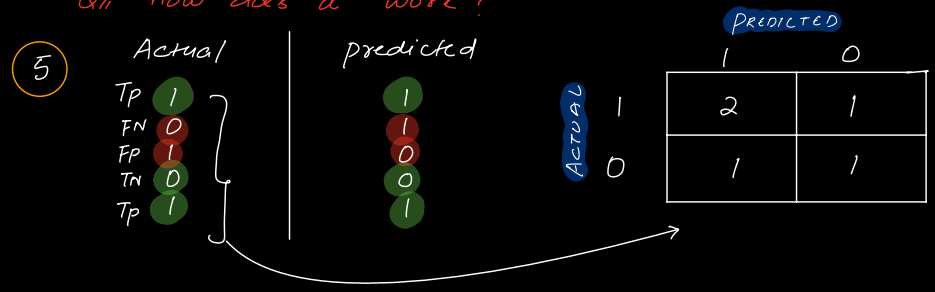
- In logistic regression, we **don't use squared error**. That's why the gradients are different.
- The process is **very similar** to linear regression, but we:
  - Use **sigmoid** for predictions
  - Use **log loss** for the cost

PERFORMANCE METRICS FOR CLASSIFICATION

CONFUSION MATRIX V.V.Imp



Q11 How does it work ?



ACCURACY = Ratio of True predictions / Total predictions

$$\frac{Tp + TN}{Tp + FP + FN + TN} \longrightarrow \frac{2+1}{2+1+1+1} = \frac{3}{5} = 0.6 \cong 60\%$$ Accuracy

PRECISION = Accuracy of positive predictions

$$\frac{Tp}{Tp + FP} = \frac{\text{True positive predictions}}{\text{Total positive predictions}}$$

Sensitivity

RECALL = True +ve Rate Out of all actual positives, how many did model predict accurately

FN is actually positive predicted Negative

$$\frac{Tp}{Tp + FN} = \frac{\text{True positive predictions}}{\text{Total Actual positives}}$$

Imp. FN is Type II error is considered to be more critical

USE CASE = Spam classify (precision) ; Cancer Detection (Recall)

Imp.

* F Score Let's suppose, we are predicting stock market crash, then I want both (PR)

F Score ; is the harmonic mean of precision and recall

→ single score that balances both

$$F_\beta = (1+\beta^2) \times \frac{\text{precision} \times \text{Recall}}{\text{precision} + \text{Recall}}$$

Q) Is it F Score or F_1 Score ?

NO, it actually depends on value of β

$\beta = 1 \longrightarrow f_1$ Score → Balanced

$$F_1 \text{ Score} = 2 \times \frac{P \times R}{P + R}$$

$\beta = 0.5 \longrightarrow f_{0.5}$ score → FP

$F_{0.5}$ Score

$\beta = 2 \longrightarrow f_2$ score → FN

F_2 Score

Ridge Logistic Regularization

In **Logistic Regression**, **Ridge regularization** refers to **L2 regularization**, which penalizes large coefficients to prevent overfitting.

📌 **Regularized Logistic Loss Function (with L2):**

$$\mathcal{L}_{\text{ridge}} = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)] + \lambda \sum_{j=1}^{p} \theta_j^2$$

Linear → $\lambda = \frac{1}{C}$ → Logistic

🔍 **How it works:**

- **Large** `C` **(e.g., 1000)** → small regularization → model fits more flexibly.
- **Small** `C` **(e.g., 0.01)** → strong regularization → keeps weights small to avoid overfitting.

Q. What is Cross validation?

Cross Validation is a technique to evaluate model's performance by splitting the data into multiple parts (K-1) folds, Let's say K = 5

```python
from sklearn.model_selection import cross_val_score
# Use neg_mean_squared_error for MSE scoring (must take negative to get actual MSE)
mse_scores = cross_val_score(linear_reg, X_train, y_train, scoring='neg_mean_squared_error', cv=5)
```

Train on 4 and Test on 1, and then changing folds for Train & Test

Use
- Reduces Overfitting
- Helps choose the best model or parameters

Q. What are Types of CV.

1. **GridSearch CV** extensively searches through all combinations of the hyperparameters specified.

```python
# Define the hyperparameter grid to search over
param_grid = {
    'lgr__max_iter': [100, 200, 300, 500],
    'lgr__C': [0.01, 0.1, 1, 10, 100]
}
```

Total Combinations
max iter = 4
C = 5
} 4 x 5 = 20

Imp: If you pass this to `RandomizedSearchCV` with `n_iter=5`, it would sample **5 random combinations** from these 20.

↳ specific to RandomCV

2. **Randomized Search CV** randomly samples from the parameter space for fixed no. of iterations

```python
random_search = RandomizedSearchCV(
    estimator=pipe,
    param_distributions=param_grid,
    n_iter=5,   # 👈 You choose how many combinations to try
    cv=5,
    random_state=42
)
```

n_iters is important

🔍 Side-by-Side Summary:

Feature	GridSearchCV	RandomizedSearchCV
Search Method	Exhaustive	Random Sampling
Speed	Slower	Faster
All Combos Tried	✅ Yes	❌ No (limited by `n_iter`)
Best for	Small param space	Large param space
Guarantees best?	✅ Yes	❌ Not guaranteed

Logistic Regression Practicals

- Categorical Data should be Encoded
- Scaling should be done only after train-test split, to avoid data leak (b/w Test & Train)
  - Scale Train Data } Same Scaler
  - Scale Test Data }

```
# 2. Fit scaler on training data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)

# 3. Transform test data using the same scaler
X_test_scaled = scaler.transform(X_test)
```

-

```
# Create a pipeline that standardizes the data, applies PCA, then fits a Logistic Regression model
lgr_pipe = Pipeline(steps=[
    ('scaler', StandardScaler()),        # Step 1: Scale features (mean=0, std=1)
    ('pca', PCA()),                      # Step 2: Apply Principal Component Analysis for dimensionality reduction
    ('lgr', LogisticRegression())        # Step 3: Train a Logistic Regression model
])

# Define the hyperparameter grid to search over
param_grid = {
    'pca__n_components': np.arange(1, X_train.shape[1] // 3),  # Try different number of PCA components
    'lgr__max_iter': [100, 200, 300, 500],                     # Try different max_iter values
    'lgr__C': [0.01, 0.1, 1, 10, 100]                          # Try different reverse regularization strengths (lower = more regularization)
}

# Use GridSearchCV to find the best combination of hyperparameters
lgr_model = GridSearchCV(
    lgr_pipe,              # The pipeline to evaluate
    param_grid=param_grid,  # The parameter grid to search
    scoring='f1',
    cv = 5     # Use accuracy as the scoring metric
)

# Fit the model on training data
lgr_model.fit(X_train, y_train)

# Print the best hyperparameters found by GridSearchCV
print('Best params: {}'.format(lgr_model.best_params_))

# Print the best cross-validation score from training
print('Best Score: {}'.format(lgr_model.best_score_))
```

Ridge by Default.

$C = 1/\lambda$ (Inverse of Regularization)

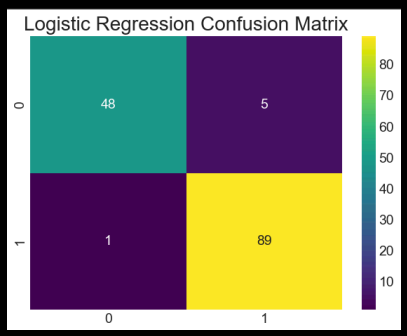
Note; when using GridSearch CV it automatically selects Best params for Model

```
from sklearn.metrics import classification_report, confusion_matrix
y_pred = lgr_model.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, cmap = 'viridis')
plt.title('Logistic Regression Confusion Matrix')
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.98      0.91      0.94        53
           1       0.95      0.99      0.97        90

    accuracy                           0.96       143
   macro avg       0.96      0.95      0.95       143
weighted avg       0.96      0.96      0.96       143
```



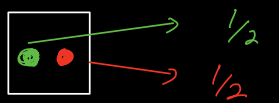
Note; we can check for metrics individually.

classification **NAIVE BAYES** supervised

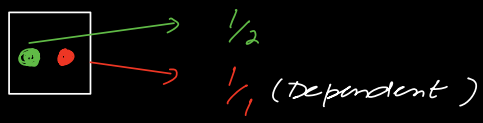
is a classification ML algo based on Bayes Theorem

- Calculates probability of each class (Y/N)
- Naive (Assumes feature Independence)

Independent Event



Dependent Event

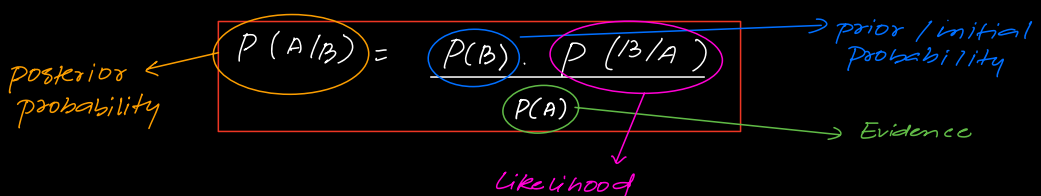


Conditional Probability

$$P(A \text{ and } B) = P(A) \cdot P(B/A)$$ ← Given

Also, $P(A \text{ and } B) = P(B \text{ and } A)$

$$P(A) \cdot P(B/A) = P(B) \cdot P(A/B)$$



- **P(A|B)**: Probability of **A given B** (posterior)
- **P(B|A)**: Probability of **B given A** (likelihood)
- **P(A)**: Probability of **A** (prior)
- **P(B)**: Probability of **B** (evidence)

BAYES THEOREM; It updates conditional probability of probability of hypothesis based on new evidence

Use Case Spam Detection, Sentiment Analysis

Types of Naive Bayes:

- **Gaussian Naive Bayes** – For continuous features (assumes normal distribution)
- **Multinomial Naive Bayes** – For count features (e.g. word frequencies)
- **Bernoulli Naive Bayes** – For binary features (e.g. 0/1 presence of words)

Pros:

- Very fast and simple
- Works well with high-dimensional data
- Requires less training data

Cons:

- Assumes feature independence
- Not ideal when features are highly correlated

Example Let's assume

A = y ⇒ Target

B = x ⇒ $x_1, x_2, x_3 \dots x_n$ Independent

$$P(y/x_1,x_2,x_3\dots x_n) = \frac{P(y) * P(x_1,x_2\dots x_n/y)}{P(x_1,x_2\dots x_n)}$$ Yes or No

$$= \frac{P(y) * P(x_1/y_1) * P(x_2/y_2) * P(x_3/y_3) \dots P(x_n/y_n)}{P(x_1) * P(x_2) * P(x_3) * \dots P(x_n)}$$

DATASET

x_1	x_2	x_3	x_4	y
→				Yes ✓
				No ✓

Yes

$$P(y=Yes/x_i) = \frac{P(Yes) * P(x_1/Yes) * P(x_2/Yes) * P(x_3/Yes) * P(x_4/Yes)}{P(x_1) * P(x_2) * P(x_3) * P(x_4)}$$

Constant Ignore ⟶ $P(x_1) * P(x_2) * P(x_3) * P(x_4)$ # fixed

NO

$$P(y=No/x_i) = \frac{P(No) * P(x_1/No) * P(x_2/No) * P(x_3/No) * P(x_4/No)}{P(x_1) * P(x_2) * P(x_3) * P(x_4)}$$

Constant ⟶ $P(x_1) * P(x_2) * P(x_3) * P(x_4)$ # fixed

Imp Let's Assume

$$P(Yes/x_i) = 0.13 \qquad P(No/x_i) = 0.05$$

Then we will Normalize

$$P(Yes/x_i) = \frac{0.13}{0.13 + 0.05} = 0.72$$

72%.

$$P(No/x_i) = 1 - 0.72 = 28\%$$

Q) Lets solve problem wrt Naive Baye's

Outlook	Cricket
RAIN	NO
CLEAR	YES
RAIN	YES
CLEAR	YES
CLEAR	YES
CLEAR	NO

	Yes	No	P(O/y)	P(O/N)
RAIN	1	1	1/4	1/2
CLEAR	3	1	3/4	1/2
	4	2		

$$P\left(\frac{yes}{rain,clear}\right) = \frac{P(yes) * P\left(\frac{rain}{yes}\right) * P\left(\frac{clear}{rain}\right)}{P(rain) * P(clear)}$$

P(clear) — Constant

$$0.083 = 1/3 * 1/4 * 3/4 = 0.0625$$

$$P\left(\frac{NO}{rain, clear}\right) = 1/3 * 1/4 * 1/4 = 0.02$$

Normalize

$$P\left(\frac{yes}{rain, clear}\right) = \frac{0.0625}{0.0625 + 0.02}$$

$$= \frac{0.0625}{0.2625} = 0.238 = 23.8\%$$

$$P\left(\frac{no}{rain, clear}\right) = 1 - 0.238 = 0.762$$

$$= 76.2\%$$

Conclusion; we can say, chances of

No Cricket = 76.2%.

