



Tajamul Khan

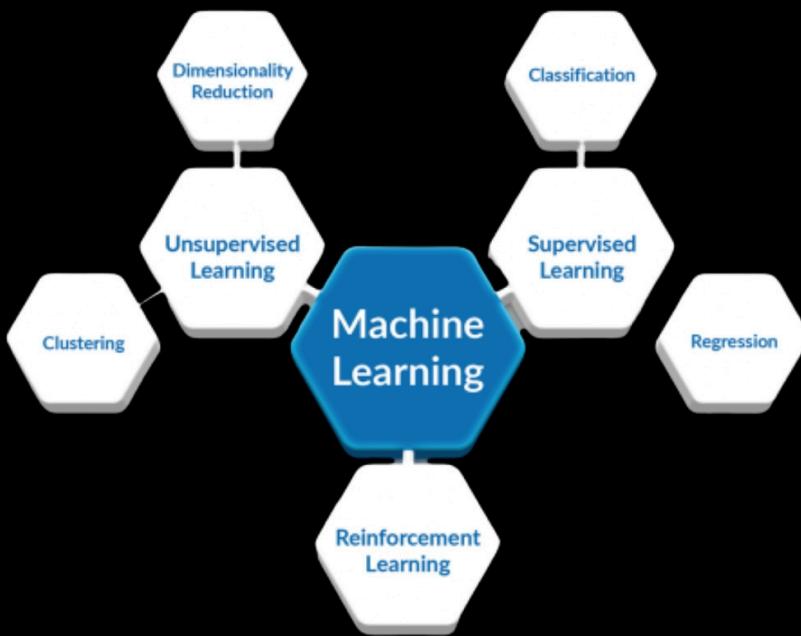
Complete Machine Learning Notes



@Tajamulkhan



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, R²
- 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 continuous number

Factor Notation Linear Relation b/w x and y

Standard

Simple
equation

$$y = \theta_0 + \theta_1 x$$

Independent variables Slope Dependent variables

Coeficient

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 $m \rightarrow$ steeper incline
 - A lower $m \rightarrow$ flatter line
 - $m = 0 \rightarrow$ 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 $y = c + m_1 x_1 + m_2 x_2$

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

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

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

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

Number of Data points Actual 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.

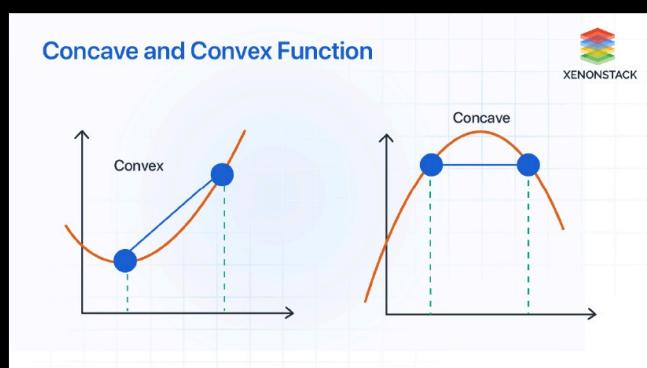
◆ Key Differences

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 $y = mx + c$ Get Cost Fx
- III. calculate Partial Derivatives cost function (Gradients) for each parameter $\frac{\partial J}{\partial m}$, $\frac{\partial J}{\partial c}$
These derivatives will tell us how to change m and c to reduce the error

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

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

(Directions)

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 \times slope

where α = Learning Rate → why using (-)

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

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

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

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

Large Datasets (use)

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.

```
from sklearn.linear_model import SGDClassifier
model = SGDClassifier(loss='log_loss', learning_rate='optimal')
model.fit(X, y)
```

```
from sklearn.linear_model import SGDRegressor
model = SGDRegressor(loss='squared_error', learning_rate='optimal')
model.fit(X, y)
```

SGD = Stochastic Gradient Descent



@Tajamulkhan

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

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
prediction accuracy

$$R^2 = 1 - \frac{\text{Residual sum of squares}}{\text{Total sum of squares}} = 1 - \frac{SS_{\text{res}}}{SS_{\text{total}}}$$

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

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:	
$R^2 = 0.90$	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

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

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

$$\frac{1 - (1 - 0.8) \cdot 100 - 1}{100 - 5 - 1} / 100 - 5 - 1$$

$$1 - (0.2) \cdot \frac{99}{99} = 0.789362$$

→ This makes sure we aren't going above R^2 either same or low.

MAE Average absolute difference between predicted and observed Data points.

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

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

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: $100^2 = 10,000 \leftarrow$ Huge impact
 - MAE: $|100| = 100 \leftarrow$ Not as dramatic

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)

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

Model performs well on Train Data = High Bias

Variance = Error due to Complexity of model { memorizes
Testing Error more sensitivity to training data { noise

💡 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

EXAMPLES;

✓ 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 (*low bias*), Not too complex (*low variance*)

⌚ Final Goal:

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

Model 1	Model 2	Model 3
Training Acc = 90%	Training Acc = 92%	Training Acc = 90%
Test Acc = 80%	Test Acc = 91%	Test Acc = 65%
↓	↓	↓
Overfitting	Generalized Model	Underfitting

⌚ 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

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

Regularization Term

RIDGE REGRESSION; L_2 Regularization

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

- ✓ · Overfitting
- ✓ · multicollinearity
- ✗ · Feature Selection (keeps all features)

shrinkage coefficients

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

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

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
e.g., pass/fail

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

Scenarios If $h_0(x) < 0.5 = 0$ Fail
 $h_0(x) \geq 0.5 = 1$ 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)

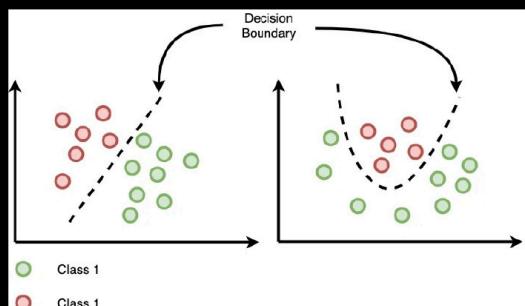
⇒ Logistic Regression uses **Sigmoid** function to handle classification problems

Imp **Binary classification** = Sigmoid function

Multi class classification = Softmax function

Decision Boundary It is a separating 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

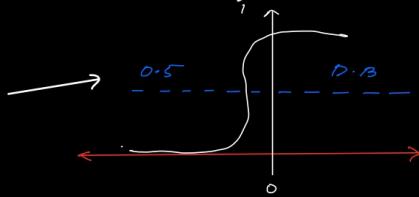
$$f(z) \text{ or } g(z) = \sigma(z) = \frac{1}{1+e^{-z}}$$

$$z = mx + c$$

- 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

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

↓
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)

$$\hat{y} = \frac{1}{1+e^{-(\theta_0x+\theta_1)}} \quad \text{or}$$

$$H_{\theta}(x) = \frac{1}{1+e^{-(\theta_0x+\theta_1)}}$$

GRADIENT DESCENT IN LOGISTIC REGRESSION

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

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

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 \rightarrow 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

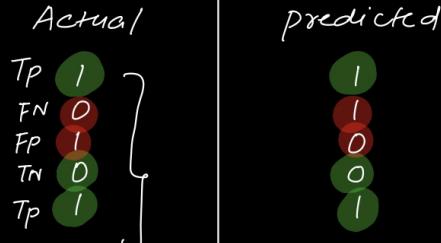
CONFUSION MATRIX V.V. IMP

		PREDICTED	
		1	0
ACTUAL	1	TP (Green)	FP (Red)
	0	FN (Red)	TN (Green)

		PREDICTED	
		1	0
ACTUAL	1	TP	FP
	0	FN	TN

Q11 How does it work?

(5)



		PREDICTED	
		1	0
ACTUAL	1	2	1
	0	1	1

ACCURACY = Ratio of True predictions / Total predictions

$$\frac{TP + TN}{TP + FP + FN + TN} \rightarrow \frac{2+1}{2+1+1+1} = \frac{3}{5} = 0.6 \approx 60\% \text{ Accuracy}$$

Precision = Accuracy of positive predictions

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

Sensitivity

RECALL = True pos 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 ; Cancer Detection
precision 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 \longrightarrow **Balanced**

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

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

$f_{0.5}$ score

$\beta = 2 \longrightarrow f_2$ score \longrightarrow **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 $\longrightarrow \lambda = \frac{1}{C} \longrightarrow$ Logistic

🔍 How it works:

- Large C (e.g., 1000) \rightarrow small regularization \rightarrow model fits more flexibly.
- Small C (e.g., 0.01) \rightarrow strong regularization \rightarrow 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 $K=5$

```
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.

```
# 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

$$\begin{aligned} \text{max_iter} &= 4 \\ C &= 5 \end{aligned} \quad \left\{ \right. \quad 4 \times 5 = 20$$

Imp:

If you pass this to RandomizedSearchCV with $n_{\text{iter}}=5$, 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

```
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-iterations 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

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 ↗ same Scaler

```
# 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',                             # Use accuracy as the scoring metric
    cv=5
)

# 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_))
```

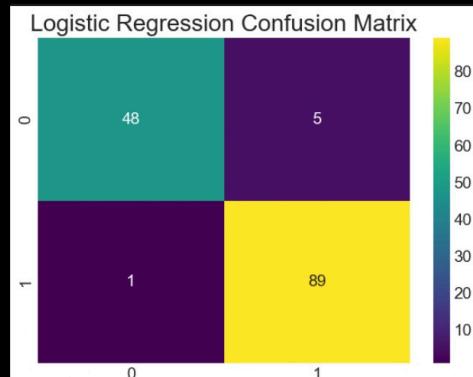
Ridge
by
Default.

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
```



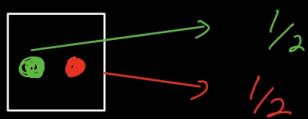
Note; we can check for Metrics individually.

