UNIT-II

MACHINE LEARNING

Dr Soumya Ranjan Mishra

Introduction to Classification

Classification is a supervised learning task where the goal is to **predict the category or class** of an observation based on input features.

Examples:

- Email spam detection (Spam/Not Spam)
- Diagnosing diseases (Positive/Negative)
- Image recognition (Cat/Dog)

Types of Classification

Binary Classification: Two possible classes (e.g., Spam/Not Spam).

Multiclass Classification: More than two classes (e.g., Handwritten digit recognition: 0–9).

Multilabel Classification: An observation can belong to multiple classes simultaneously.

Logistic Regression

Definition: Logistic Regression is a statistical model used for binary classification problems.

Despite its name, it is a classification algorithm, not a regression algorithm.

Why Logistic Regression?

Linear regression isn't suitable for predicting probabilities because it can predict values outside the range of 0 and 1.

Logistic Regression Model

- **Hypothesis**:

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

where:

  - $h_\theta(x)$ is the predicted probability.
  - $\theta^T x$ is the linear combination of weights and features.

- **Sigmoid Function**:

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

  - Compresses the output to a range between 0 and 1.

Decision Boundary

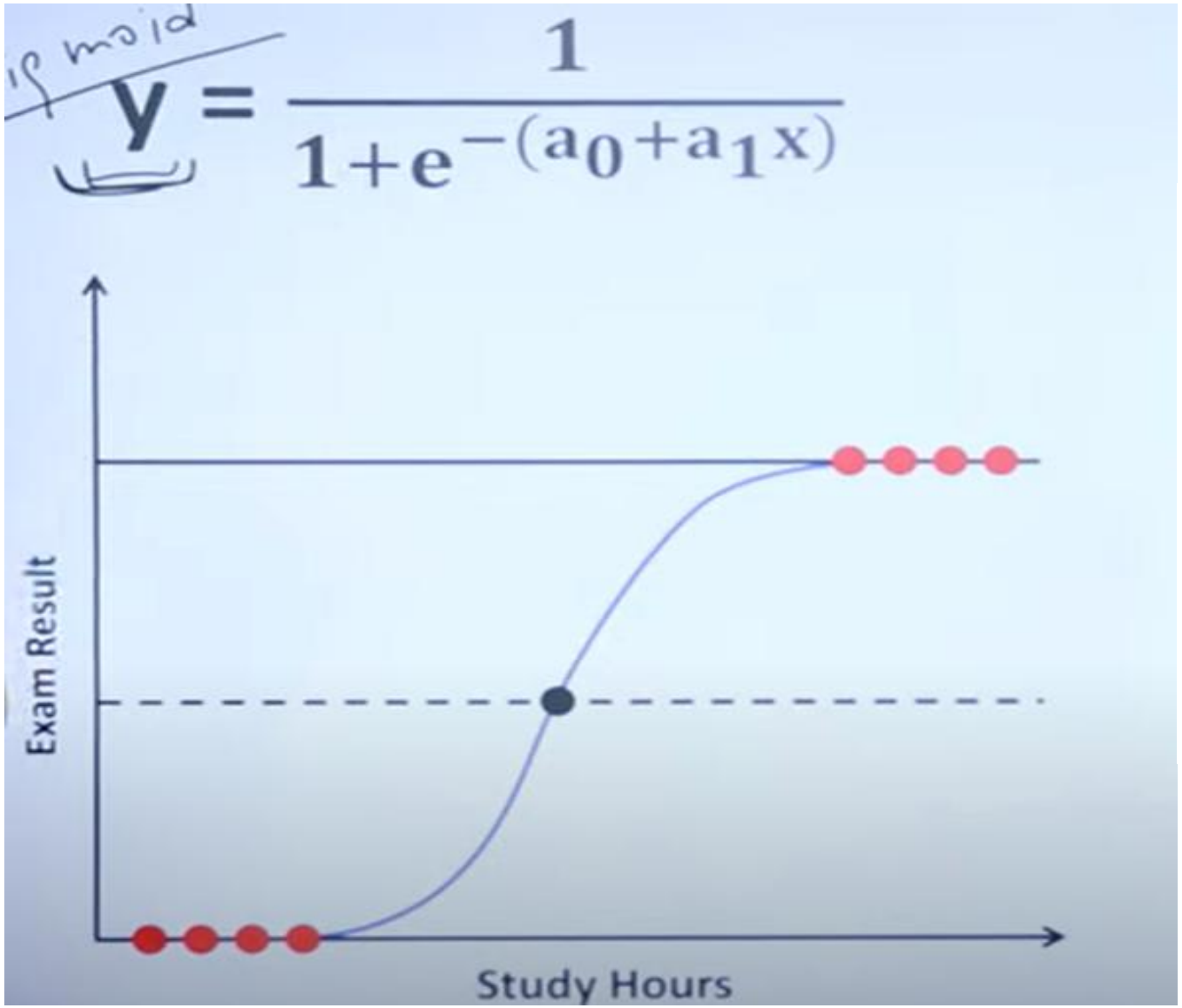
Threshold: A probability threshold (commonly 0.5) is used to classify observations:

- $h_\theta(x) \geq 0.5 \rightarrow$ Class 1
- $h_\theta(x) < 0.5 \rightarrow$ Class 0

Logistic Regression Model

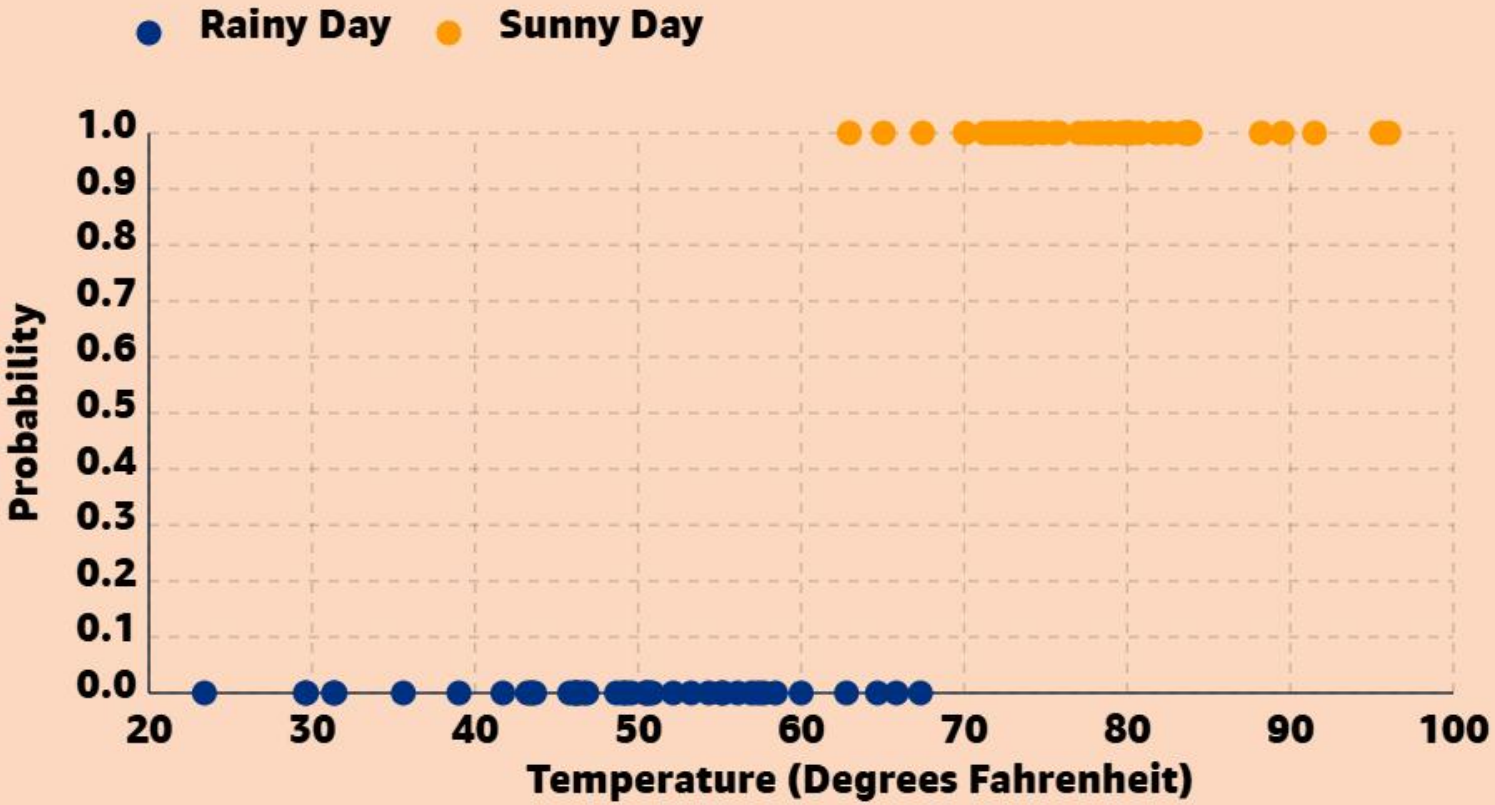
Study Hours	Exam Result
1	0
2	0
3	0
4	0
5	1
6	1
7	1
8	1
9	1
10	1