Yes Cricket = 23.8%.

So, NO cricket is final prediction

Naive Bayes Practicals

```
gnb_pipe = Pipeline(steps=[
    ('scaler', StandardScaler()),
    ('pca', PCA()),
    ('gnb', GaussianNB())
])

param_grid = {
    'pca__n_components': np.arange(1, X_train.shape[1]+1)
}

gnb_model = GridSearchCV(gnb_pipe, param_grid=param_grid, verbose=1, n_jobs=-1)
gnb_model.fit(X_train, y_train)
print('Best params: {}'.format(gnb_model.best_params_))
print('Training Score: {}'.format(gnb_model.score(X_train, y_train)))
print('CV Score: {}'.format(gnb_model.best_score_))
print('Test Score: {}'.format(gnb_model.score(X_test, y_test)));
```

✓ 0.0s

```
Fitting 5 folds for each of 30 candidates, totalling 150 fits
Best params: {'pca__n_components': np.int64(7)}
Training Score: 0.9295774647887324
CV Score: 0.9251436388508892
Test Score: 0.916083916083916
```

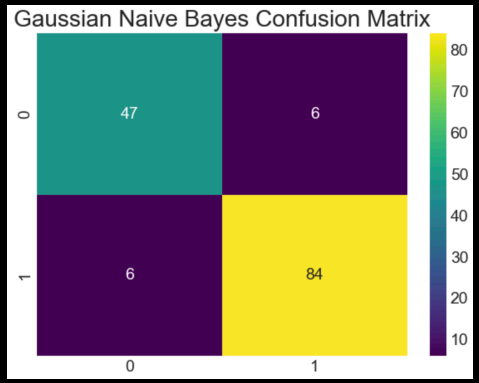
```
from sklearn.metrics import classification_report, confusion_matrix
y_pred = gnb_model.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, cmap = 'viridis')
plt.title('Gaussian Naive Bayes Confusion Matrix')
print(classification_report(y_test, y_pred))
```

✓ 0.0s

```
              precision    recall  f1-score   support

           0       0.89      0.89      0.89        53
           1       0.93      0.93      0.93        90

    accuracy                           0.92       143
   macro avg       0.91      0.91      0.91       143
weighted avg       0.92      0.92      0.92       143
```



KNN

K Nearest Neighbours is a supervised algorithm used for both classification & Regression

🧠 How It Works (Intuition)

When you give a new data point to the model:

1. It **calculates the distance** from this point to all training data points (usually using **Euclidean distance**).
2. It selects the **K closest points** (neighbors).
3. For **classification**, it assigns the class most common among the K neighbors.
4. For **regression**, it returns the **average** (or weighted average) of the K neighbors' values.

📌 Example (Classification):

Suppose you want to predict whether a fruit is an **apple** or **orange** based on its **weight** and **color**.

- You input a new fruit's weight and color.
- KNN finds the **K nearest known fruits**.
- If most of those neighbors are **apples**, it classifies the new one as an **apple**.

📌 Example (KNN Regression with One Feature):

Suppose you want to **predict the price of a house** based **only on its size (in square feet)**.

You input a new house size:
🟦 **1200 sq ft**

The KNN algorithm will:

1. Look for the **K houses** in the dataset whose sizes are **closest** to 1200 sq ft (using distance — like Manhattan or Euclidean).
2. Take the **average price** of those K nearest houses.
3. Predict that average as the **price for the 1200 sq ft house**.

Error Rate

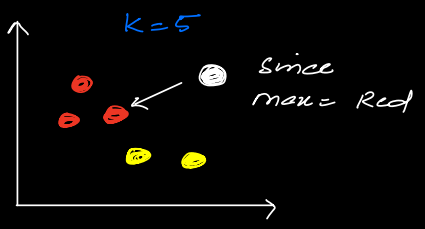
🔷 Example:

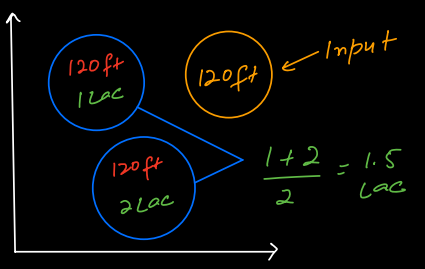
You tested KNN with **K = 3** on 20 samples and:

- It predicted 16 correctly
- 4 were wrong

Then:

$$\text{Error Rate} = \frac{4}{20} = 0.20 \text{ or } 20\%$$





🔁 In summary:

- **KNN for classification** → Majority class among neighbors
- **KNN for regression** → Average (or weighted average) of neighbor values

Q. Which distances are used ?

1. Euclidean Distance

Shortest straight line Distance between the two points

$$\sqrt{(x_1 - x_2) + (y_1 - y_2)^2}$$

2. Manhattan Distance

Stairs like (horizontal and vertical) distance b/w 2 points

$$|x_1 - x_2| + |y_1 - y_2|$$

KNN PRACTICALS

KNN CLASSIFIER

```
knn_pipe = Pipeline(steps=[
    ('scaler', StandardScaler()),
    ('pca', PCA()),
    ('knn', KNeighborsClassifier())
])

param_grid = {
    'pca__n_components': np.arange(1, X_train.shape[1]+1),
    'knn__n_neighbors': np.arange(1, X_train.shape[1], 2)
}

knn_model = GridSearchCV(knn_pipe, param_grid=param_grid, verbose=1, n_jobs=-1)
knn_model.fit(X_train, y_train)

print('Best params: {}'.format(knn_model.best_params_))
print('Training Score: {}'.format(knn_model.score(X_train, y_train)))
print('CV Score: {}'.format(knn_model.best_score_))
print('Test Score: {}'.format(knn_model.score(X_test, y_test)));
```

✓ 1.0s

```
Fitting 5 folds for each of 450 candidates, totalling 2250 fits
Best params: {'knn__n_neighbors': np.int64(11), 'pca__n_components': np.int64(9)}
Training Score: 0.9788732394366197
CV Score: 0.974281805745554
Test Score: 0.951048951048951
```

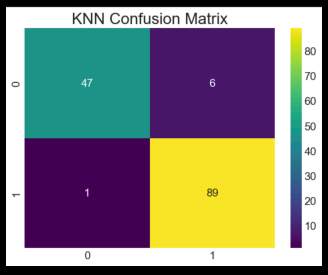
```
from sklearn.metrics import classification_report, confusion_matrix
y_pred = knn_model.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, cmap = 'viridis')
plt.title('KNN Confusion Matrix')
print(classification_report(y_test, y_pred))
```

✓ 0.0s

```
              precision    recall  f1-score   support

           0       0.98      0.89      0.93        53
           1       0.94      0.99      0.96        90

    accuracy                           0.95       143
   macro avg       0.96      0.94      0.95       143
weighted avg       0.95      0.95      0.95       143
```



KNN REGRESSOR

```
knn = KNeighborsRegressor(n_neighbors=10)
knn.fit(X_train, y_train)

#To get predictions
y_pred4 = knn.predict(X_test)
```

✓ 0.0s

```
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# Evaluation Metrics
knn_mse = mean_squared_error(y_test, y_pred4)
knn_rmse = np.sqrt(knn_mse)
knn_r2_score = r2_score(y_test, y_pred4)

print(f"Mean Squared Error (MSE) using KNN Regressor: {knn_mse:.4f}")
print(f"Root Mean Squared Error (RMSE) using KNN Regressor: {knn_rmse:.4f}")
print(f"R² Score using KNN Regressor: {knn_r2_score:.4f}")
```

✓ 0.0s

```
Mean Squared Error (MSE) using KNN Regressor: 3190105.5410
Root Mean Squared Error (RMSE) using KNN Regressor: 1786.0867
R² Score using KNN Regressor: 0.8416
```

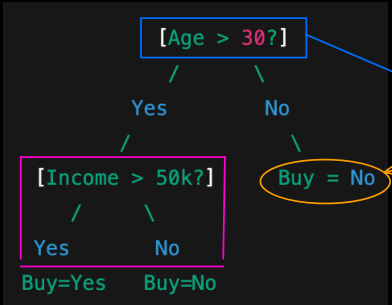
Q1 Difference b/w Euclidean & Manhattan

Feature	Manhattan Distance	Euclidean Distance
Path shape	Grid/stepwise	Straight line
Sensitivity to outliers	Less sensitive	More sensitive
Use case	Sparse/high-dimensional data	Low-dimensional, continuous data
Computation cost	Cheaper (no squares/roots)	Slightly costlier (uses square root)

Decision Tree

is a supervised algorithm used for both classification and Regression.

Imp - It models decisions in the form of a Tree



Key Concepts:

- **Root Node:** The first decision point.
- **Leaf Nodes:** Final prediction outcomes.
- **Splitting:** Choosing a feature and threshold to split the data.
- **Gini Impurity / Entropy**: Used to decide the best splits in classification.
- **Mean Squared Error (MSE)**: Used for regression trees.

Advantages:

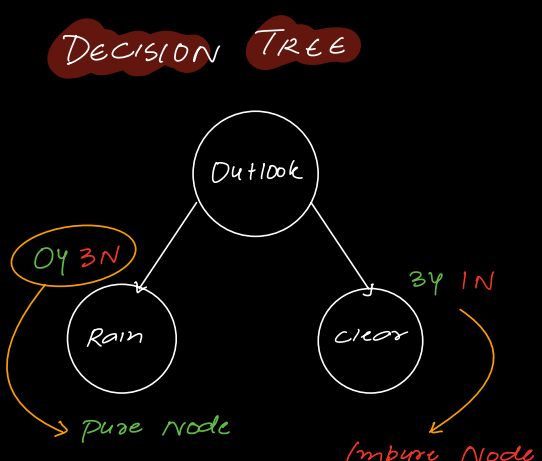
- Easy to understand and interpret.
- Requires little data preparation.
- Handles both numerical and categorical data.

Disadvantages:

- Prone to **overfitting**.
- Can be unstable (small changes in data → different tree).
- Not as accurate as ensemble methods like Random Forests.