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

$$P(A \text{ and } B) = P(A) \cdot P(B|A)$$

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

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

$$P(A|B) = \frac{P(B) \cdot P(A|B)}{P(A)}$$

Posterior probability $P(A|B)$ is calculated by dividing the Likelihood $P(A|B)$ by the Evidence $P(A)$. The term $P(B)$ is labeled as prior / initial probability.

- $P(A|B)$: 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

$A = Y \Rightarrow$ Target

$B = X \Rightarrow x_1, x_2, x_3 \dots x_n$ 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

$$= P(y) + P(x_1/y) * P(x_2/y) * P(x_3/y) \dots P(x_n/y)$$

$$P(x_1) * P(x_2) * P(x_3) \times \dots \times P(x_n)$$

DAMMET

$$\underline{x_1} \quad \underline{x_2} \quad \underline{x_3} \quad \underline{x_4} \quad \underline{\dots}$$

$$\rightarrow \underline{| \quad \quad \quad |} \quad \text{Yes } \checkmark$$

$$\underline{| \quad \quad \quad |} \quad \text{No } \checkmark$$

Yes

$$\left\{ \begin{array}{l} P(y=\text{Yes}/x_i) = \frac{P(\text{Yes}) * P(x_1/\text{Yes}) * P(x_2/\text{Yes}) * P(x_3/\text{Yes}) * P(x_4/\text{Yes})}{P(x_1) * P(x_2) * P(x_3) * P(x_4)} \\ \text{constant} \quad \rightarrow P(x_1) * P(x_2) * P(x_3) * P(x_4) \quad \# \text{fixed} \end{array} \right.$$

$$\left\{ \begin{array}{l} P(y=\text{No}/x_i) = \frac{P(\text{No}) * P(x_1/\text{No}) * P(x_2/\text{No}) * P(x_3/\text{No}) * P(x_4/\text{No})}{P(x_1) * P(x_2) * P(x_3) * P(x_4)} \\ \text{constant} \quad \rightarrow P(x_1) * P(x_2) * P(x_3) * P(x_4) \quad \# \text{fixed} \end{array} \right.$$

Imp

Let's Assume

$$P(\text{Yes}/x_i) = 0.13 \quad P(\text{No}/x_i) = 0.05$$

Then we will normalize

$$P(\text{Yes}/x_i) = \frac{0.13}{0.13 + 0.05} = 0.72 \quad 72\%$$

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

Q, Lets solve problem cont Naive Bayes

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

$$P(\text{yes} / \text{rain, clear}) = \frac{P(\text{yes}) * P(\text{rain}/\text{yes}) * P(\text{clear}/\text{rain})}{P(\text{rain}) * P(\text{clear})} \text{ Constant}$$

$$= \frac{1/3 * 1/4 * 3/4}{0.0625} = 0.0625$$

$$P(\text{No} / \text{rain, clear}) = 1/3 * 1/4 * 1/4 = 0.02$$

Normalize

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

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

$$P(\text{No} / \text{rain, clear}) = 1 - 0.238 = 0.762$$

$$= 76.2\%$$

Conclusion; we can say, chances of
 No Cricket = 76.2%
 Yes Cricket = 23.8%.

so, No cricket is final prediction

```

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

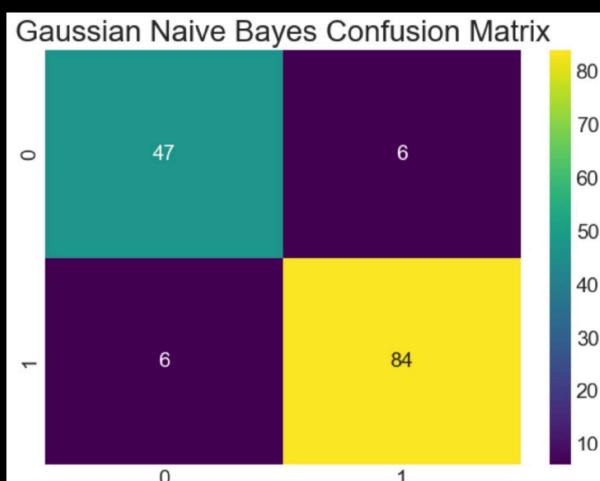
```

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.

Error Rate

Example:

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

- It predicted 16 correctly
- 4 were wrong

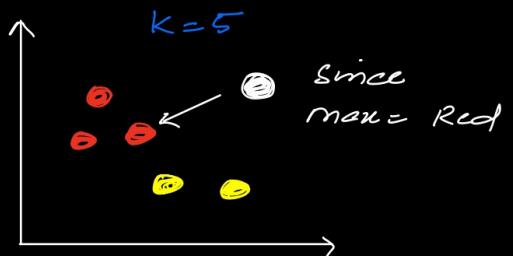
Then:

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

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):

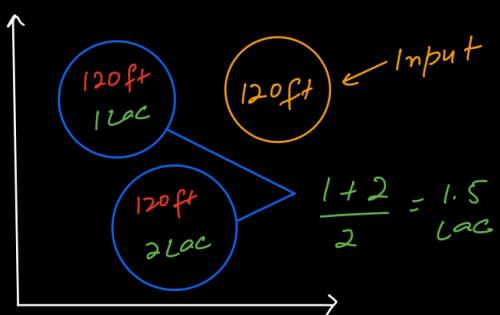
Suppose you want to predict the price of a house based only on its size (in square feet).

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.



In summary:

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

Qn which distances are used?