Logistic Regression Model



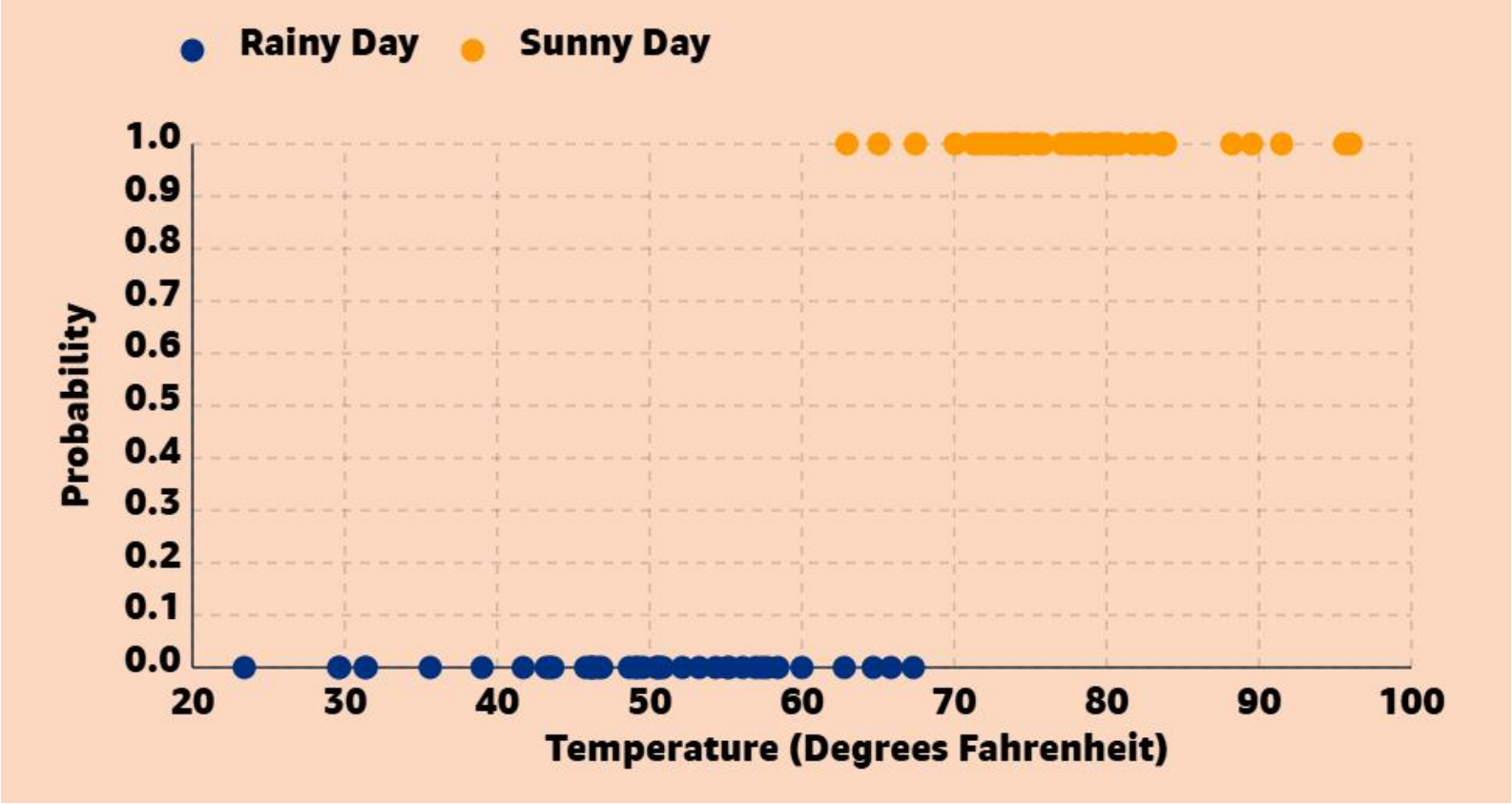
Study Hours	Exam Result
1	0
2	0
3	0
4	0
5	1
6	1
7	1
8	1
9	1
10	1