Example

Outlook	Cricket
RAIN	NO
CLEAR	Yes
RAIN	NO
CLEAR	Yes
CLEAR	YES
CLEAR	NO



- **PURE Node**; If all the Data points belong to the same class

  e.g., Rain = OVERCAST → Definitely NO cricket

  (NO uncertainity)

- **IMPURE Node**; If data points belong to mix of different classes

  (Uncertainity exists)

Imp. In Decision Tree, our main aim is to split data in a way that increases purity

Q// How to measure Impurity ?

1. Entropy 2. Gini Index / Impurity

Q// How features are selected

Information Gain

Purity Test

ENTROPY; measure of impurity in a Node (0-1) Range

— uses log of probabilities

1. Low Entropy → low impurity (mostly one class)
2. High Entropy → high impurity (multiple classes)

$$\text{Formula} = -P_+ \log_2 P_+ - P_- \log_2 P_-$$

P_+ = Yes

P_- = NO

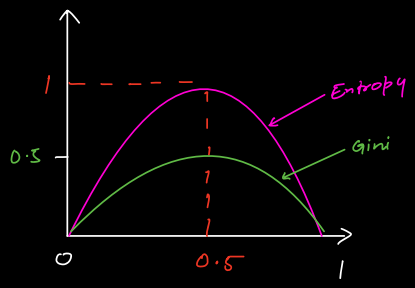
GINI IMPURITY to measure impurity of Nodes

— uses squared probabilities (0-0.5) Range

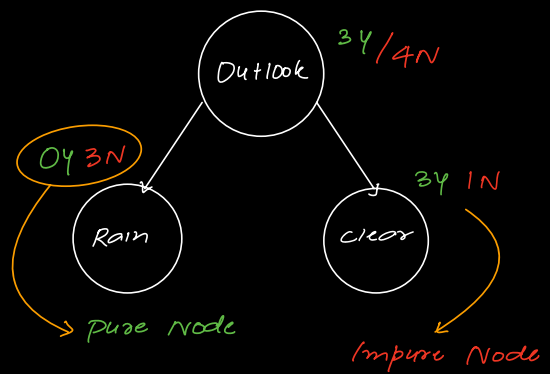
$$\text{Formula} = 1 - \sum_{\pm} (P)^2$$

Entropy vs Gini

Feature	Entropy	Gini Index
Math	Uses logarithm	Uses squared probabilities
Range (Binary)	0 to 1	0 to 0.5
Speed	Slightly slower (log calc)	Faster (no log)
Preference	More information-theoretic	More efficient in practice



Example for Entropy and Gini Impurity



	P(Y)	P(N)
Rain	0/3	3/4
Clear	3/3	1/4

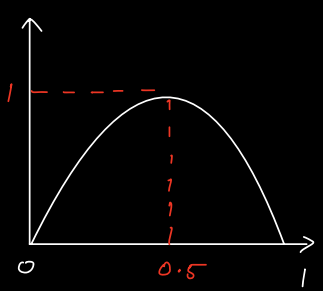
Let's Consider **Clear**

Entropy $H(S) = \frac{3}{3}\log_2\frac{3}{3} - \frac{1}{4}\log_2\frac{1}{4}$

$= 1\log_2 1 - 0.25\log_2 0.25$

$= 0 - (-0.5)$

$= 0.5 \longrightarrow$ Impure Split



Imp: Entropy Ranges 0-1

(0.5, 0.5) = mean high impurity (Max Entropy)

(1,0) or (0,1) = mean less impurity

Information Gain helps us to choose best feature to split on at each step

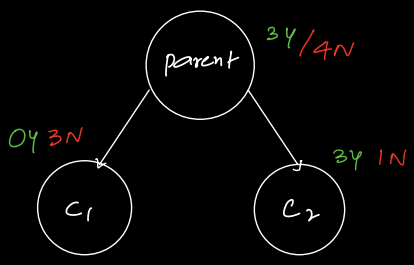
Formula:

$$\text{Information Gain} = \text{Entropy (Parent)} - \sum\left(\frac{\text{Samples in Child}}{\text{Samples in Parent}} \times \text{Entropy (Child)}\right)$$

Steps:

1. Calculate **Entropy** of the parent node (before split).
2. Calculate the **Entropy** for each child node (after split).
3. Take the **weighted average** of child entropies.
4. Subtract this from the parent entropy → that's your **Information Gain**.

→ Let's Calculate Information Value



1. **Entropy** (parent)

$= \frac{3}{7}\log_2\frac{3}{7} - \frac{4}{7}\log_2\frac{4}{7}$

≈ 0.98

2. **Entropy** (child C_1)

$= \frac{0}{3}\log_2\frac{0}{3} - \frac{3}{3}\log_2\frac{3}{3}$

$= 0$

3. Entropy (child C_2)

$= \frac{3}{4}\log_2\frac{3}{4} - \frac{1}{4}\log_2\frac{1}{4} \approx 0.811$

4. Samples = Total Samples = 7

C_1 samples = 3

C_2 samples = 4

Information Gain $= 0.90 - \left[\frac{3}{7} \times 0 + \frac{4}{7} \times 0.811\right]$

$\rightarrow$ 0.5164

Imp. :- If gain of feature 2 = 0.490

then I would prefer feature 1 because

$Gain(f_1) > Gain(f_2)$

NOTE; The feature with max information gain is given preffernce.

Q// when to use Entropy & when to use Gini?

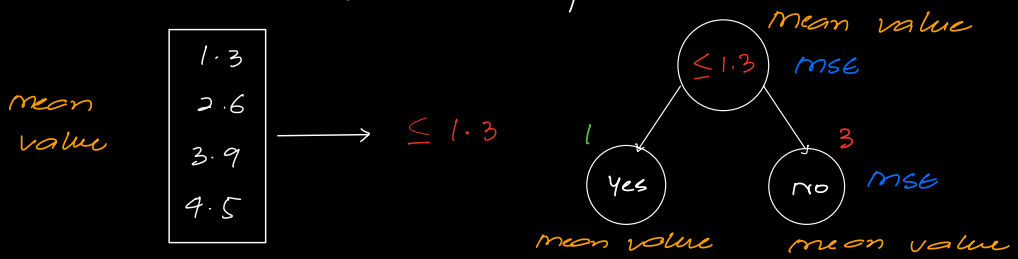
Use Entropy for simple data, where number of features is less.

- Entropy uses Log which uses a lot of computational power

Preffered; Go for Gini, when Data is complex as it will save alot of time

DECISION TREE REGRESSOR.

used to predict continuous variable by learning Decision Rules. It uses MSE or MAE as measure of impurity



- For Impurity we use MSE ;
- For Information Gain $MSE_{parent} - MSE_{split}$
  ↳ which is best feature to split on

V. Imp

📌 What Information Gain really does:

It **quantifies** how good a particular feature is for **reducing impurity**.

Let's say you're considering 3 features:

- Feature A
- Feature B
- Feature C

Each of them can split the node in different ways.

What Information Gain tells you is:

> *"If I split using Feature A, the impurity drops by this much. If I split using Feature B, the drop is less. So I should pick Feature A — it gives the highest 'gain' in purity."*

✅ Decision Tree Classifier vs Regressor (with Formulas)

Feature	Classifier	Regressor
Impurity Metric	**Gini Index:** $Gini = 1 - \sum_{i=1}^{C} p_i^2$ **Entropy:** $H = -\sum_{i=1}^{C} p_i \log_2 p_i$	**Mean Squared Error (MSE):** $MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2$
Purity Definition	Pure if all samples belong to **one class** (i.e., Gini = 0 or Entropy = 0)	Pure if all target values are **very similar** (i.e., MSE ≈ 0)
Split Evaluation Metric	**Information Gain:** $IG = H(\text{parent}) - \sum_{k=1}^{K} \frac{n_k}{n} H(\text{child}_k)$	**Reduction in MSE:** $\Delta MSE = MSE(\text{parent}) - \sum_{k=1}^{K} \frac{n_k}{n} MSE(\text{child}_k)$
Prediction at Leaf	**Most common class** in the leaf (mode)	**Mean** of target values in the leaf $\hat{y}_{\text{leaf}} = \frac{1}{n}\sum_{i=1}^{n} y_i$

Imp. Decision Tree is more sensitive to outliers
we can use pruning or go for Random Forest

Hyperparameters

1. Pruning → cutting back tree to avoid Overfitting → avoid Noise

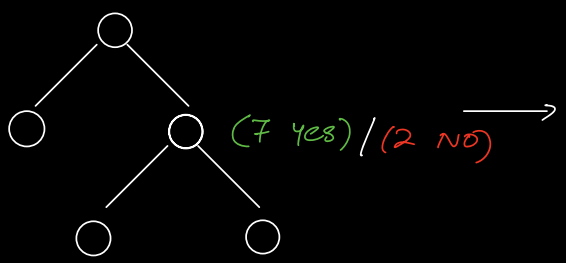
More used

Prepruning
Early stopping → max_depth, max_leaf no. of samples

Post Pruning
Grow full tree then cut branches
ccp_alpha parameter

What is need of pruning ?

So let say



Here it is obvious it is yes, so we should prun here to avoid capturing Noise

DECISION TREE PRACTICAL

1. Use plot_Tree to visualize
2. Mostly focus on Max_Depth (Hyper parameter)

→ Max level = 4 = Max Depth 4

min_samples at leaf = 4

min_samples at split = 4

```python
# Define the grid with important pre-pruning hyperparameters
param_grid = {
    # max_depth controls how deep the tree can grow.
    # Lower depth means simpler trees, helps prevent overfitting.
    'DTC__max_depth': [2, 3, 4, 5, 6, 7, 8, 9, 10],

    # min_samples_split defines the minimum number of samples required to split an internal node.
    # Higher values prevent the tree from growing too deep with small splits.
    'DTC__min_samples_split': [2, 5, 10],

    # min_samples_leaf sets the minimum number of samples required to be at a leaf node.
    # Higher values smooth the model and reduce overfitting by preventing small leaf sizes.
    'DTC__min_samples_leaf': [1, 2, 4]
}

# Fit model with cross-validation
DTC_model = GridSearchCV(DTC_pipe, param_grid, cv=5)
DTC_model.fit(X_train, y_train)

# Print best parameters and scores
print('Best params:', DTC_model.best_params_)
print('Training Score:', DTC_model.score(X_train, y_train))
print('CV Score:', DTC_model.best_score_)
print('Test Score:', DTC_model.score(X_test, y_test))

# ✅ Extract the best estimator (the pipeline)
best_tree_model = DTC_model.best_estimator_.named_steps['DTC']

# 🎨 Visualize the pruned tree
plt.figure(figsize=(20, 10))
plot_tree(
    best_tree_model,
    filled=True,
    feature_names=X_train.columns if hasattr(X_train, 'columns') else None,
    class_names=True,
    rounded=True
)
plt.title("Pruned Decision Tree")
```

Evaluation

```python
from sklearn.metrics import classification_report, confusion_matrix
y_pred = DTC_model.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, cmap = 'viridis')
plt.title('Decision Tree Confusion Matrix')
print(classification_report(y_test, y_pred))
```

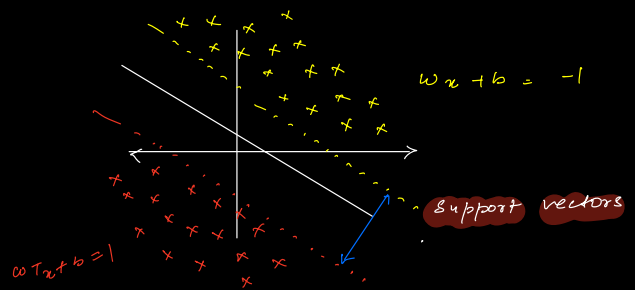
✓ 0.0s

	precision	recall	f1-score	support
0	0.91	0.92	0.92	53
1	0.96	0.94	0.95	90
accuracy			0.94	143
macro avg	0.93	0.93	0.93	143
weighted avg	0.94	0.94	0.94	143

SVM mostly classification

Support Vector machine is used for classification and Regression both.