1. Euclidean Distance

2. Manhattan Distance

Shortest straight line distance between the two points



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

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



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



@Tajamulkhan

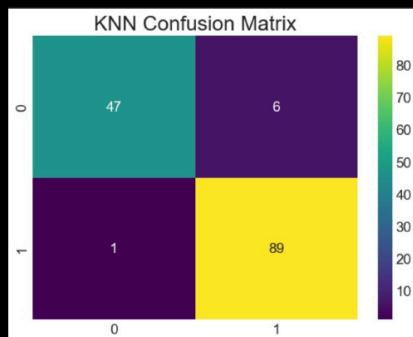
KNN

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 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
```

Q,, Difference b/w Euclidean & Manhattan

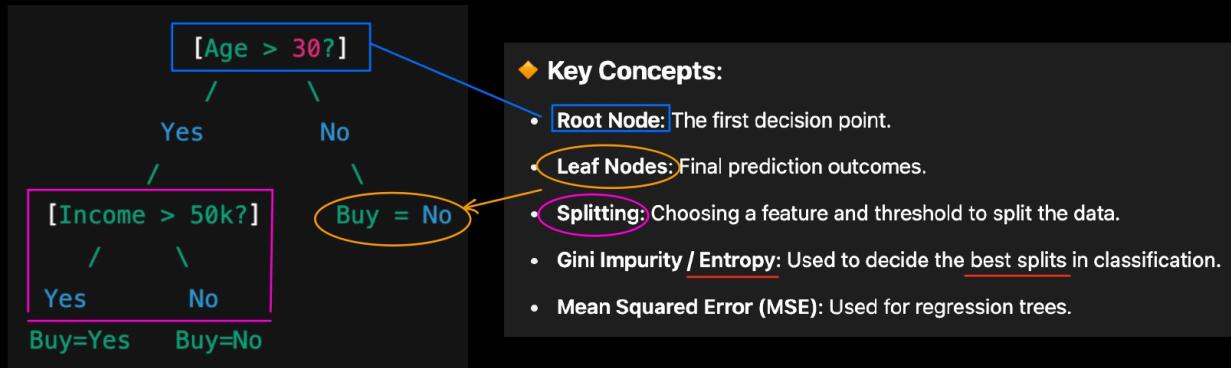
Feature	Manhattan Distance	Euclidean Distance
Path shape	Grid/stepwise	Straight line
Sensitivity to outliers	Less sensitive	More sensitive
Use case	Sparse/high-dimensional data	Low-dimensional, continuous data
Computation cost	Cheaper (no squares/roots)	Slightly costlier (uses square root)

DECISION TREE

is a supervised algorithm used for both classification and regression.

Imp

- It models decisions in the form of a Tree



Advantages:

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

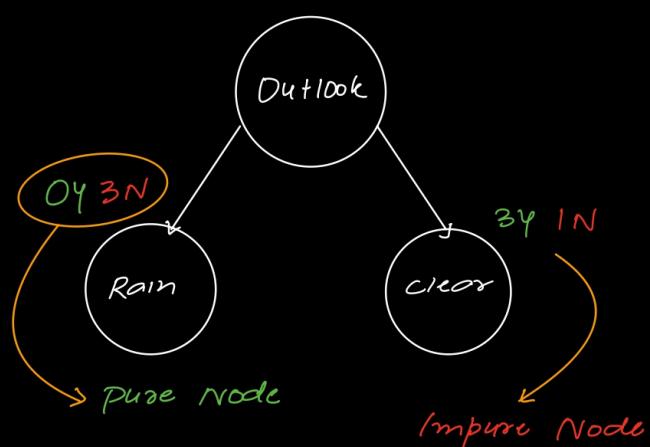
Disadvantages:

- 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

DECISION TREE



- **PURE Node**; If all the Data points belong to the same class
e.g., Rain = OY3N → definitely NO cricket (NO uncertainty)
- **IMPURE Node**; If data points belong to mix of different classes (Uncertainty exists)

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

Q₁ How to measure Impurity?

1. Entropy
2. Gini Index / Impurity

Q₂ How features are selected

Information Gain

purity test

ENTROPY; measure of Impurity in a Node (0 - 1)
- uses log of probabilities Range

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

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

$$\begin{cases} P_+ = \text{Yes} \\ P_- = \text{No} \end{cases}$$

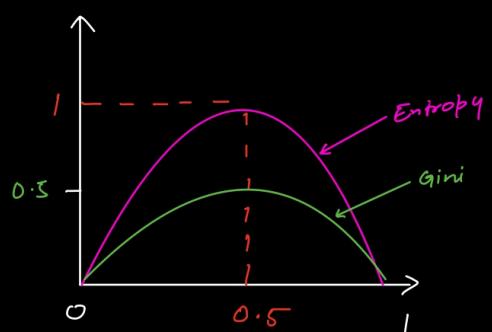
GINI IMPURITY to measure impurity of Nodes

- uses squared probabilities (0 - 0.5) Range

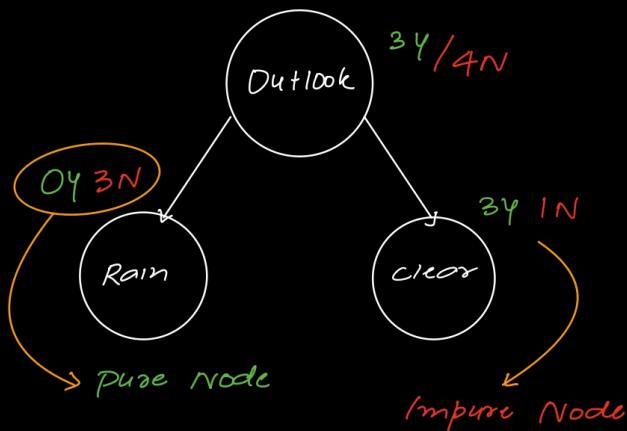
$$\text{Formula} = 1 - \sum_{\pm} (P_{\pm})^2$$

vs 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



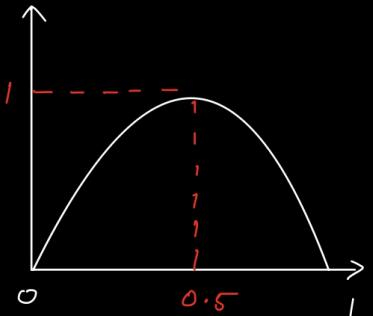
Example for Entropy and Gini Impurity



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

Let's Consider Clear

$$\begin{aligned} \text{Entropy} &= \frac{1}{3} \log_2 \frac{1}{3} - \frac{2}{3} \log_2 \frac{2}{3} \\ H(S) &= 1 \log_2 1 - 0.25 \log_2 0.25 \\ &= 0 - (-0.5) \\ &= 0.5 \rightarrow \text{Impure Split} \end{aligned}$$



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

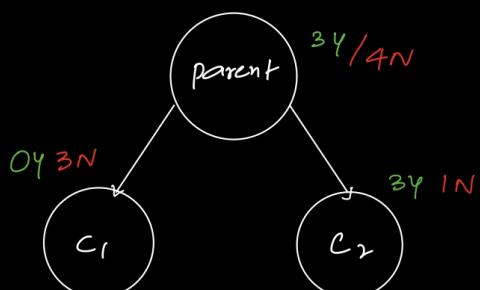
Formula:

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

Steps:

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

→ Let's calculate Information value



1. Entropy (parent)

$$= \frac{3}{7} \log_2 \frac{3}{7} - \frac{4}{7} \log_2 \frac{4}{7} \approx 0.98$$

2. Entropy (child C1)

$$= 0/3 \log_2 0/3 - 3/3 \log_2 3/3 = 0$$

3. Entropy (child C₂)

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

$$4. \text{ Samples} = \text{Total Samples} = 7$$

C1 Samples = 3

$$C_2 \text{ samples} = 4$$

$$\text{Information Gain} = 0.98 - \left[\frac{3}{7} \times 0 + \frac{4}{7} \times 0.011 \right]$$

0.5169

Imp. If gain of feature 2 = 0.490

then I would prefer feature 1 because

$$\text{Gain}(f_1) > \text{Gain}(f_2)$$

Note: The feature with max information gain is given preference.

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

- Entropy uses log which uses a lot of computational power

Preferred; Go for Gini, when Data is complex as it will save a lot 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

✓ 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

more used
Prepruning
Early stopping → max depth, max_leaf
no. of samples

Post Pruning

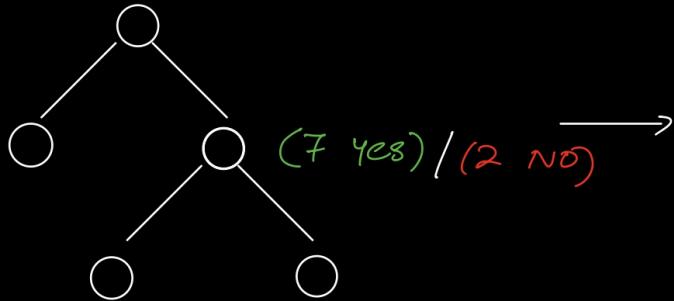
Grow full tree then cut branches

ccp_alpha parameter



What is need of pruning?

So let's say



Here it is obvious
it is Yes, so
we should prune
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

```
# 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))

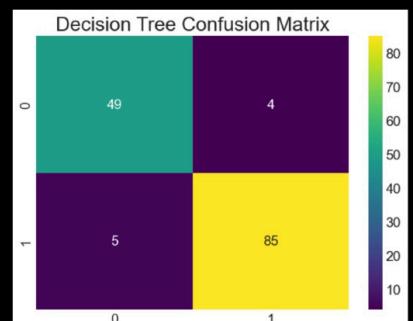
# Extract the best estimator (the pipeline)
best_tree_model = DTC_model.best_estimator_.named_steps['DTC']

# 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

```
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))
```

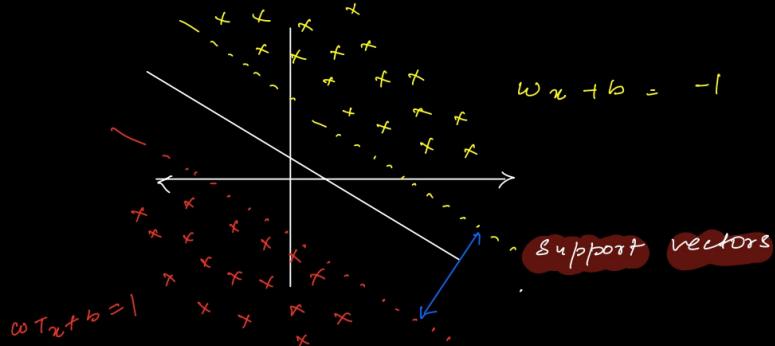
	precision	recall	f1-score	support
0	0.91	0.92	0.92	53
1	0.96	0.94	0.95	90
accuracy			0.94	143
macro avg	0.93	0.93	0.93	143
weighted avg	0.94	0.94	0.94	143



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)



$$\text{Hyperplane} = w^T x + b = 0$$

$$\begin{aligned} w^T x_1 + b &= -1 \\ w^T x_2 + b &= 1 \\ \hline w^T(x_1 - x_2) &= 2 \\ \hline |w| &= \frac{2}{|w|} \end{aligned}$$

Maximize margin

$\frac{1}{2} \|w\|^2$ ← Minimize this

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

Optimization in Real world (soft margin)

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

\sum of distance
Allowed errors 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

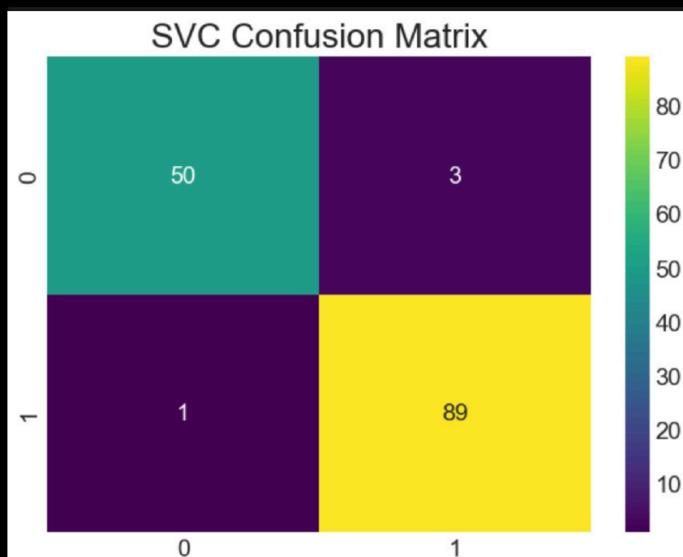
Svc Support Vector Classifier

```
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'],  