Logistic Regression Model

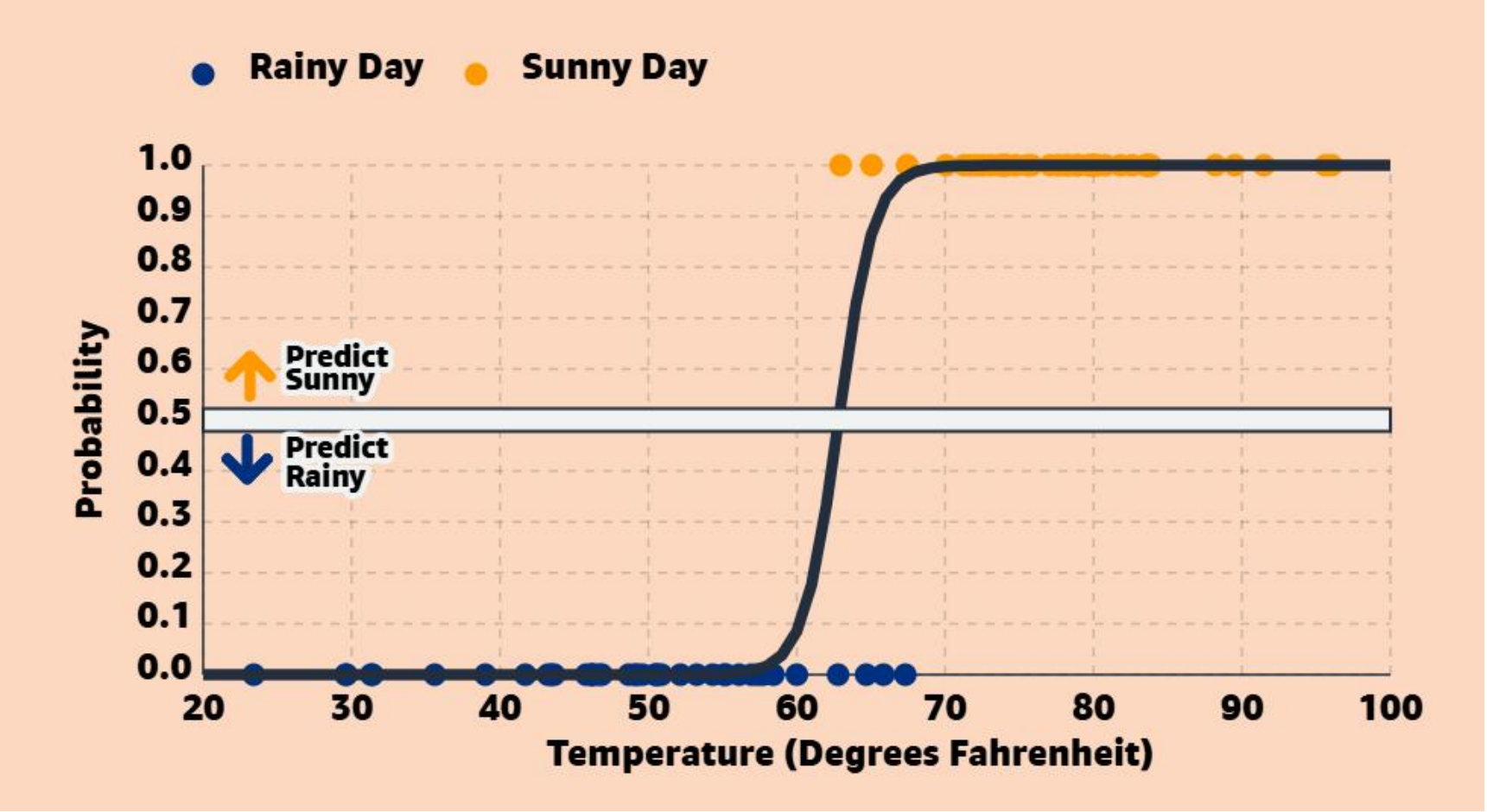


Assume there are two classes: Rainy Days and Sunny Days. We can assign a numeric value of 0 and 1 to each class, say 0 to a Rainy Day and 1 to a Sunny Day. We have one continuous feature: the temperature, in degrees Fahrenheit. For each day, we can plot this value along with the corresponding temperature.

Clearly, we should not fit a **linear regression model to these data**. The outcomes of