Aim SVM tries to find the best boundary (hyperplane) that separates classes based on max distance b/w separating line and data points of both classes. (support vectors)



Hyperplane = $w^T x + b = 0$

$w^T x_1 + b = -1$

$w^T x_2 + b = 1$

$w^T (x_1 - x_2) = 2$

$\frac{w^T (x_1 - x_2)}{|w|} = \frac{2}{|w|}$

$\frac{2}{|w|} = \max$ ← maximize margin

$\frac{1}{2} \|w\|^2$ ← Minimize this

s.t. $y_i (w \cdot x_i + b) \geq 1$

Optimization in Real world (soft margin)

$\frac{1}{2} \|w\|^2 + C \sum \varepsilon_i$

C ↓ Allowed Errors

$\sum \varepsilon_i$ → Σ of distance of wrong points

Q. What is Kernel trick in SVM?

In classification problems, sometimes non linear data e.g. circle data can't be separated by a line, so in that case with kernel trick, SVM can map this data to higher dimension where straight hyperplane can separate the classes

Svm Practicals

SVC Support vector Classifier

```
svc_pipe = Pipeline(steps=[
    ('scaler', StandardScaler()),
    ('pca', PCA()),
    ('svc', SVC())
])
param_grid = {
    'pca__n_components': np.arange(1, X_train.shape[1]//3),
    'svc__C': np.logspace(0, 3, 10),
    'svc__kernel': ['rbf'],
    'svc__gamma': np.logspace(-4, -3, 10)
}
svc_model = GridSearchCV(svc_pipe, param_grid=param_grid, verbose=1, n_jobs=-1)
svc_model.fit(X_train, y_train)
print('Best params: {}'.format(svc_model.best_params_))
print('Training Score: {}'.format(svc_model.score(X_train, y_train)))
print('CV Score: {}'.format(svc_model.best_score_))
print('Test Score: {}'.format(svc_model.score(X_test, y_test)));
```

✓ 2.0s

```
Fitting 5 folds for each of 900 candidates, totalling 4500 fits
Best params: {'pca__n_components': np.int64(8), 'svc__C': np.float64(100.0), 'svc__gamma': np.float64(0.001), 'svc__kernel': 'rbf'}
Training Score: 0.9906103286384976
CV Score: 0.9906155950752394
Test Score: 0.972027972027972
```

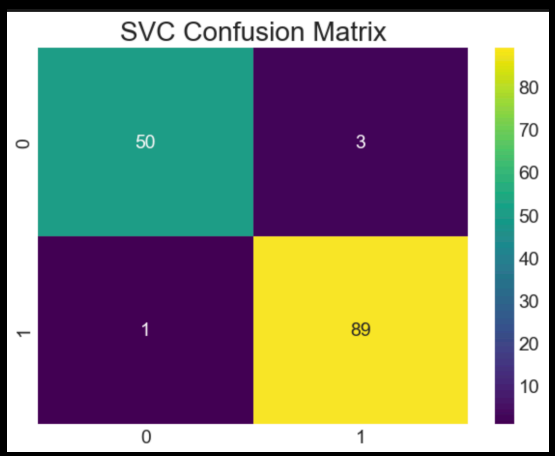
```
from sklearn.metrics import classification_report, confusion_matrix
y_pred = svc_model.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, cmap = 'viridis')
plt.title('SVC Confusion Matrix')
print(classification_report(y_test, y_pred))
```

✓ 0.0s

```
              precision    recall  f1-score   support

           0       0.98      0.94      0.96        53
           1       0.97      0.99      0.98        90

    accuracy                           0.97       143
   macro avg       0.97      0.97      0.97       143
weighted avg       0.97      0.97      0.97       143
```



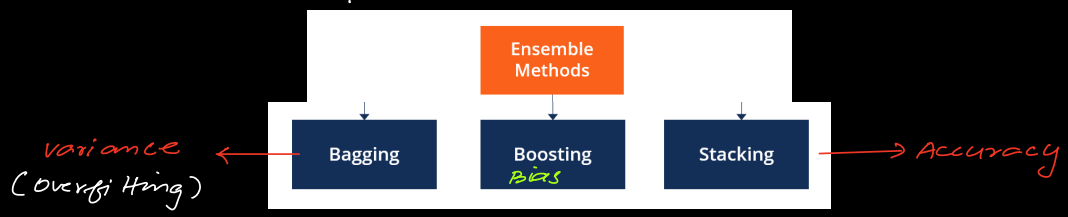
Ensemble Techniques

Combining multiple model together to improve performance.

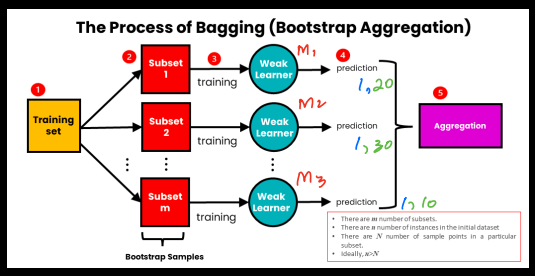
E.g., Wisdom of Crowd

DT_1 = NO, DT_2 = Yes, DT_3 = NO

Final Outcome = NO



Feature	Bagging (low Var)	Boosting (low Bias)
Definition	Combines models trained in **parallel** on random subsets	Combines models trained **sequentially**, each learning from previous errors
Goal	Reduce **variance** (avoid overfitting)	Reduce **bias** (avoid underfitting)
Model Dependency	Models are **independent**	Models are **dependent** (each builds on previous)
Data Sampling	Uses **bootstrapped** (random with replacement) subsets	Uses the **entire dataset**, but adjusts weights for errors
Final Prediction	**Average** (regression) or **majority vote** (classification)	**Weighted sum** of all weak models
Overfitting	Less prone to overfitting	Can overfit if too many models
Popular Algorithms	Random Forest	AdaBoost, Gradient Boosting, XGBoost, LightGBM



Classification ;
Majority Voting = 1

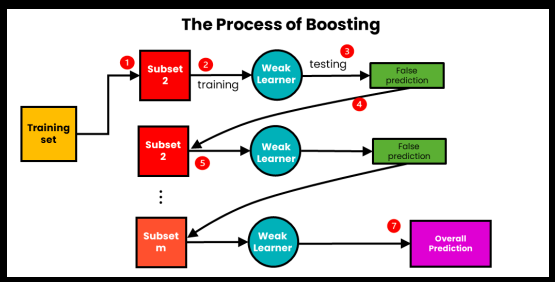
Regression ;
Average = $\frac{20+30+10}{3}$ = 20

where M_1, M_2, M_3 are different Models (Decision Tree)

The steps of bagging are as follows:

1. We have an initial training dataset containing n-number of instances.
2. We create a m-number of subsets of data from the training set. We take a subset of N sample points from the initial dataset for each subset. Each subset is taken with replacement. This means that a specific data point can be sampled more than once.
3. For each subset of data, we train the corresponding weak learners independently. These models are homogeneous, meaning that they are of the same type.
4. Each model makes a prediction.
5. Aggregating the predictions into a single prediction. For this, using either max voting or averaging.

Imp:
Bootstap Sampling: Randomly select samples (with Replace)



Boosting

Many models work on same subset, and work on it till it becomes strong

The steps of bagging are as follows:

1. We have an initial training dataset containing n-number of instances.
2. We create a m-number of subsets of data from the training set. We take a subset of N sample points from the initial dataset for each subset. Each subset is taken with replacement. This means that a specific data point can be sampled more than once.
3. For each subset of data, we train the corresponding weak learners independently. These models are homogeneous, meaning that they are of the same type.
4. Each model makes a prediction.
5. Aggregating the predictions into a single prediction. For this, using either max voting or averaging.

Bagging

1. Random Forest Classifier / Regressor

Boosting

1. Ada boost
2. Gradient
3. Xg Boost

Q. what is the main problem of DT?

Overfitting

→ Low Bias, High Variance

Bagging RANDOM FOREST (with Replacement)

Random Rows
Random features

is an ensemble ML model that builds multiple Decision Trees and combines their output to make more Accurate & Robust predictions

- It is an ensemble of multiple decision trees.
- Each tree is trained on a random subset of the data and a random subset of features.
- The final prediction is made by averaging (for regression) or majority voting (for classification).

Imp

How it reduces overfitting:

1. **Bagging (Bootstrap Aggregating):**
   Trains each tree on a different random subset of the training data, which reduces variance and prevents overfitting to specific data points.
2. **Random Feature Selection:**
   At each split, a random subset of features is considered, leading to diverse trees and less correlation between them.

Q1 Does RF require Norm / Standardization ?

No, because it is not a distance based algorithm like KNN

Q1

Random Forest

not impacted by outliers

KNN / Decision Tree

Impacted by outliers.

RANDOM FOREST PRACTICALS

Classifier

```
rdf_pipe = Pipeline(steps=[
    ('scaler', StandardScaler()),
    ('rdf', RandomForestClassifier())
])

param_grid = {
    'rdf__n_estimators': np.arange(200, 1001, 200),
    'rdf__max_depth': np.arange(1,4),
}

rdf_model = GridSearchCV(rdf_pipe, param_grid=param_grid, verbose=1, n_jobs=-1)
rdf_model.fit(X_train, y_train)
print('Best params: {}'.format(rdf_model.best_params_))
print('Training Score: {}'.format(rdf_model.score(X_train, y_train)))
print('CV Score: {}'.format(rdf_model.best_score_))
print('Test Score: {}'.format(rdf_model.score(X_test, y_test)));
```

✓ 6.9s

```
Fitting 5 folds for each of 15 candidates, totalling 75 fits
Best params: {'rdf__max_depth': np.int64(3), 'rdf__n_estimators': np.int64(600)}
Training Score: 0.9835680751173709
CV Score: 0.9554582763337894
Test Score: 0.9370629370629371
```

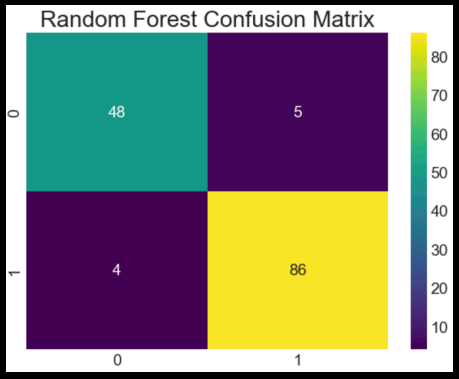
```
from sklearn.metrics import classification_report, confusion_matrix
y_pred = rdf_model.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, cmap = 'viridis')
plt.title('Random Forest Confusion Matrix')
print(classification_report(y_test, y_pred))
```

✓ 0.0s

```
              precision    recall  f1-score   support

           0       0.92      0.91      0.91        53
           1       0.95      0.96      0.95        90

    accuracy                           0.94       143
   macro avg       0.93      0.93      0.93       143
weighted avg       0.94      0.94      0.94       143
```



Regressor

```
from sklearn.ensemble import RandomForestRegressor

random_forest= RandomForestRegressor()
random_forest.fit(X_train, y_train)

#To get predictions
y_pred2 = decision_tree.predict(X_test)
```

✓ 0.2s

```
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# Evaluation Metrics
random_forest_mse = mean_squared_error(y_test, y_pred2)
random_forest_rmse = np.sqrt(random_forest_mse)
random_forest_r2_score = r2_score(y_test, y_pred2)

print(f"Mean Squared Error (MSE) using Random Forest Regressor: {random_forest_mse:.4f}")
print(f"Root Mean Squared Error (RMSE) using Random Forest Regressor: {random_forest_rmse:.4f}")
print(f"R² Score using Random Forest Regressor: {random_forest_r2_score:.4f}")
```

✓ 0.0s

```
Mean Squared Error (MSE) using Random Forest Regressor: 2947393.7148
Root Mean Squared Error (RMSE) using Random Forest Regressor: 1716.7975
R² Score using Random Forest Regressor: 0.8537
```

Boosting

Learn from failures of previous ones

- In boosting (like AdaBoost or Gradient Boosting), **all data points are used** in every iteration.
- But the model **assigns higher weights or error gradients** to previously misclassified or poorly predicted points, so the next model pays more attention to them.