```

```
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



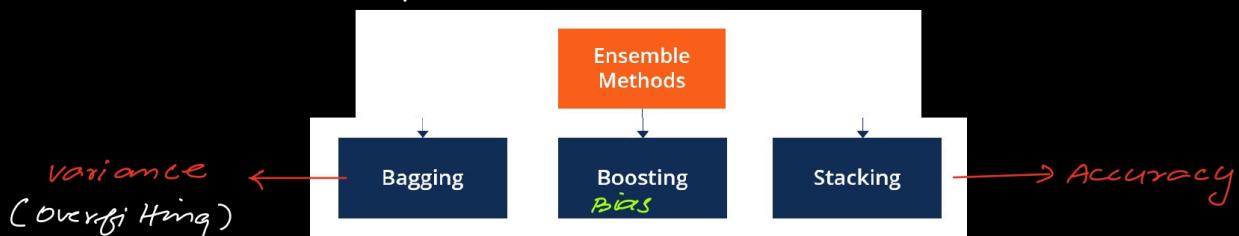
Ensemble Techniques

Combining multiple model together to improve performance.

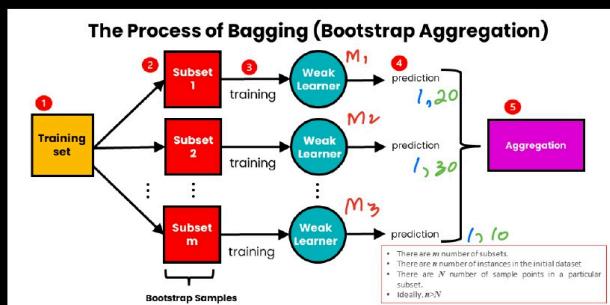
E.g., wisdom of crowd

$DT_1 = \text{No}$, $DT_2 = \text{Yes}$, $DT_3 = \text{No}$

Final outcome = No



Feature	Bagging	Boosting
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 = $\frac{20+30+10}{3} = 20$

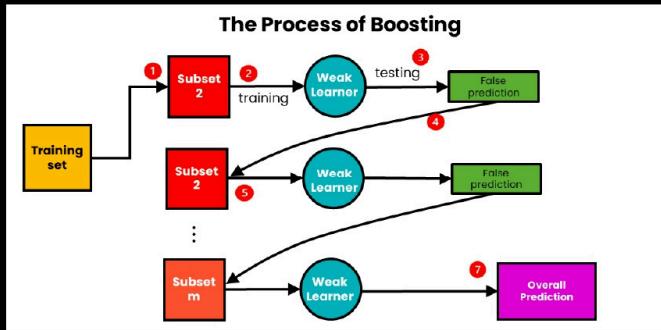
where M_1, M_2, M_3 are different models (Decision Tree)

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.

Bootstrap 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

(with replacement)

Bagging RANDOM FOREST

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.

Q11 Does RF require Norm / standardization?

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

Q11 Random Forest

not impacted
by outliers

KNN / Decision Tree

Impacted by
outliers.

RANDOM FOREST PRACTICALS

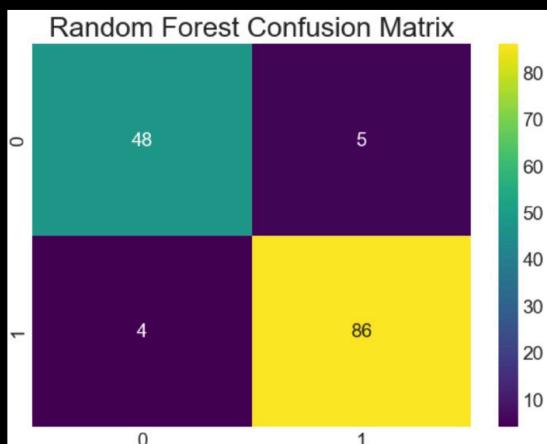
Classifier

```
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  
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("Mean Squared Error (MSE) using Random Forest Regressor: {}".format(random_forest_mse))  
print("Root Mean Squared Error (RMSE) using Random Forest Regressor: {}".format(random_forest_rmse))  
print("R^2 Score using Random Forest Regressor: {}".format(random_forest_r2_score))  
✓ 0.0s  
  
Mean Squared Error (MSE) using Random Forest Regressor: 2947393.7148  
Root Mean Squared Error (RMSE) using Random Forest Regressor: 1716.7975  
R^2 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	Add weights AdaBoost	Gradient Boosting (GBM) Residuals	XGBoost <i>(GBM + Regularization + optimization)</i>
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. ADA BOOST 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	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	$\frac{\text{New } w}{\text{Total } w}$	

STEPS

1. Find Total Error



2. Performance of Stump

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

3. Re calculate weights

$$\begin{aligned} \text{For correct Records} &= \text{weight} \times e^{-P_s} \\ &= 1/2 \times e^{-0.895} = 0.05 \end{aligned}$$

$$\begin{aligned} \text{For Incorrect Records} &= \text{weight} \times e^{+P_s} \\ &1/2 \times e^{+0.895} = 0.349 \end{aligned}$$

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

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

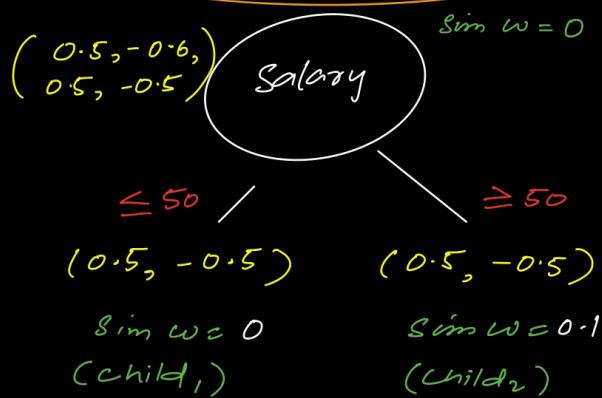
2. Calculate similarity weight

$$\text{Residuals} = \frac{\sum_{\text{Residuals}} (\text{Residual})^2}{\sum_{\text{Residuals}} (\text{pr}(1-\text{pr}) + \lambda)}$$

when $\lambda = 1$

$$\text{Sim } w = \frac{(0.5 - 0.5)^2}{\sum_{\text{Residuals}} 0.5(1-0.5) + \lambda} = 0$$

Imp. **Always Binary DT**



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 ;

$$\leftarrow \begin{array}{c} \text{Base Model} \\ \downarrow \\ \text{Sigmoid} \end{array} \rightarrow \left[0 + \alpha_1 (\text{DT}_1) + \alpha_n \alpha_2 (\text{DT}_n) \right] \downarrow \text{Learning Rate}$$

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.
=

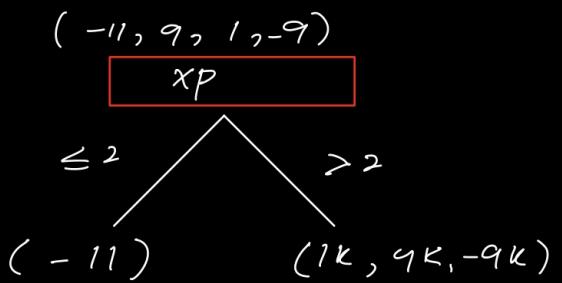
"In XGBoost for classification, instead of using impurity like Gini or entropy, it builds binary decision trees using gradients and second-order values (called Hessians). Each new tree focuses on fixing the mistakes made by the previous one. At every split, XGBoost checks how similar the grouped data 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

Base model = \$1K

X_P	Gap	Salary	R_i
2	Yes	40K	-11K
2.5	Yes	42K	-9K
3	No	52K	1K
3.5	Yes	60K	9K



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

when $\lambda = 1$

$$\delta w_1 = \frac{121}{1+1} = 60.5$$

$$\delta w_2 = \frac{121}{3+1} = 30.25$$

$$\delta w_{\text{parent}} = -\frac{10}{4} = -2.5$$

$$\begin{aligned} \text{Information Gain} &= 60.5 + 30.25 - 2.5 \\ &= 88.25 \end{aligned}$$

$$\begin{aligned} \text{Finally} &= \text{Base pred} + \alpha_1 (DT_1) \quad (\text{mean}) \\ &= 51 + \alpha_1 (-\frac{11}{1}) + \alpha_2 (\frac{1-9+9}{3}) \end{aligned}$$

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: $g = (\hat{y} - y)$
 - **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:

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

- 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:

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

- $f(\mathbf{x})$ 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**.