a linear regression model can take any numerical value, but these data can only take on outcomes of 0 or 1, so the predictions of a linear model may not be meaningful.



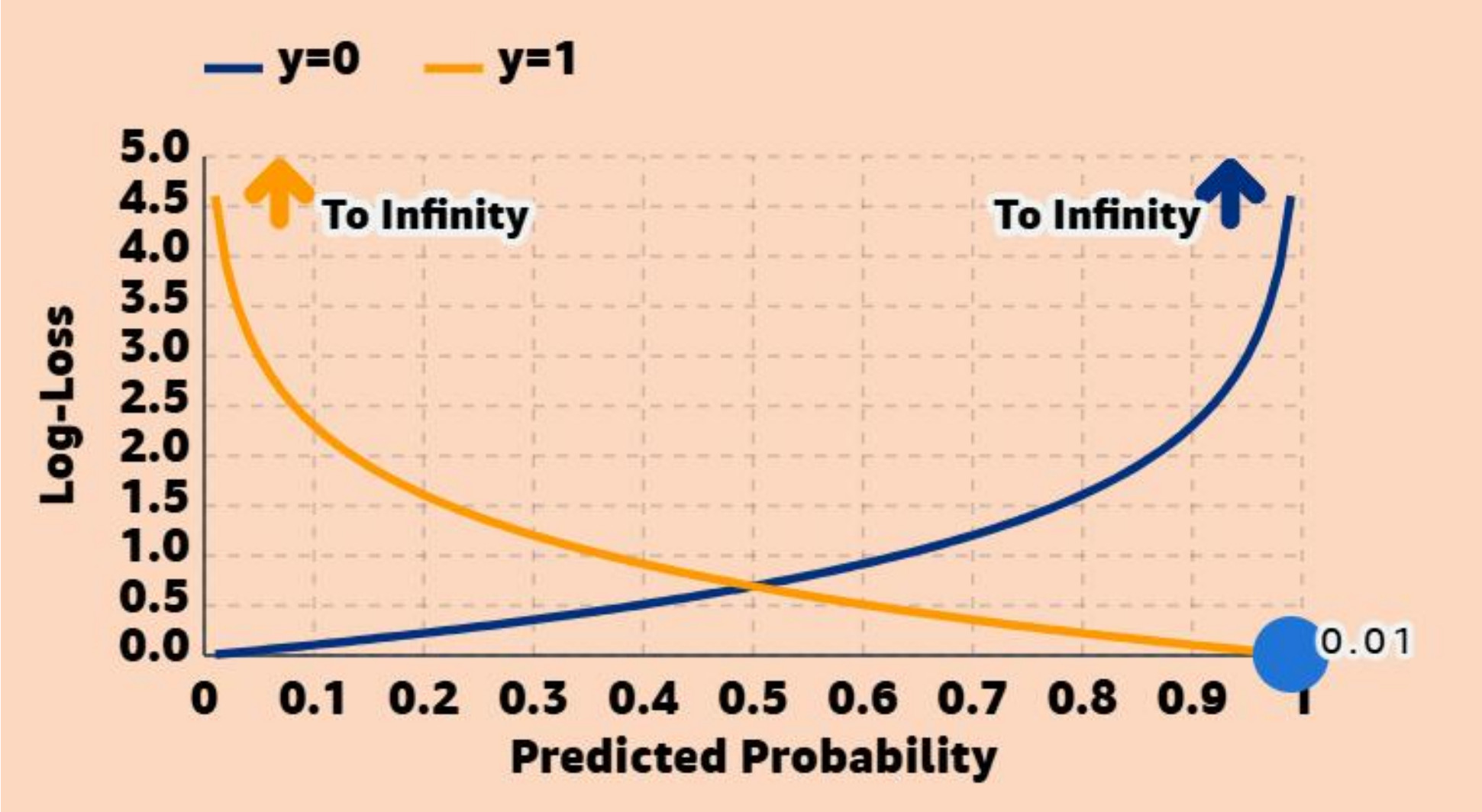
Instead, we can fit a **logistic function** to the data. The values of this function can be interpreted as probabilities, as the values range between 0 and 1. We can interpret the line as the probability of a sunny day given a particular temperature.