Feature	AdaBoost (Add weights)	Gradient Boosting (GBM) (Residuals)	XGBoost (GBM + Regularization + optimization)
Core Idea	Focus on misclassified samples	Fit next model on residuals	Enhanced GBM with speed & regularization
Base Learner	Weak models (e.g., stumps)	Decision Trees (shallow/deep)	Decision Trees
Update Method	Increases weight on wrong predictions	Uses gradient of loss function	Uses gradient + Hessian (2nd-order derivative)
Overfitting Control	No regularization (can overfit)	Needs careful tuning	Built-in L1 & L2 regularization
Speed	Slow to moderate	Moderate	Fast (optimized, parallel, scalable)
Categorical Support	Manual encoding required	Manual encoding required	Manual encoding required
Best For	Simple classification tasks	General classification/regression	Large-scale, high-performance applications

Summary:

- **AdaBoost**: reweights misclassified examples → next model focuses more on them.
- **GBM**: fits new models to correct residuals (errors) → learns from mistakes step-by-step.

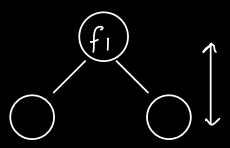
1. AdaBoost (mostly DT (stumps)) (weights)

adds more weight to Misclassified samples, so next model focusses more on them

	F_1	F_2	F_3	O/p	weight	New weights	Normalized weight (because we want out of (1))	Buckets
Correct	–	–	–	Yes	1/3	0.05	0.111	0 – 0.12
Wrong	–	–	–	No	1/3	0.349	0.777	0.2 – 0.9
Correct	–	–	–	Yes	1/3	0.05	0.111	0.05 – 0.12
Total =					1	0.449	New W / Total W	

STEPS

1. Find Total Error



$\varepsilon = 1/7$

2. performance of stump

$$P_s = \frac{1}{2} \log_e \left(\frac{1 - 1/7}{1/7} \right) = 0.895$$

3. Re calculate weights

For correct Records = weight $\times$ e^{-P_s}

$= 1/7 \times e^{-0.095} = 0.05$

For Incorrect Records = weight $\times$ e^{+P_s}

$1/7 \times e^{+0.095} = 0.349$

4. Normalize weights = $\frac{(\text{New weights})_i}{\Sigma \text{ Total weight}}$

5. Create Buckets = After creating Buckets the errors will fall into large Bucket which then is treated into by other model

* Note; even some (few) correct learners may pass down in buckets

Finally; The buckets with bigger numbers (errors) will be passed down to another stump and the entire process will Repeat

Imp:

- A **decision stump** is a tree with **only one split.**
- A **decision tree** can have **many splits and levels,** allowing more complex decision boundaries.

2. XG BOOST – Classifier (Black Box)

trains models sequentially, Each model corrects residual errors of the previous one

Example; Base model probab = 0.5

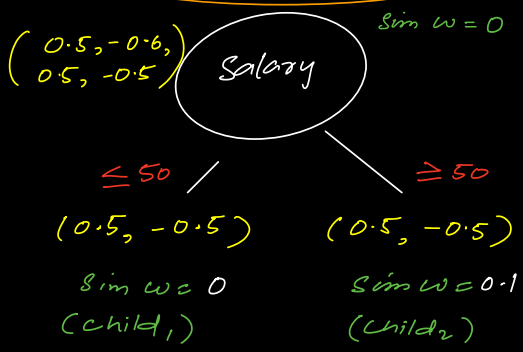
Salary	Credit	Approval	Residual
< 50	B	0	0 − 0.5 = −0.5
<= 50	B	1	0.5
= 50	G	0	−0.5
> 50k	N	1	0.5

STEPS

1. Create **binary decision** tree using the feature

2. Calculate Similarity weight

Imp. Always Binary DT



Residuals = $\frac{\sum (\text{Residual})^2}{\sum (pr(1-pr) + \lambda}$ Gradients; Hessians

when $\lambda = 1$

$$\text{Sim } w = \frac{(0.5 - 0.5)^2}{\sum 0.5(1-0.5) + \lambda}$$

Gradient = 0; Hessian

3. Calculate Information Gain

$$\text{Sim } w_{(\text{child 1})} + \text{Sim } w_{(\text{child 2})} - \text{Sim } w_{(\text{parent})}$$

$$0 + 0 - 0 = 0$$

Note; similarity score in XGBOOST tells how valuable a split / Node is

Finally;

$$\sigma\left[0 + \alpha_1(DT_1) + \alpha_n \alpha_2 (DT_n)\right]$$

Base Model; Sigmoid; learning Rate

XG Boost Classifier Summary

✅ Step-by-Step Explanation:

1. **Start with Predictions = 0**
   - For all data points, we assume the initial prediction is 0.
   - This translates to 0.5 probability after applying the sigmoid.
2. **Use Binary Decision Trees**
   - XGBoost uses binary trees to split the data.
   - But instead of using Gini or entropy, it uses **Gradient & Hessian** (like loss derivatives).
3. **Compute Gradients and Hessians**
   - For each sample, we compute:
     - **Gradient (g)**: how much the current prediction is wrong.
     - **Hessian (h)**: how confident we are in that error.
   - These are like "error signals" to guide the next tree.
4. **Similarity Score (Node Quality)**
   - For each node (group of samples), compute:

$$\text{Similarity} = -\frac{(\sum g)^2}{\sum h + \lambda}$$

   - This is like a custom impurity score — higher similarity = purer node.
5. **Information Gain for a Split**
   - When considering a split into left and right nodes:

$$\text{Gain} = \text{Similarity(left)} + \text{Similarity(right)} - \text{Similarity(parent)}$$

6. **Build the Tree**
   - Repeat the process to build the tree by adding splits based on gain.
7. **Update Predictions**
   - For each sample, update the prediction using:

$$\hat{y}_{\text{new}} = \hat{y}_{\text{old}} + \alpha \cdot f(x)$$

   - Where `f(x)` is the leaf value (based on gradient math), and `α` is the learning rate.
8. **Apply Sigmoid**
   - Final prediction = sigmoid of the accumulated outputs:

$$\text{Prob} = \sigma(\hat{y}) = \frac{1}{1 + e^{-\hat{y}}}$$

Imp.

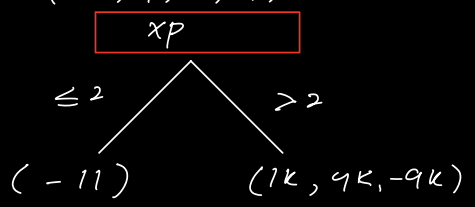
"In XGBoost for classification, instead of using impurity like Gini or entropy, it builds binary decision trees using gradients and second-order values (called Hessians). Each new tree focuses on fixing the mistakes made by the previous one. At every split, XGBoost checks how similar the grouped data is — the more similar, the better the split. Then it calculates how much better the prediction becomes (called gain). Finally, the model updates the predictions using a learning rate and applies the sigmoid function to convert scores into probabilities."

2a. XG Boost – Regressor

Xp	Gap	Salary	R_1
2	Yes	40k	-11k
2.5	Yes	42k	-9k
3	NO	52k	1k
3.5	Yes	60k	9k

Base model = 51k

(-11, 9, 1, -9)



$$\text{Similarity weight} = \frac{\Sigma(\text{Residuals})^2}{\text{No. of Residuals} + \lambda}$$

when $\lambda = 1$

$$Sw_1 = \frac{121}{1+1} = 60.5$$

$$Sw_2 = \frac{121}{3+1} = 30.25$$

$$Sw_{parent} = \frac{-10}{4} = -2.5$$

$$\text{Information Gain} = 60.5 + 30.25 - 2.5$$

$$= 88.25$$

Finally = Base pred + α_1 (DT_1) (mean).

$$= 51 + \alpha_1 \left(\frac{-11}{1}\right) + \alpha_2 \left(\frac{1-9+9}{3}\right)$$

Note;

The entire procedure is same but Sigmoid function is not needed.

XG Boost Regressor Summary

✅ XGBoost for Regression – Step-by-Step (Simplified)

1. Start with Initial Predictions

- All predictions start with a **constant value**, usually the **mean** of the target variable.
- No sigmoid is applied here (unlike classification), since output is **continuous**.

2. Use Binary Decision Trees

- XGBoost builds **shallow binary decision trees** to predict **residual errors** — i.e., how far the current prediction is from the actual target.

3. Compute Gradient and Hessian

- For each sample, compute:
  - **Gradient (g):** How far off is the prediction from the actual value?
    Example: `g = (ŷ - y)`
  - **Hessian (h):** Second derivative of the loss (for squared error, it's just 1).

These guide the tree where to focus next.

4. Compute Similarity Score (Node Quality)

For each node (group of samples), compute:

$$\text{Similarity} = -\frac{(\sum g)^2}{\sum h + \lambda}$$

- A high similarity score = better node = low variance = good to split.

5. Information Gain for Split

When trying a split:

$$\text{Gain} = \text{Similarity(left)} + \text{Similarity(right)} - \text{Similarity(parent)}$$

- Choose the **split with highest gain**.

6. Build the Tree

- Repeat steps 3–5 to grow the tree (until stopping conditions are met).

7. Update Predictions

For each sample, update prediction with:

$$\hat{y}_{\text{new}} = \hat{y}_{\text{old}} + \alpha \cdot f(x)$$

- `f(x)` is the output from the current tree's leaf.
- `α` is the **learning rate**.

🚫 No Sigmoid Needed

- Since this is regression, we **don't apply sigmoid** at the end.
- Final output = sum of all tree outputs → your predicted value.

🔷 **XGBoost for Regression** builds **binary decision trees** using **gradients and Hessians** (not Gini/entropy).
🔷 Each tree tries to correct the **residual errors** of the previous one.
🔷 At every split, it calculates a **similarity score** and **gain** to choose the best split.
🔷 Final predictions are updated by **adding tree outputs**, scaled by a **learning rate**.
🔷 No sigmoid — the output is the **direct predicted value**.