@Tajamulkhan

BOOSTING PRACTICALS

XG BOOST CLASSIFIER

```
# Optional as it consumes a lot of time**\n\nxgb_pipe = Pipeline(steps=[\n    ('scaler', StandardScaler()),\n    #('pca', PCA()),\n    ('xgb', XGBClassifier())\n])\nparam_grid = {\n    #'pca_n_components': np.arange(1, X_train.shape[1]//3),\n    'xgb_n_estimators': [100],\n    'xgb_learning_rate': np.logspace(-3, 0, 10),\n    'xgb_max_depth': np.arange(1, 6),\n    'xgb_gamma': np.arange(0, 1.0, 0.1),\n    'xgb_reg_lambda': np.logspace(-3, 3, 10)\n}\nxgb_model = GridSearchCV(xgb_pipe, param_grid=param_grid, verbose=1, n_jobs=-1)\nxgb_model.fit(X_train, y_train)\nprint('Best params: {}'.format(xgb_model.best_params_))\nprint('Training Score: {}'.format(xgb_model.score(X_train, y_train)))\nprint('CV Score: {}'.format(xgb_model.best_score_))\nprint('Test Score: {}'.format(xgb_model.score(X_test, y_test)))\n\n✓ 1m 48.0s\n\nFitting 5 folds for each of 5000 candidates, totalling 25000 fits\nBest params: {'xgb_gamma': np.float64(0.3000000000000004), 'xgb_learning_rate': np.float64(0.46415888336127775)}\nTraining Score: 0.9976525821596244\nCV Score: 0.9695212038303694\nTest Score: 0.951048951048951
```

XG BOOST REGRESSOR

```
xgb = xgb.XGBRegressor()\nxgb.fit(X_train, y_train)\n\n# To get predictions\ny_pred5 = xgb.predict(X_test)\n\n✓ 0.2s\n\n\nfrom sklearn.metrics import mean_squared_error, r2_score\nimport numpy as np\n\n# Evaluation Metrics\nxgb_reg_mse = mean_squared_error(y_test, y_pred5)\nxgb_reg_rmse = np.sqrt(xgb_reg_mse)\nxgb_reg_r2_score = r2_score(y_test, y_pred5)\n\nprint(f"Mean Squared Error (MSE) using XGBoost Regressor: {xgb_reg_mse:.4f}")\nprint(f"Root Mean Squared Error (RMSE) using XGBoost Regressor: {xgb_reg_rmse:.4f}")\nprint(f"R2 Score using XGBoost Regressor: {xgb_reg_r2_score:.4f}")\n\n✓ 0.0s\n\nMean Squared Error (MSE) using XGBoost Regressor: 4046267.4282\nRoot Mean Squared Error (RMSE) using XGBoost Regressor: 2011.5336\nR2 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

python

Regressor

```
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

```
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

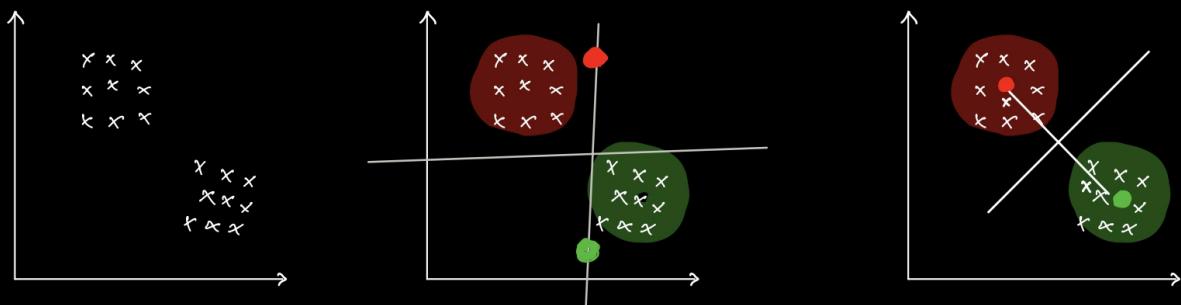
2. Dimensionality Reduction

- PCA

centroids K MEANS clustering

is an unsupervised algorithm used to group similar data points.

$K \rightarrow$ no. of clusters (with centroid)

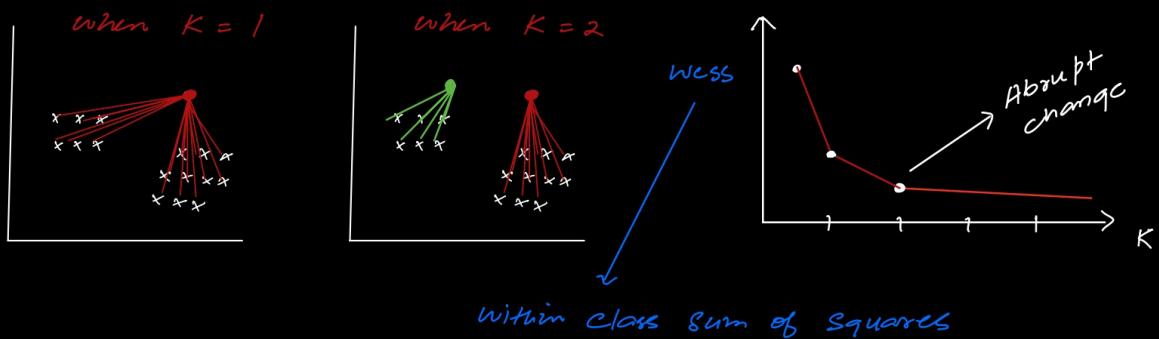


STEPS

- I we take 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

Q4 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

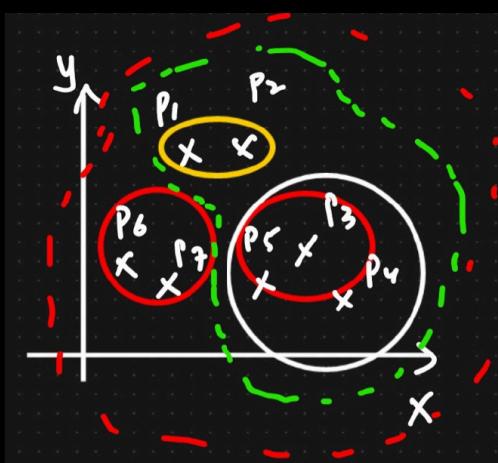
Approaches

1. Agglomerative

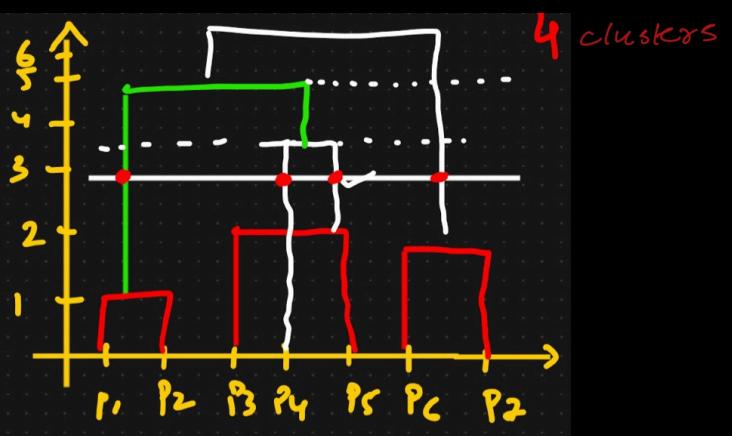
Bottom to top

2. Divisive

Top - Bottom



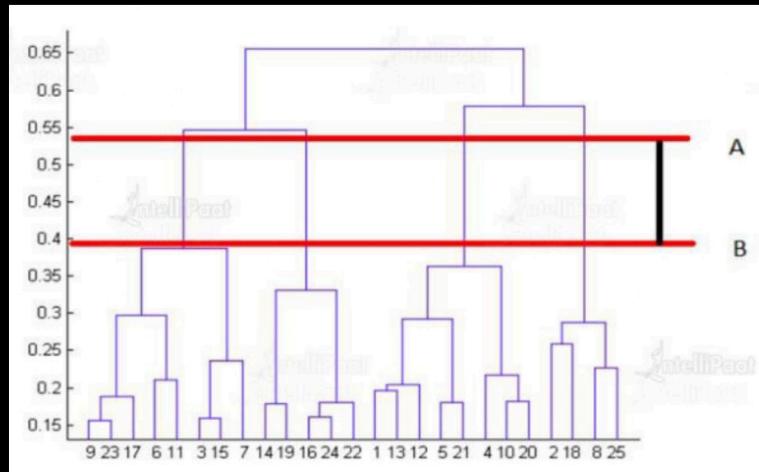
(Euclidean Distance)



(Dendrogram) ↗

Q11 How to find optimal number of Groups?

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



Q11 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)

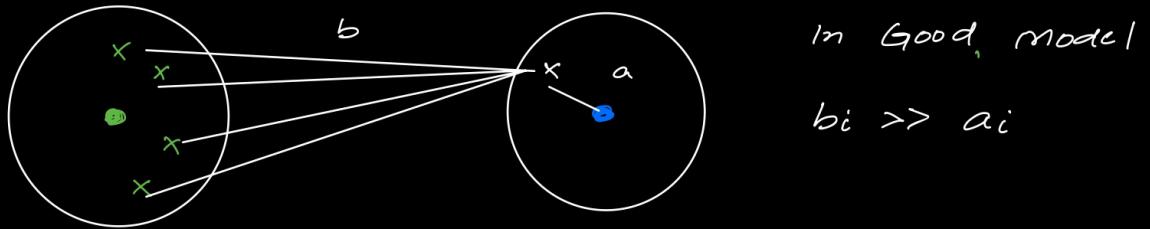
Validation of Clustering

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

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

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"



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

which means $+1 = \text{Good model}$
 $-1 = \text{Bad model}$

Qn what is K means ++ ?

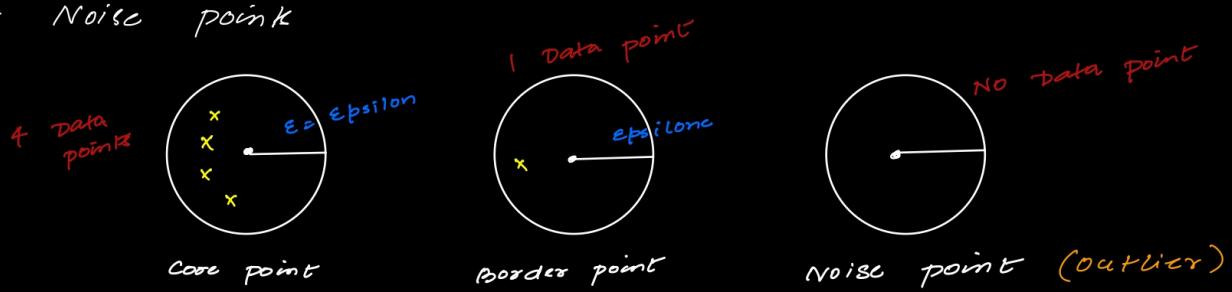
It makes sure centroids are initialise away from data points

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.

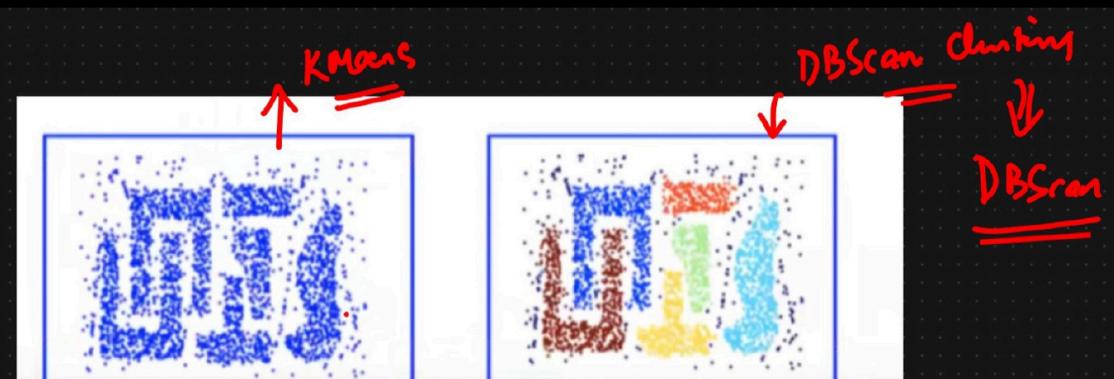
1. Min points \longrightarrow Minpts = 4 (Hyperparameter)
2. Core points
3. Border points
4. Noise points



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 preferred.

↳ use Always.

UNSUPERVISED PRACTICALS

K MEANS PRACTICALS