What is Log Loss?

- Log Loss (or Logistic Loss) is the **cost function** used in Logistic Regression.
- It measures the performance of a classification model where the prediction is a probability value between 0 and 1.
- The goal is to minimize the log loss to make the model more accurate.

Log Loss


y=0
y=1
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Log-Loss
To Infinity
To Infinity
0.01
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
Predicted Probability

Formula for Log Loss

For binary classification, the log loss for a single data point is:

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

Where:

- y: The actual label (0 or 1).
- $\hat{y}$: The predicted probability of the positive class ($P(y = 1)$).

For m data points in the dataset, the total log loss is the average:

$$\text{Log Loss} = -\frac{1}{m} \sum_{i=1}^{m} \left[y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i)\right]$$

Example

For a binary classification problem:

- True label (y) = 1
- Predicted probability ($\hat{y}$) = 0.9

The log loss is:

$$L = -[1 \cdot \log(0.9) + (1 - 1) \cdot \log(1 - 0.9)] = -\log(0.9) \approx 0.105$$

True Label (y)	Predicted Probability ($\hat{y}$)
1	0.9
0	0.2

Log Loss Calculation:

Using the formula for total log loss:

$$\text{Log Loss} = -\frac{1}{m}\sum_{i=1}^{m}[y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i)]$$

For $m = 2$:

$$\text{Log Loss} = -\frac{1}{2}\left[(1 \cdot \log(0.9) + (1-1) \cdot \log(1-0.9)) + (0 \cdot \log(0.2) + (1-0) \cdot \log(1-0.2))\right]$$

Simplify:

- For the first data point ($y_1 = 1, \hat{y}_1 = 0.9$):

$$L_1 = -\log(0.9) \approx 0.105$$

- For the second data point ($y_2 = 0, \hat{y}_2 = 0.2$):

$$L_2 = -\log(1-0.2) = -\log(0.8) \approx 0.223$$

Average the loss:

$$\text{Log Loss} = \frac{1}{2}(0.105 + 0.223) = 0.164$$

Problem:

A company is using logistic regression to predict whether a candidate will be hired ($y = 1$) or not hired ($y = 0$) based on two features:

1. **Score** (x_1): Test score of the candidate.
2. **Experience** (x_2): Number of years of experience.

The logistic regression model is given by:

$$P(y = 1) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 \cdot x_1 + \beta_2 \cdot x_2)}}$$

The coefficients are:

$$\beta_0 = -3,\ \beta_1 = 0.5,\ \beta_2 = 1.0$$

Candidate	Score (x_1)	Experience (x_2)
1	60	2
2	40	1
3	80	3

The logit is given by:

$$z = \beta_0 + \beta_1 \cdot x_1 + \beta_2 \cdot x_2$$

1. **Candidate 1**:

$$z = -3 + (0.5 \cdot 60) + (1.0 \cdot 2) = -3 + 30 + 2 = 29$$

2. **Candidate 2**:

$$z = -3 + (0.5 \cdot 40) + (1.0 \cdot 1) = -3 + 20 + 1 = 18$$

3. **Candidate 3**:

$$z = -3 + (0.5 \cdot 80) + (1.0 \cdot 3) = -3 + 40 + 3 = 40$$

1. **Candidate 1**:

$$P(y=1) = \frac{1}{1+e^{-29}} \approx 1.0$$

2. **Candidate 2**:

$$P(y=1) = \frac{1}{1+e^{-18}} \approx 1.0$$

3. **Candidate 3**:

$$P(y=1) = \frac{1}{1+e^{-40}} \approx 1.0$$

Final Results:

Candidate	Score (x_1)	Experience (x_2)	z (Logit)	$P(y=1)$	Predicted Outcome
1	60	2	29	1.0	1 (Hired)
2	40	1	18	1.0	1 (Hired)
3	80	3	40	1.0	1 (Hired)

HOME ASSIGNMENT

Problem Statement:

A company wants to predict whether a job candidate will be hired based on their interview score and work experience. The hiring outcome is binary: 1 for hired, 0 for not hired. The logistic regression model has the following equation:

$$P(\text{hired} = 1) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 \cdot \text{Score} + \beta_2 \cdot \text{Experience})}}$$

The model coefficients are:

- $\beta_0 = -4$ (intercept)
- $\beta_1 = 0.8$ (coefficient for Score)
- $\beta_2 = 1.2$ (coefficient for Experience)

Nearest Neighbor Algorithm

- The nearest neighbor algorithm is a simple method used for **classification or regression** tasks.
- It operates by **finding the single closest data point in the feature space** to the test instance.
- **Classification:** Assigns the class of the nearest neighbor to the test instance.
- **Regression:** Assigns the value of the nearest neighbor to the test instance.
- **Applications:** Pattern recognition, image recognition, and simple decision-making tasks.

K-Nearest Neighbor (KNN) Algorithm

KNN is a more generalized version of the nearest neighbor algorithm.

Concept:

- Instead of considering just one closest data point, KNN considers the **k nearest data points** to classify or predict the value of a test instance.

- The value of **k** is chosen based on experimentation and data characteristics.

Steps:

1. Select the number of **neighbors, k**.

2. Calculate the distance (e.g., Euclidean, Manhattan) between the test instance and all training instances.

3. Identify the **k nearest neighbors**.

For Classification:

- Count the frequency of each class among the neighbors.
- Assign the class with the highest frequency.

For Regression:

- Compute the mean (or sometimes median) of the neighbor values.

Distance Metrics:

- Euclidean Distance: $\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$
- Manhattan Distance: $\sum_{i=1}^{n} |x_i - y_i|$

Advantages

- Simple to understand and implement.
- Non-parametric: Makes no assumptions about data distribution.
- Works well for small datasets with fewer dimensions.

Disadvantages

- Computationally expensive for large datasets.
- Sensitive to irrelevant features and feature scaling.
- Performance depends on the choice of k

Example

Classification Example:

Dataset: Animals classified as cats and dogs based on features (weight, height).

Test instance: An unknown animal with certain weight and height.

Steps:

- Compute distances from the test instance to all animals in the dataset.
- Choose k = 3. Identify the 3 nearest neighbors.
- If 2 are cats and 1 is a dog, classify the test instance as a cat.

Regression Example:

Dataset: Predict house prices based on size.

Test instance: A house with a certain size.

Steps:

- Compute distances to all houses in the dataset.
- Choose k = 5. Find the 5 nearest houses.
- Take the average price of these 5 houses as the prediction.

Problem Statement:

You are given a dataset of students with their **Hours of Study** and **Grades (Pass or Fail)**:

Student	Hours of Study	Grade (Pass=1, Fail=0)
A	2	0
B	4	0
C	6	1
D	8	1
E	10	1

Predict the grade of a new student who studies **5 hours** using the **K-Nearest Neighbor Algorithm** with $k = 3$. Use **Euclidean distance** as the distance metric.

1. **Compute the Distance**: Calculate the Euclidean distance between the new student's hours (5) and each student in the dataset:

$$\text{Distance} = |x_{\text{new}} - x_{\text{existing}}|$$

2. **Select the Nearest Neighbors**:
   - Sort the distances: $B = 1, C = 1, A = 3, D = 3, E = 5$.
   - Choose the **3 nearest neighbors**: B, C, A.
3. **Check the Grades**:
   - B → Grade = 0 (Fail)
   - C → Grade = 1 (Pass)
   - A → Grade = 0 (Fail)
4. **Vote**:
   - Fail = 2 votes (B, A)
   - Pass = 1 vote (C)
5. **Assign the Grade**:
   - Majority vote = **Fail (0)**.