Boosting Practicals

XG Boost Classifier

```
# Optional as it consumes alot of time**

xgb_pipe = Pipeline(steps=[
    ('scaler', StandardScaler()),
    #('pca', PCA()),
    ('xgb', XGBClassifier())
])
param_grid = {
    #'pca__n_components': np.arange(1, X_train.shape[1]//3),
    'xgb__n_estimators': [100],
    'xgb__learning_rate': np.logspace(-3, 0, 10),
    'xgb__max_depth': np.arange(1, 6),
    'xgb__gamma': np.arange(0, 1.0, 0.1),
    'xgb__reg_lambda': np.logspace(-3, 3, 10)
}
xgb_model = GridSearchCV(xgb_pipe, param_grid=param_grid, verbose=1, n_jobs=-1)
xgb_model.fit(X_train, y_train)
print('Best params: {}'.format(xgb_model.best_params_))
print('Training Score: {}'.format(xgb_model.score(X_train, y_train)))
print('CV Score: {}'.format(xgb_model.best_score_))
print('Test Score: {}'.format(xgb_model.score(X_test, y_test)));
```

✓ 1m 48.0s

```
Fitting 5 folds for each of 5000 candidates, totalling 25000 fits
Best params: {'xgb__gamma': np.float64(0.30000000000000004), 'xgb__learning_rate': np.float64(0.46415888336127775)
Training Score: 0.9976525821596244
CV Score: 0.9695212038303694
Test Score: 0.951048951048951
```

XG Boost Regressor

```
xgb = xgb.XGBRegressor()
xgb.fit(X_train, y_train)

#To get predictions
y_pred5 = xgb.predict(X_test)
```

✓ 0.2s

```
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# Evaluation Metrics
xgb_reg_mse = mean_squared_error(y_test, y_pred5)
xgb_reg_rmse = np.sqrt(xgb_reg_mse)
xgb_reg_r2_score = r2_score(y_test, y_pred5)

print(f"Mean Squared Error (MSE) using XGBoost Regressor: {xgb_reg_mse:.4f}")
print(f"Root Mean Squared Error (RMSE) using XGBoost Regressor: {xgb_reg_rmse:.4f}")
print(f"R² Score using XGBoost Regressor: {xgb_reg_r2_score:.4f}")
```

✓ 0.0s

```
Mean Squared Error (MSE) using XGBoost Regressor: 4046267.4282
Root Mean Squared Error (RMSE) using XGBoost Regressor: 2011.5336
R² Score using XGBoost Regressor: 0.7991
```

Find Best Model

To Get Best Performing Model

Classification

```
models = pd.DataFrame({
    'Model' : ['Linear Regression', 'Decision Tree', 'Random Forest',
               'Gradient Boosting', 'KNN', 'XGBoost'],
    'RMSE' : [linear_reg_rmse, decision_tree_rmse, random_forest_rmse,
              gradient_boosting_rmse, knn_rmse, xgb_reg_rmse],
    'r2_score' : [linear_reg_r2_score, decision_tree_r2_score, random_forest_r2_score,
    gradient_boosting_r2_score, knn_r2_score, xgb_reg_r2_score]
})

models.sort_values(by='RMSE', ascending=True)
```

✓ 0.0s

	Model	RMSE	r2_score
0	Linear Regression	1421.610974	0.853680
3	Gradient Boosting	1628.614610	0.868325
1	Decision Tree	1715.761946	0.853856
2	Random Forest	1716.797517	0.853680
4	KNN	1786.086656	0.841631
5	XGBoost	2011.533601	0.799127

Regressor

python Copy Edit

```
import pandas as pd

# Compare model performance
models = pd.DataFrame({
    'Model': ['Linear Regression', 'Decision Tree', 'Random Forest',
              'Gradient Boosting', 'KNN', 'XGBoost'],
    'RMSE': [linear_reg_rmse, decision_tree_rmse, random_forest_rmse,
             gradient_boosting_rmse, knn_rmse, xgb_reg_rmse],
    'R2 Score': [linear_reg_r2_score, decision_tree_r2_score, random_forest_r2_score,
                 gradient_boosting_r2_score, knn_r2_score, xgb_reg_r2_score]
})

# Sort by RMSE (ascending — better models on top)
models.sort_values(by='RMSE', ascending=True)
```

Plot it

📊 Bonus Tip – Plot it:

python Copy Edit

```
import seaborn as sns
import matplotlib.pyplot as plt

sns.barplot(x='RMSE', y='Model', data=models.sort_values(by='RMSE'))
plt.title('Model Comparison (Lower RMSE is Better)')
plt.show()
```

UNSUPERVISED LEARNING

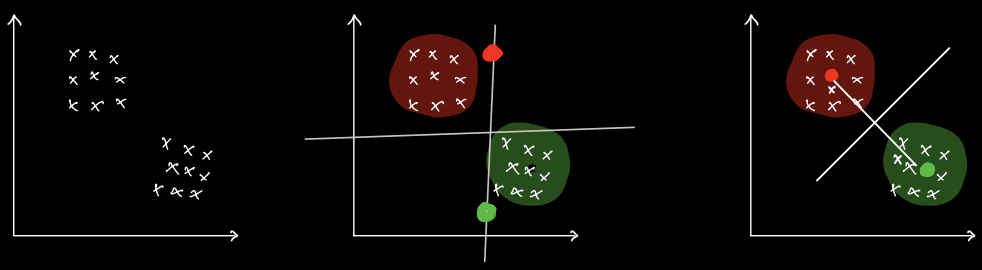
is a type of a machine learning where the algorithm is not given any Labels (Targets) It has to find patterns or structure from the data on its own

1. Clustering
   - K means
   - Hierarchical
   - DB Scan
2. Dimensionality Reduction
   - PCA

centroids KMEANS clustering

is an unsupervised algorithm used to group similar Data points

K ⟶ no. of clusters (with centroid)



STEPS

I. we tay K values; Lets say K = 2

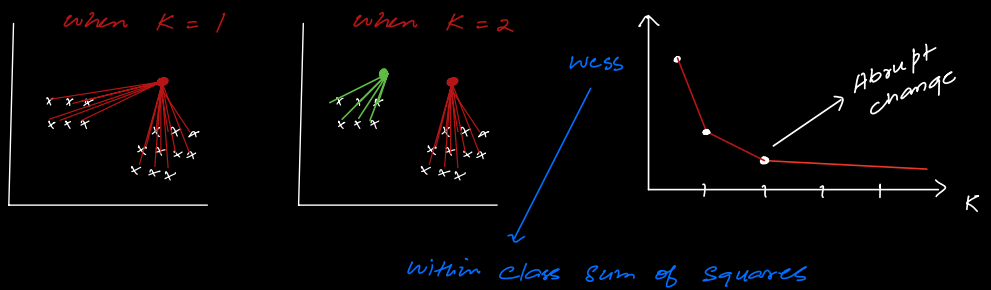
II. then we initialize 2 Centroids

III. check distance (Euclidean Distance) b/w centroid and Data points

IV. Compute the mean, so centroid is updated

Q. How to find optimal value of K ?

1. **Elbow Method** is a method, to find optimal value of K (post which our model's performance doesn't change significantly).



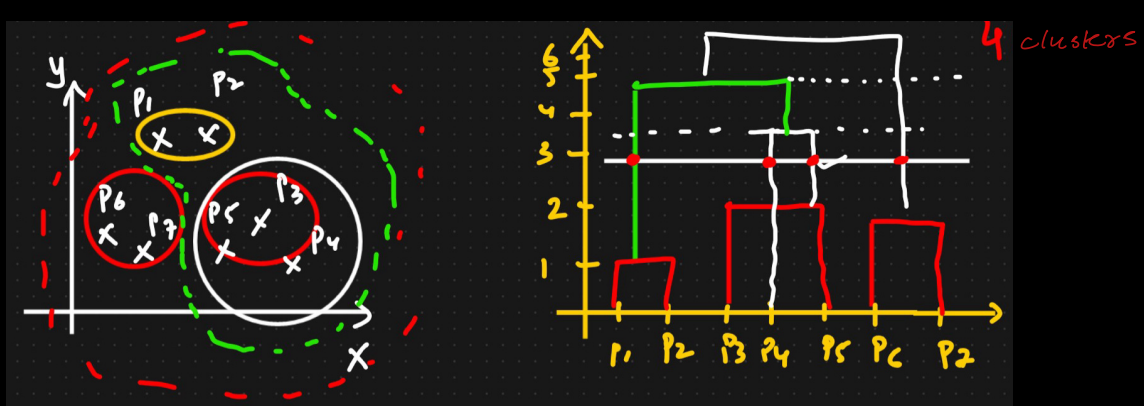
prone to outliers

HIERARCHICAL CLUSTERING

Initially every data point is considered as a cluster and as we go on, the data points near to each other are grouped into one till we have just one cluster

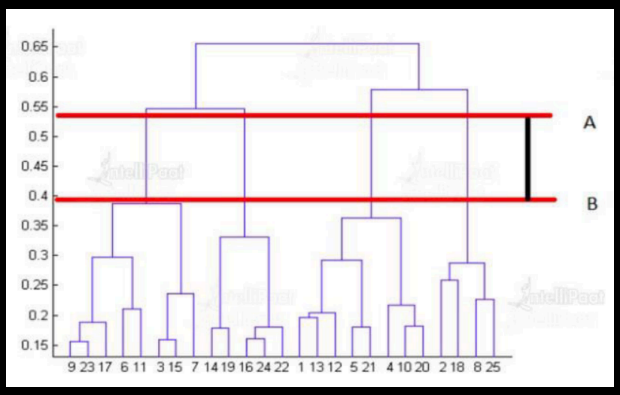
Approaches

1. **Agglomerative** — Bottom to top
2. **Divisive** — Top — Bottom



Q// How to find optimal number of Groups?

we need to find the longest vertical line that has no horizontal line passing through it.



Q// Which one is faster

- Kmeans is faster than Hierarchical
- Large Datasets = Kmeans

Feature	K-Means	Hierarchical Clustering
Need to predefine k?	✅ Yes	❌ No (dendrogram helps decide)
Scalability	✅ Fast for large datasets	❌ Slower, best for small datasets
Output	Flat clusters	Tree-like structure (dendrogram)

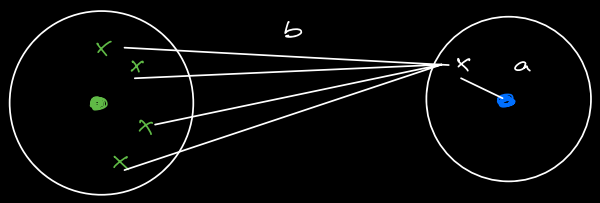
Validation of Clustering

- Silhoutte Score ; is a metric used to evaluate how well clusters are formed in clustering algorithms

$$\text{Silhoutte Score} = \frac{b - a}{\max(a, b)}$$

Where ;

> ✅ "a = average distance of a point to all other points in its **own cluster**"
> ✅ "b = average distance of that point to all points in the **nearest neighboring cluster**"



In Good model

$b_i >> a_i$

Imp; Silhouette score ranges from $-1 \rightarrow 1$

which means $+1$ = Good model

-1 = Bad model

Q. what is K means ++ ?

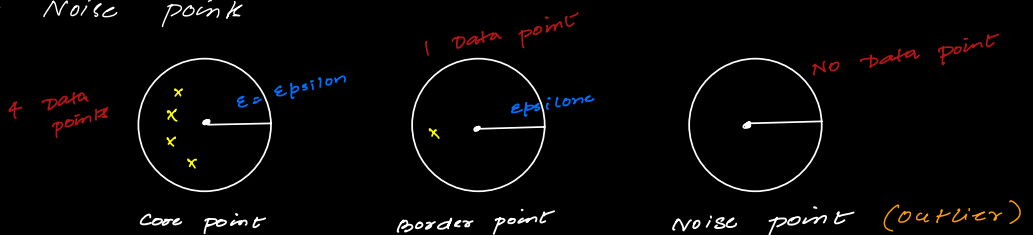
It makes sure centroids are initialise away from Data points

Unsupervised DB SCAN (outlier, Density Based)

Density Based clustering algorithm that groups together data points that are closely packed / high Density neighbours (points with many nearby neighbours), and labels points in low-density regions as outliers.