```
# 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_)
```

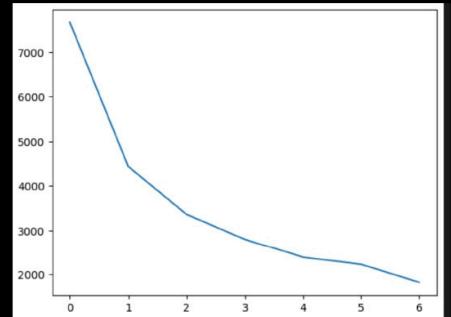
ELBOW METHOD

```
# 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

```
# 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:

    # initialise 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))

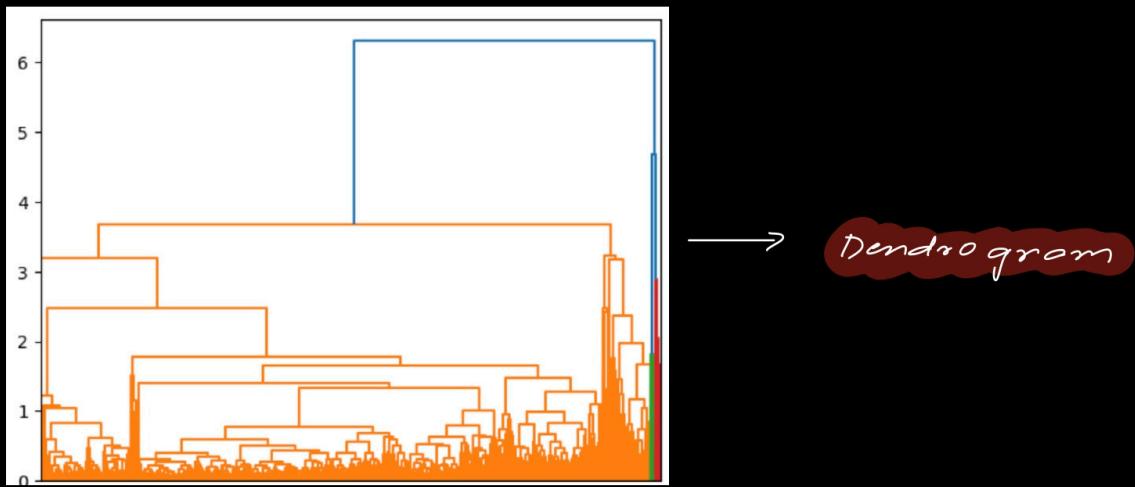
    ✓ 1.2s

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 0.4146742211900517
For n_clusters=8, the silhouette score is 0.40130271413920454
```

HIERARCHICAL CLUSTERING

```
# Average linkage

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



DBSCAN

```

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

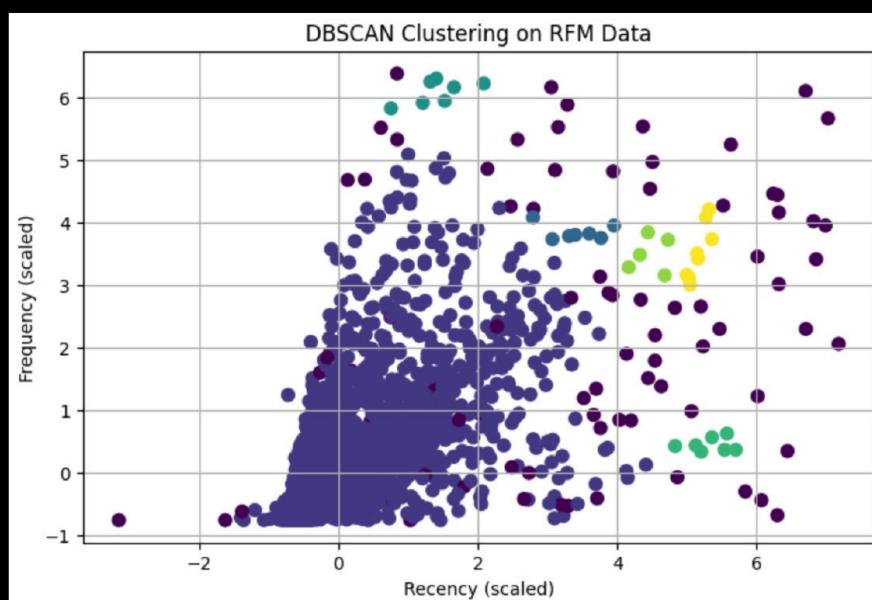
# 1. Apply DBSCAN clustering
dbSCAN = DBSCAN(eps=0.5, min_samples=5) # ✓ adjust eps if needed based on k-distance plot
dbSCAN_labels = dbSCAN.fit_predict(rfm_df_scaled)

# 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

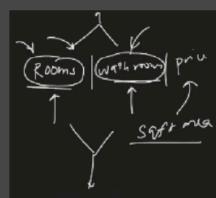
refers to techniques for reduction of number of input variables in training data

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

INDEPENDENT COMPONENT ANALYSIS

T-DISTRIBUTED STOCHASTIC NEIGHBOR EMBEDDING

Non Linear Combination

KERNEL PCA

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



Overfitting Synonym

CURSE OF DIMENSIONALITY

The Concept of COD states

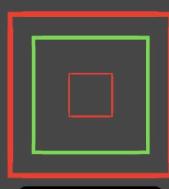
The Model Performance Tends to increase only until a certain number of features are added

Disadvantages

Performance Decrease
Complex Computation

- When the number of features is more
 - The data matrices become **sparser**
 - The distance between the samples increase tremendously
 - Making predictions in the **higher** dimensions becomes difficult than making predictions in lower dimensions with the same number of samples
- One possible solution
 - Have more training instances to have sufficient density for making reliable predictions

4 = Max Performance
5 = No Use of Extra One



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

Memory saving
Speed up training
May filter our noise

Cons

Loss of information
Performance might go down



@Tajamulkhan

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.

Principal Components \leq Number of Features

Advantages
Less Space
Fast Execution
Visualization



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

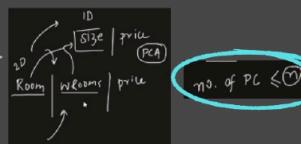
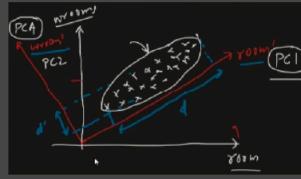
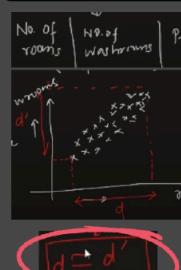
2. Geometric Intuition



Problem 2

Select a feature when $d = d'$

i.e both features are important

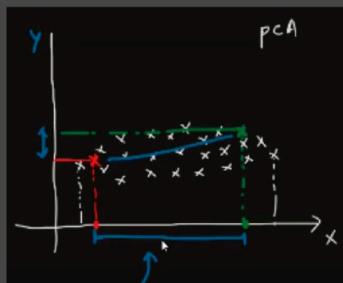


IMPORTANCE OF VARIANCE

Variance \propto 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

Mean Absolute Deviation \rightarrow We take modulus
 \rightarrow And Mod as a function is not differentiable with respect to 0 \rightarrow So, we cannot differentiate
 \rightarrow Whereas square function is differentiable \rightarrow That's why we consider variance \rightarrow Instead of standard deviation.
 \rightarrow Requires Max Variance \rightarrow so that one component collects the most "uniqueness" from the data set.