Unlike Kmeans, it can exclude outlier alone.

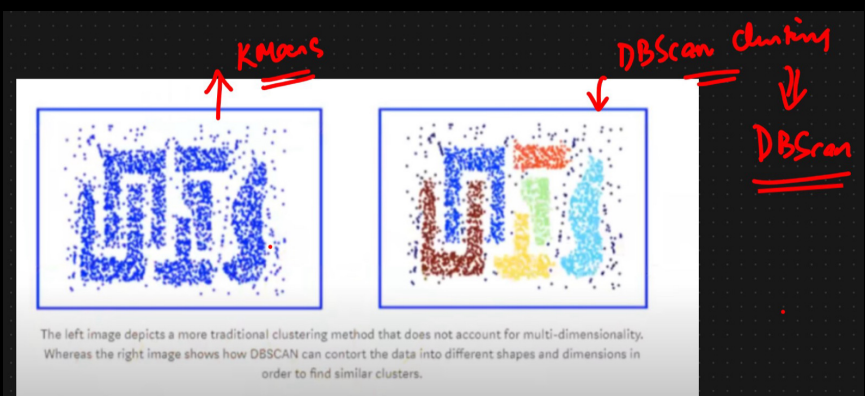
1. Min points ⟶ MinPts = 4 (Hyperparameter)
2. Core points
3. Border points
4. Noise point



Term	Meaning
ε (Epsilon)	Radius defining a neighborhood around a point.
MinPts	Minimum number of points needed within ε to define a dense region.
Core Point	A point that has at least MinPts within its ε-radius.
Border Point	A point that lies within the ε-radius of a core point but doesn't itself have enough neighbors to be a core point.
Noise Point	A point that is **not** a core point or a border point. It lies in a sparse region.

🔍 DBSCAN vs K-Means:

Feature	DBSCAN	K-Means
Cluster Shape	Arbitrary shapes	Only spherical
Need to specify K?	❌ No	✅ Yes
Outlier Detection	✅ Yes	❌ No



DB Scan more preffered.

↳ Use Always.

Unsupervised Practicals

K means Practicals

```python
# k-means with some arbitrary k

from sklearn.cluster import KMeans

wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init = 'k-means++', random_state=0)
    kmeans.fit(rfm_df_scaled)
    wcss.append(kmeans.inertia_)
```

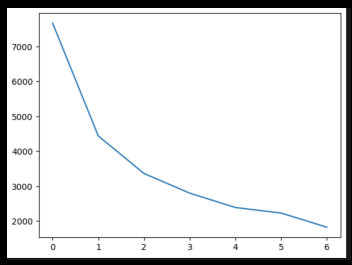
Elbow Method

```python
# Elbow-curve/SSD

ssd = []
range_n_clusters = [2, 3, 4, 5, 6, 7, 8]
for num_clusters in range_n_clusters:
    kmeans = KMeans(n_clusters=num_clusters, max_iter=50)
    kmeans.fit(rfm_df_scaled)

    ssd.append(kmeans.inertia_)

# plot the SSDs for each n_clusters
plt.plot(ssd)
```



Silhouette Score

```python
# Silhouette analysis is used to find performance of the clustering algorithm
# e.g., whether our model performance is valid on 4 cluster by elbow method

range_n_clusters = [2, 3, 4, 5, 6, 7, 8]

for num_clusters in range_n_clusters:

    # intialise kmeans
    kmeans = KMeans(n_clusters=num_clusters, max_iter=50)
    kmeans.fit(rfm_df_scaled)

    cluster_labels = kmeans.labels_

    # silhouette score
    silhouette_avg = silhouette_score(rfm_df_scaled, cluster_labels)
    print("For n_clusters={0}, the silhouette score is {1}".format(num_clusters, silhouette_avg))
```

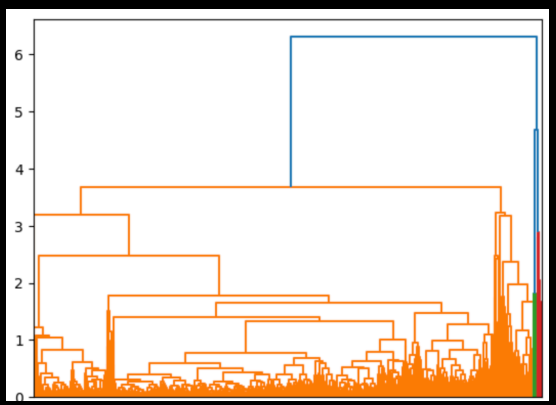
✓ 1.2s

```
For n_clusters=2, the silhouette score is 0.402908664484819
For n_clusters=3, the silhouette score is 0.5087593101969974
For n_clusters=4, the silhouette score is 0.47882975419861457
For n_clusters=5, the silhouette score is 0.46462518625510935
For n_clusters=6, the silhouette score is 0.4349637910757859
For n_clusters=7, the silhouette score is 0.414674221900517
For n_clusters=8, the silhouette score is 0.40130271413920454
```

Hierarchical Clustering

```python
# Average linkage

mergings = linkage(rfm_df_scaled, method="average", metric='euclidean')
dendrogram(mergings)
plt.show()
```



Dendrogram

DBSCAN

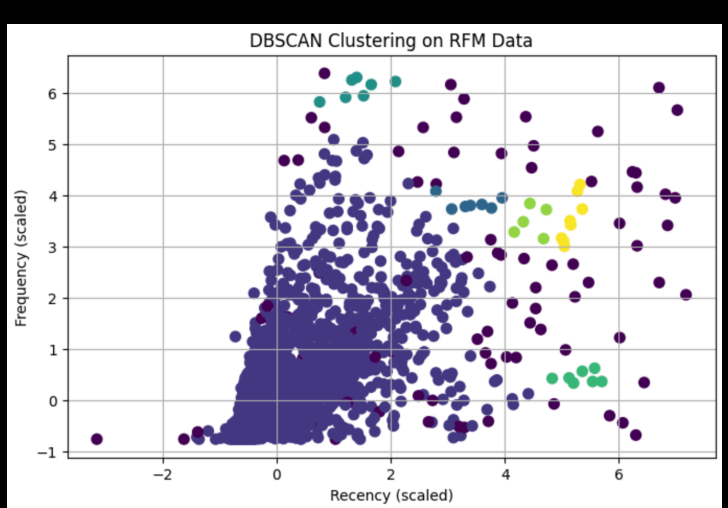
```python
from sklearn.cluster import DBSCAN
import matplotlib.pyplot as plt

# 1. Apply DBSCAN clustering
dbscan = DBSCAN(eps=0.5, min_samples=5)  # ✅ adjust eps if needed based on k-distance plot
dbscan_labels = dbscan.fit_predict(rfm_df_scaled)

# 2. Add cluster labels to original DataFrame
rfm_df['Cluster'] = dbscan_labels

# 3. Visualize DBSCAN clustering (2D plot: Recency vs Frequency)
plt.figure(figsize=(8, 5))
plt.scatter(rfm_df_scaled.iloc[:, 0], rfm_df_scaled.iloc[:, 1],
            c=dbscan_labels, cmap='viridis', s=50)
plt.title('DBSCAN Clustering on RFM Data')
plt.xlabel('Recency (scaled)')
plt.ylabel('Frequency (scaled)')
plt.grid(True)
plt.show()

# 4. View cluster distribution
print(rfm_df['Cluster'].value_counts())
```



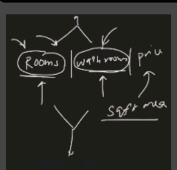
Dimensionality Reduction

refers to techniques for reduction of number of input variables in training data

FEATURE EXTRACTION SUMMARIZED VERSION

Feature Extraction aims to reduce the number of features in a dataset by creating new features from the existing ones (and then discarding the original features).

These new reduced set of features should then be able to summarize most of the information contained in the original set of features. In this way, a summarised version of the original features can be created from a combination of the original set



TECHNIQUES

Linear Combination

PRINCIPAL COMPONENT ANALYSIS
INDEPENENT COMPONENT ANALYSIS
T-DISTRIBUTED STOCHASTIC NEIGHBOR EMBEDDING

Non Linear Combination

KERNAL PCA

link = https://towardsdatascience.com/feature-extraction-techniques-d619b56e31be

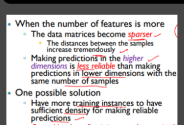
CURSE OF DIMENSIONALITY

OverFitting Synonym

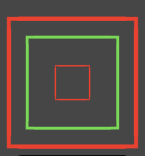
Disadvantages

Performance Decrease
Complex Computation

The Concept of COD states
The Model Performance Tends to increase only untill a certain number of features are added



4 = Max Performance
5 = No Use of Extra One



Optimal Features
Less or More Features

DIMENSIONALITY REDUCTION

Feature Extraction
PCA
LDA
TSNE

WHY DIMENSIONAL REDUCTION

Dimensionality Reduction is performed because

- Reduces Overfitting

Decreases the complexity of the model and its inference

- Simple models are robust on small datasets
- Fewer features

- Better idea to understand the underlying process
- Reduced memory and computation
- Visualization

- 2D and 3D data can be analysed and visualized for patterns and outliers

Pros	Cons
Memory saving	Loss of information
Speed up training	Performance might go down
May filter our noise	

PCA

PCA

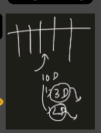
UnSupervised | No Labels | Find Lower D Hyper Plane | Preserve Variance

Principal component analysis, is a dimensionality-reduction method that is often used to reduce the dimensionality of large data sets

Principal component analysis, that transforms the data to a new coordinate system such that the greatest variance by scalar projection of the data comes to lie on the first coordinate (called the first principal component), the second greatest variance on the second coordinate, and so on.

Advantages
Less Space
Fast Execution
Visualization

Easy Plotting

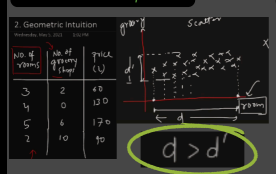


Principal Components <= Number of Features

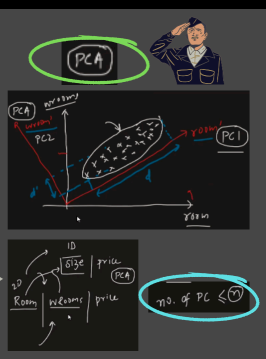
GOAL to find the combinations of the variables that reduce the dimensionality and preserve the variance in the data

PCA GEOMETRIC INTUITION

Problem 1
How to Select a feature?
More Variance / Importance



Problem 2
Select a feature when d = d'?
i.e both features are important

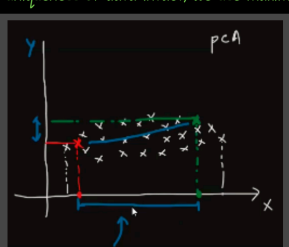


IMPORTANCE OF VARIANCE

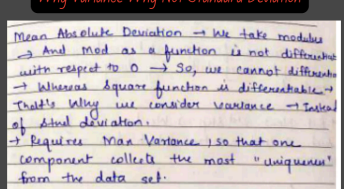
Variance ∝ Spread of data

When reducing Dimension (3D - 2D) The distance between points gets disturbed.