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

$$D \xrightarrow{?} 3 \xrightarrow{?} 95\%$$

- 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



✓ Step-by-Step Explanation:

1. Start with your dataset

- You have a dataset with many features (like height, weight, income, etc.).

2. Standardize the data

- Make sure all features are on the same scale (mean = 0, std = 1), otherwise large features dominate.

3. Find the directions of maximum variance

- PCA finds new axes (called Principal Components) that capture most of the variation in the data.

4. Compute the covariance matrix

- Measures how features vary together.

5. Calculate eigenvalues and eigenvectors

- Eigenvectors = directions of new axes
- Eigenvalues = how much variance each direction captures

6. Sort and select top components

- Pick the top "k" eigenvectors that capture most variance (say 95%).

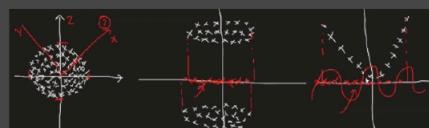
7. Project data onto new space

- Multiply the original data with the selected eigenvectors → this gives the reduced data.

PCA LIMITATIONS

✗ Sensitive to outliers

✗ 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 Model:

```
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_)
#plotting plt.plot(np.cumsum(pca.explained_variance_ratio_))
```

Visualization



LDA

LDA

Multi Class Classification Algorithm

$\leq C-1$

Supervised

Dimensionality Reduction

Linear Discriminant Analysis is a dimensionality reduction technique that is commonly used for supervised classification 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.

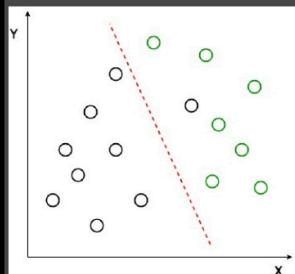


AIM

The 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.

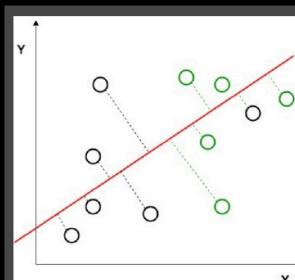
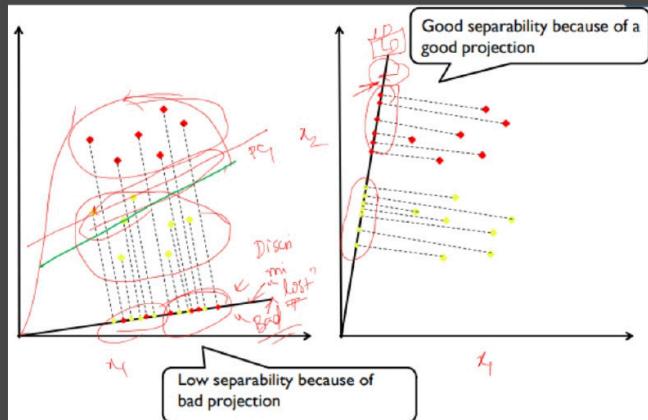


Here, Linear Discriminant Analysis uses both the axes (X and Y) to create a new axis and projects data onto a new axis in a way to maximize the separation of the two categories and hence, reducing the 2D graph into a 1D graph.

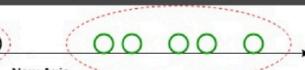
Two criteria are used by LDA to create a new axis:

- 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)

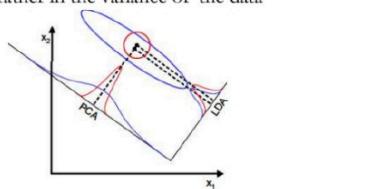
LDA SUMMARY

- 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). S_W S_B S_{WB}
- 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 $N \times k$ dimensional matrix W (where every column represents an eigenvector).
- Use this $N \times k$ 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 $m \times n$ -dimensional matrix representing the m samples, and Z are the transformed $m \times k$ -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:

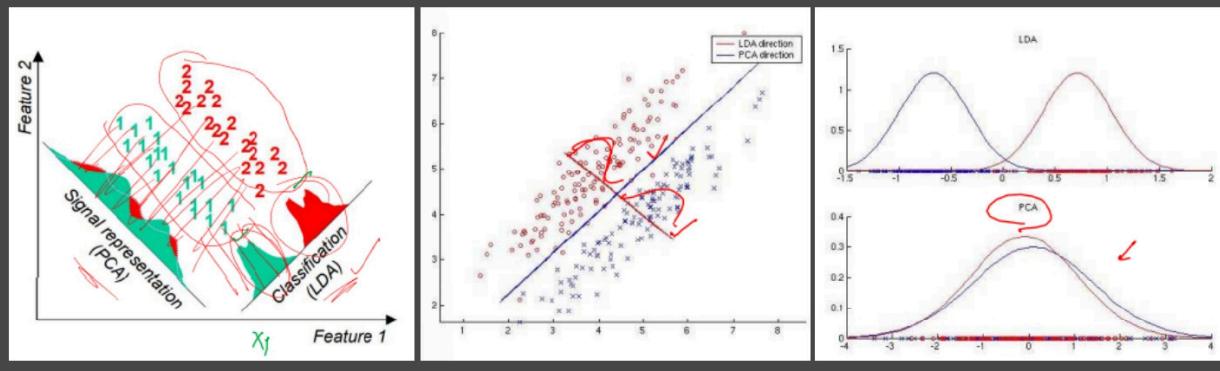
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis  
  
lda = LinearDiscriminantAnalysis(n_components=2)  
X_train = lda.fit_transform(X_train, y_train)  
X_test = lda.transform(X_test)  
  
(< n_classes - 1) for dimensionality reduction.  
  
Visualization  
plt.scatter(X_train[:,0], 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

Comparison of PCA and LDA

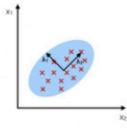
PCA: Perform dimensionality reduction while preserving as much of the Variance in the high dimensional space as possible.

LDA: Perform dimensionality reduction while preserving as much of the class discriminatory information as possible.

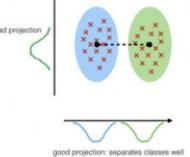
PCA is the standard choice for unsupervised problems (no labels)
LDA exploits class labels to find a subspace so that separates the classes as good as possible

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:
component axes that maximize the variance



LDA:
maximizing the component axes for class-separation



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 (Programmatically)

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

Quick Revision

IMPORTANT POINTS

- MSE = Sensitive to Outliers
MAE = Is Not
- Gradient Descent = SGD, SGR

```
from sklearn.linear_model import SGDClassifier
model = SGDClassifier(loss='log_loss', learning_rate='optimal')
model.fit(X, y)

from sklearn.linear_model import SGDRegressor
model = SGDRegressor(loss='squared_error', learning_rate='optimal')
model.fit(X, y)
```

- Adjusted R^2 is always less than R^2 and it is used to avoid overfitting
- MSE = prone to Outliers \neq MAE
- Lasso is capable of Feature Selection

BIAIS-VARIANCE TRADE OFF

Lower Bias \rightarrow Higher Var
and vice versa

Sol: Lower Bias / Lower Variance (Balance)

- Regularization Terms = penalizes large coefficients

$$L_1 = \lambda \cdot |m|$$

$$L_2 = \lambda \cdot m^2$$

Model Steps.

- Import model
- model.fit(Xtrain, ytrain)
- model.predict(X-test)
- From sklearn.metrics import *
MSE(y-test, y-pred)
r2-score(y-test, y-pred)

- Grid search CV (model, params, scoring, cv)

GS-model.best_params_['alpha']
best_score_

- Binary classification = Sigmoid function
Multi class "r" = Softmax function

- Sigmoid func is used to convert any real numbers in probabilities of 0 & 1

$$\frac{1}{1+e^{-z}} \quad \text{where } e = 2.71 \text{ natural logarithm}$$

$$\text{If } z=0, P=0.5$$

$$z<0, P=0$$

$$z>0, P=1$$



@Tajamulkhan

- Cost Function = $\begin{array}{l} \text{MSE} - \text{Linear} \\ \text{Log Loss} - \text{Logistic} \end{array}$
- $\text{MSE} = \frac{1}{2m} (y_i - \hat{y}_i)^2$
- $\text{Log loss} = \frac{1}{m} (y_i \cdot \log \hat{y}_i + (1-y_i) \cdot \log (1-\hat{y}_i))$ } To achieve Global minima
- Difference in Gradient Descent in Linear Regression & logistic Regression

Linear ; calculate Gradients wrt w & c

$$\frac{\partial J}{\partial w} \quad \& \quad \frac{\partial J}{\partial c}$$

Logistic ; calculate wrt to weights of w

$$\frac{\partial J(\theta)}{\partial (\theta_j)} = \frac{1}{n} (y - \hat{y}) \cdot x \cdot j^{(i)}$$

TP	FP
FN	TN
→ Type 2 Error	

FN is actually TP predicted wrong

$$P = \frac{TP}{TP+FP} \quad ; \quad R = \frac{TP}{TP+FN}$$

- F Score = Harmonic Mean of precision & recall

$$F_{\text{score}} = 2 \times \frac{P \times R}{P + R} \quad . \quad \underline{\text{classification-report}}$$

- L2 Regularization for Logistic Regression

$$\text{Log loss} + \lambda (\theta_j)^2$$

- Cross Validation ; split data in multiple folds to evaluate model's performance

$$k-1 = 5-1 = 4 \quad ; \quad 4 = \text{train}, 1 = \text{test}$$

Random Search CV

chooses randomly

* must specify n_iter

Grid search CV

slow, but all combos specified in hyperparameters

$$\begin{array}{l} a = 4 \text{ params} \\ b = 5 \text{ params} \end{array} \quad ? \quad \left. \begin{array}{l} 4 \times 5 \\ = 20 \text{ total} \end{array} \right.$$

- Categorical Data should be encoded using Label_Encoder etc.