So to keep uniqueness of data intact, we use maximum variance



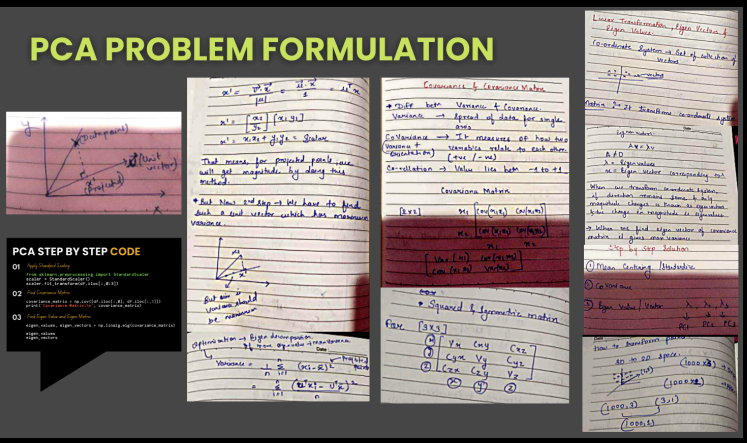
Why Variance Why Not Standard Deviation



INTUITION

- Goal
  - Project the data into lower dimensional hyperplane which preserves maximum variance in the data
  - Usually we select the number of Principal components that preserves 95% variance
- What matrix will explain the variance in the data with multiple features?
  - Answer: Covariance matrix
- Covariance matrix
  - top singular vectors of Covariance matrix are the directions of maximal variance in the data
  - the associated singular values are equal to these variances

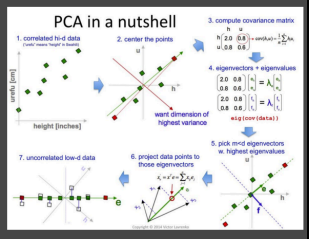
PCA PROBLEM FORMULATION



EIGEN VALUES AND VECTORS

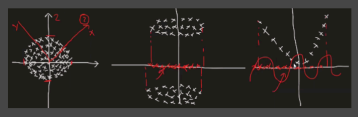
Eigen decomposition – Computing Eigenvectors and Eigenvalues
The eigenvectors and eigenvalues of a covariance (or correlation) matrix represent the "core" of a PCA:

The Eigenvectors (principal components) determine the directions of the new feature space, and the eigenvalues determine their magnitude.
In other words, the eigenvalues explain the variance of the data along the new feature axes. It means corresponding eigenvalue tells us that how much variance is included in that new transformed feature.
To get eigenvalues and Eigenvectors we need to compute the covariance matrix. So in the next step let's compute it.



PCA LIMITATIONS

- ⚠ *Sensitive to ouliers*
- ⚠ *Scaling of data is needed*
- ⚠ *Assumes that the features are linearly dependent*
  - *The transformed variables are linear combinations of original variables*
- ⚠ *When we have these Shapes like above*



PCA APPLY TIP

First, try to train the system with original data before resorting to dimensionality reduction ◦ If the training is too slow, then resort to dimensionality reduction.

PCA PRACTICALS

Compare Before and After Results of Mode:

```
from sklearn.decomposition import PCA
pca = PCA(n_components = 200)
X_train_trf = pca.fit_transform(X_train)
X_test_trf = pca.transform(X_test)
X_train_trf.shape
```

For 2D n_components = 2
For 3D, n_components = 2
None = All Features

To Find Best n_components in PCA

```
for i in range(1,785):
pca = PCA(n_components=i)
X_train_trf = pca.fit_transform(X_train)
X_test_trf = pca.transform(X_test)
```

Best n_components

To Find Eigen Value and Eigen Vectors

```
Eigen Value = pca.explained_variance_
Eigen Vector = pca.components_.shape
pca.explained_variance_ratio_
np.cumsum(pca.explained_variance_ratio_)
#plotiing plt.plot(np.cumsum(pca.explained_variance_ratio_))
```

Visualization

LDA

LDA

Multi Class Classification Algorithm | <=C-1 | Supervised | Dimensionality Reduction

Linear Discriminant Analysis is a dimensionality reduction technique that is commonly used for supervised classifiation problems. It is used for modelling difference in groups i.e. separating two or more classes.

In Dimensionality Reduction, As a result of applying the LDA algorithm, we get a new feature (set which can be used for prediction of group membership.

Grouping of Classes

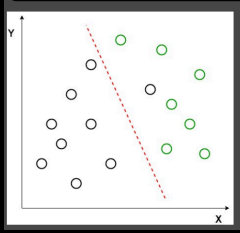
For example,
we have two classes and we need to separate them efficiently. Classes can have multiple features. Using only a single feature to classify them may result in some overlapping as shown in the below figure. So, we will keep on increasing the number of features for proper classification.

AIM

he aim of LDA is to maximize the between-class variance and minimize the within-class variance, through a linear discriminant function

EXAMPLE

Suppose we have two sets of data points belonging to two different classes that we want to classify. As shown in the given 2D graph, when the data points are plotted on the 2D plane, there's no straight line that can separate the two classes of the data points completely. Hence, in this case, LDA (Linear Discriminant Analysis) is used which reduces the 2D graph into a 1D graph in order to maximize the separability between the two classes.

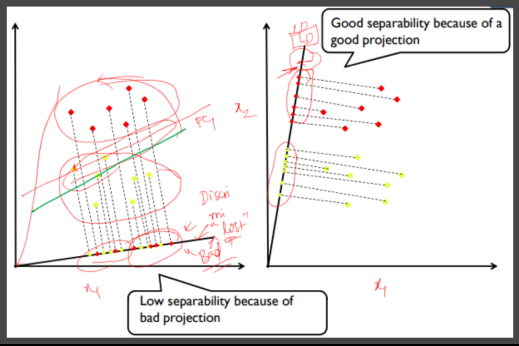


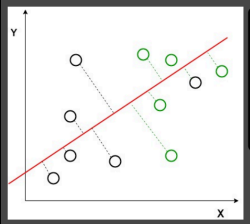
Here, Linear Discriminant Analysis uses both the axes (X and Y) to create a new axis and projects data onto a new axis in a way to maximize the separation of the two categories and hence, reducing the 2D graph into a 1D graph.

Two criteria are used by LDA to create a new axis:

- Maximize the distance between means of the two classes.
- Minimize the variation within each class.

OBJECTIVE





In this graph, it can be seen that a new axis (in red) is generated and plotted in the 2D graph such that it maximizes the distance between the means of the two classes and minimizes the variation within each class. In simple terms, this newly generated axis increases the separation between the data points of the two classes. After generating this new axis using the above-mentioned criteria, all the data points of the classes are plotted on this new axis and are shown in the figure given below.

WHEN DOES IT NOT WORK

Linear Discriminant Analysis fails when the mean of the distributions are shared, as it becomes impossible for LDA to find a new axis that makes both the classes linearly separable. In such cases, we use non- LDA

Quadratic Discriminant Analysis (QDA) | Flexible Discriminant Analysis (FDA) | Regularized Discriminant Analysis (RDA)

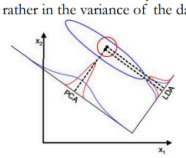
LDA SUMMARY

- Compute the N-dimensional mean vectors for the different classes from the dataset.
- Compute the scatter matrices (in-between-class and within-class scatter matrix).
- Compute the eigenvectors $(e_1, e_2, \dots, e_N)$ and corresponding eigenvalues $(\lambda_1, \lambda_2, \dots, \lambda_d)$ for the scatter matrices.
- Sort the eigenvectors by decreasing eigenvalues and choose k eigenvectors with the largest eigenvalues to form a N×k dimensional matrix **W** (where every column represents an eigenvector).
- Use this N×k eigenvector matrix to transform the samples onto the new subspace. This can be summarized by the matrix multiplication: $Z = X \times W$ (where X is a m×n-dimensional matrix representing the m samples, and Z are the transformed m×k-dimensional samples in the new subspace).

LIMITATIONS OF LDA

- **LDA produces at most C-1 feature projections**
  - If the classification error estimates establish that more features are needed, some other method must be employed to provide those additional features
- **LDA is a parametric method since it assumes unimodal Gaussian likelihoods**
  - If the distributions are significantly non-Gaussian the LDA projections will not be able to preserve any complex structure of the data, which may be needed for classification.

- LDA will fail when the discriminatory information is not in the mean but rather in the variance of the data



LDA PRACTICALS

Compare Before and After Results of Model:

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

lda = LinearDiscriminantAnalysis(n_components=2)
X_train = lda.fit_transform(X_train, y_train)
X_test = lda.transform(X_test)
```

(< n_classes - 1) for dimensionality reduction.

Visualization

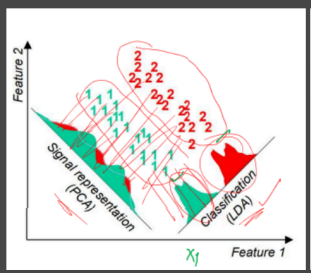
```
plt.scatter(X_train[:,0],X_train[:,1],c=y_train,cmap='rainbow')
```

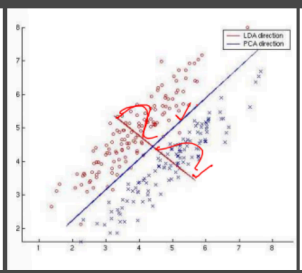
GOAL

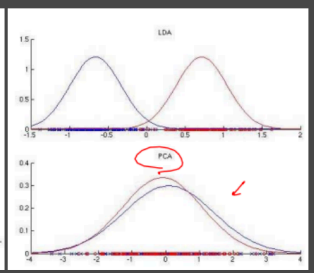
to find the combinations of the variables that reduce the dimensionality and preserve the discrimination in the data

DIFFERENCE BETWEEN PCA AND LDA

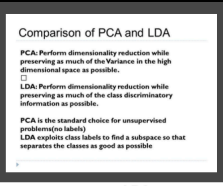
Preserving Variance vs Preserving Class discriminatory information



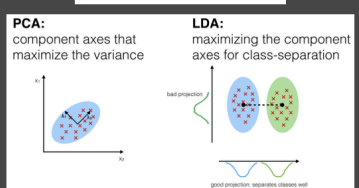




	PCA	LDA
Type of Transformation	Linear	Linear
Supervised vs Un-supervised	Un-supervised	Supervised
Relationship	Relationship between independent variables	Relationship between dependent and independent Variable
Objective	Capture the variability by finding Principal Components	Class separation by identifying a lower dimensional space which has better discriminatory power



Name	Method	Great For
PCA	Supervised	Reducing multicollinearity, feature extraction for linear data, data exploration
LDA	Unsupervised	Linearly separable dataset, categorizing two or more groups
t-SNE	Supervised	Visualizing high-dimensional space, feature extraction of non-linear high-dimensional data



PCA performs better in case where number of class is less. Whereas LDA works better with large dataset having multiple classes; class separability is an important factor while reducing dimensionality.

DIFFERENCE BETWEEN FS, FC AND FE

What is the difference between Feature Construction and Feature Extraction?

Feature Construction = Manual Process

Feature Extraction = Automatic Process (Programatically)

What is the main difference between Feature Extraction and Feature Selection?

Feature selection aims instead to rank the importance of the existing features in the dataset and discard less important ones (no new features are created)

What is the purpose of reducing Features?

To Avoid Overfitting

To save Space and Time