- Scaling should be done after Train - Test split to avoid data leak

$x_{\text{train}} \cdot \text{scale}$ Train Data
 $x_{\text{test}} \cdot \text{scale}$ Test Data } same scalar

- logistic Regularization Term

$$\Rightarrow C = 1/\lambda (\text{inverse})$$

- Confusion matrix = confusion-matrix (y_{test} , y_{pred})

- Naive Bayes** = Based on Bayes theorem which is
Independence of features mostly based on conditional probability

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

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

$$P(A/B) = \frac{P(A) \cdot P(B/A)}{P(B)} \quad \begin{matrix} \text{initial probability} \\ \text{likelihood} \end{matrix} \longrightarrow \text{evidence}$$

In short, it updates probability on the basis of new evidence. (Yes, No)

- Gaussian, Bernoulli, multinomial

- KNN** Distance based ML algo (Classification, Regression)

- If $K=5$ (for class)

= • (majority vote)

- if $K=5$ (Regression)

$$\frac{5}{5} = \frac{5+5+5}{3} = 5 \text{ (mean)}$$

K Neighbour classifier (n-neighbours = 10)

- Distances** = Euclidean \rightarrow shortest $\rightarrow \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$
 Manhattan \rightarrow hypot dist. $\rightarrow |x_1 - x_2| + |y_1 - y_2|$

- Euclidean** \leftarrow Outliers \rightarrow **Manhattan**

- Decision Tree** Models Decisions like Trees
 - prone to outliers

- Impurity = No. of classes in a feature
 the more = more impurity

Pure Node = (Same class)



Imp. In decision Tree our aim is to split nodes in such a way that it increases purity

Key Terms in DT

$$\hookrightarrow \text{Purity Test} \rightarrow \text{Entropy} = -(P_+ \log P_+ + P_- \log P_-)$$

$+ = \text{Yes}$
 $- = \text{No}$

choosing best feature for split

$$\rightarrow \text{Gini} = 1 - \sum_i (P_i)^2$$

$$\text{Information Gain} = P_{\text{Entropy}} - \left[\left(\frac{\text{Samples in C}}{\text{Samples in P}} \right) \times C_{\text{Entropy}} \right]$$

- Entropy \rightarrow More time (\log), 0-1 (small Data)

Gini \rightarrow less time, 0-0.5 (Large Data)

Go for Gini when data is complex as it saves lots of time

- DT Regressor ; Impurity = MSE

$$\text{Info Gain} = P_{\text{MSE}} - C_{\text{MSE}}$$

(Leaf)

\hookrightarrow Mean is taken than Most Common

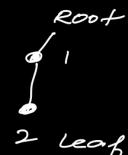
Qs How to avoid overfitting in Trees

use pruning

```

    /   \
  pre   post
  (max-depth)
  hyperparameter
  
```

max-depth = 2



- SVM = Max Margin b/w hyperplane and supporting vectors

$$y = w^T n + b \quad \text{Goal} = \frac{1}{2} \|w\|^2 = \text{minimize}$$

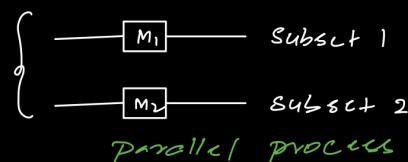
Kernel Trick = In classification, a 2D line can't separate data like  , in this case SVM maps data in higher dimension so data can be separated by a line

- Ensemble ; Combination of multiple models for better performance

Bagging

Majority Vote classifier

Average Regression



Boosting

$M_1 \rightarrow$ Dataset with errors

$M_2 \rightarrow$ Strong learner



@Tajamulkhan

- RF or DT do not require normalization as it is not decision based algo.
- Why XGBoost, naive Bayes don't require
- RF is not impacted by outliers

• BOOSTING; Conversion of weak \rightarrow strong learners

weight + binning • ADA Boost add more weight to wrong predictions so next model focusses more on it

STEPS:

- calculate Errors (E) = $1/\gamma$ 

• calculate performance

$$(\text{performance of stump}) P_s = \frac{1}{2} \log \left(\frac{1-E}{E} \right) = 0.895$$

• Re calculate weights

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

$$\text{incorrect} = \text{weight} \times e^{+P_s}$$

• normalize weights = $\frac{\text{new weight}}{\text{Total weight}}$

• Finally buckets. \rightarrow errors will be in big bucket

• XGBoost; *(Residual)*

- Base model
- calculate Residuals
- calculate Gradients & Hessians

$$\text{Similarity} = \frac{(x - \hat{x}_i)^2}{\text{Pr}(\text{Pr}(1-\text{Pr}) + \lambda)}$$

entropy weight \hookrightarrow used to calculate impurity

$$IG = SW_{\text{child}} - SW_{\text{parent}} \longrightarrow \text{Best feature to split}$$

$$\underbrace{\sigma}_{\text{Sigmoid fx}} \left(0 + \underbrace{\alpha_1(DT)}_{\text{Learning Rate}} + \underbrace{\alpha_2(DT_2)}_{\dots} \dots \right)$$

\hookrightarrow to convert into probabilities

• Unsupervised Learning

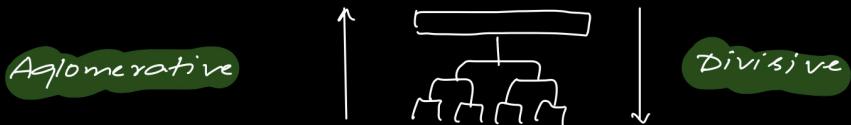
- | | | | |
|------------|----------------|--------------------------|-------|
| clustering | • Kmeans | Dimensionality Reduction | • PCA |
| | • Hierarchical | • T-SNE | |
| | • DBSCAN | Doutliers | |



- K-Means ; $K = \text{no. of clusters}$
 - Initiate K centroids
 - Assign each DP to nearest centroid
(using Euclid distance)
 - update centroid (by taking mean of all DP's)
 - Repeat 3 & 4

• optimal number of clusters = Elbow method

• Hierarchical clustering ; prone to outliers



Kmeans is more preferred

Optimal Number of clusters = The tallest vertical line without any horizontal lines passing thru

Quality of clusters = Silhouette Score

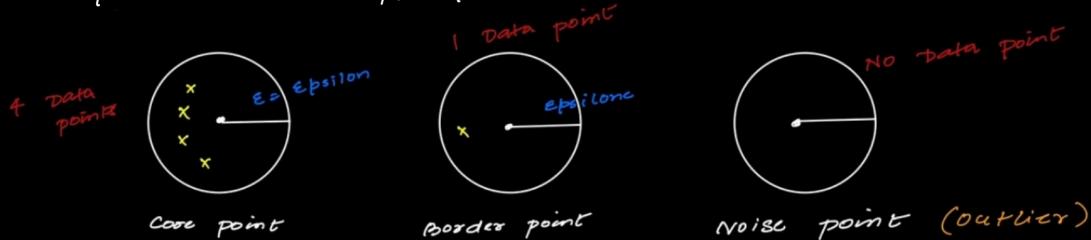
$$S_s = \frac{b - a}{\max(a, b)}$$

Q4 Kmeans \rightarrow centroid, are far from DP's

• DBSCAN ; Density Based Spatial Clustering of Applications with Noise

High Density = cluster low Density = outlier

Min pts = 4 = hyperparameter



DBSCAN is preferred more

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



@Tajamulkhan

PCA

✓ Step 1: Standardize the Data (Mean = 0, Variance = 1)

PCA is sensitive to the scale of the variables.

Standardize each column using:

$$Z = \frac{X - \mu}{\sigma}$$

✓ Step 2: Create the Covariance Matrix

Shows how features vary together (positive or negative correlation). This matrix helps identify directions in which the data varies the most.

$$\text{Cov}(X, Y) = \frac{1}{n-1} \sum (X_i - \bar{X})(Y_i - \bar{Y})$$

✓ Step 3: Compute Eigenvectors and Eigenvalues

Eigenvectors represent new directions (principal components). Eigenvalues show the magnitude of variance in those directions. Think of each eigenvector as an "axis" and its eigenvalue as how important that axis is.

✓ Step 4: Sort Eigenvalues and Select Top k Components

Sort eigenvalues from highest to lowest.

Pick top k eigenvectors corresponding to the largest eigenvalues. This lets you keep the most important information (e.g., 95% of variance).

✓ Step 5: Create the Projection Matrix

Stack the top k eigenvectors to form a matrix:

$$W = [PC_1, PC_2, \dots, PC_k]$$

✓ Step 8: Transform the Original Data

Multiply your standardized dataset by the projection matrix:

$$Z_{\text{reduced}} = X_{\text{standardized}} \cdot W$$

This gives you a new dataset with reduced dimensions.

Use in ML

Faster training

Less overfitting

Better model generalization

**Found
Helpful?
Repost**



Follow for more!