The predicted grade for a student studying 5 hours is Fail (0)

CLASS ASSIGNMENT

Problem Statement:

You are given the following dataset, where **X1** and **X2** are features, and **Y** is the class label (0 or 1):

Data Point	X1	X2	Y
A	1	1	0
B	2	2	0
C	3	3	1
D	6	6	1
E	7	7	1

Predict the class for a new data point **P (4, 4)** using the **K-Nearest Neighbor algorithm** with $k = 3$.

Use the **Euclidean distance** formula to calculate distances.

Calculate the Euclidean Distance: The formula for Euclidean distance is:

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

Compute the distance between point $P(4, 4)$ and each data point in the dataset:

Data Point	Coordinates	Distance to $P(4, 4)$
A	(1, 1)	$\sqrt{(4-1)^2 + (4-1)^2} = \sqrt{9+9} = \sqrt{18} \approx 4.24$
B	(2, 2)	$\sqrt{(4-2)^2 + (4-2)^2} = \sqrt{4+4} = \sqrt{8} \approx 2.83$
C	(3, 3)	$\sqrt{(4-3)^2 + (4-3)^2} = \sqrt{1+1} = \sqrt{2} \approx 1.41$
D	(6, 6)	$\sqrt{(4-6)^2 + (4-6)^2} = \sqrt{4+4} = \sqrt{8} \approx 2.83$
E	(7, 7)	$\sqrt{(4-7)^2 + (4-7)^2} = \sqrt{9+9} = \sqrt{18} \approx 4.24$

Sort by Distance: Arrange the distances in ascending order:

Data Point	Distance	Class
C	1.41	1
B	2.83	0
D	2.83	1
A	4.24	0
E	4.24	1

Select the $k = 3$ Nearest Neighbors: The 3 nearest neighbors are:

- C (Distance = 1.41, Class = 1)
- B (Distance = 2.83, Class = 0)
- D (Distance = 2.83, Class = 1)

Majority Voting:

- Among the 3 neighbors:
  - Class 1 appears **2 times** (C, D).
  - Class 0 appears **1 time** (B).
- The majority class is **1**.

Train/Test Split, validation set

In machine learning, data is typically divided into three sets:

- **Training Set:** Used to train the model. The model learns patterns, relationships, and features from this data.
- **Validation Set:** Used to tune model hyperparameters and evaluate its performance during training.
- **Testing Set:** Used to assess the final performance of the model on unseen data to ensure it generalizes well.

Why Split the Data?

- To prevent overfitting or underfitting.
- To ensure the model generalizes well to new, unseen data.
- To optimize the model’s performance by fine-tuning its parameters.

Data Split Proportions

Training Set: Usually 60-80% of the total data.

Validation Set: Typically 10-20%.

Testing Set: Usually 10-20%.

Example: If a dataset has 1000 samples:

Training: 800 samples.

Validation: 100 samples.

Testing: 100 samples.

Workflow

Training Phase: The model learns from the training set. Example: For a house price prediction model, the training set includes features (e.g., square footage, location) and the target variable (price).

Validation Phase: Evaluate the model on the validation set after every training epoch. Adjust hyperparameters like learning rate, batch size, or the number of layers based on validation performance.

Testing Phase: After finalizing the model, evaluate it on the testing set to report metrics like accuracy, precision, recall, or mean squared error (MSE).

Example:

A student learning for exams:

Training Set: Study material (books, notes).

Validation Set: Practice tests to evaluate readiness.

Testing Set: The actual exam.

Output Example

Training set size: 309
Validation set size: 67
Testing set size: 66
Alpha: 0.1, Validation MSE: 2954.8634
Alpha: 1, Validation MSE: 2906.9517
Alpha: 10, Validation MSE: 2920.2729
Alpha: 100, Validation MSE: 3243.5606

Best alpha based on validation set: 1
Test MSE with best alpha: 2800.1235

How Validation Improves Performance

- By using the validation set, we identified the optimal alpha value (1 in this case), which minimized the MSE.
- If we had chosen alpha arbitrarily, the model might not have performed as well on unseen data (test set), potentially leading to overfitting or underfitting.
- This workflow demonstrates the importance of a validation set in fine-tuning hyperparameters to improve the model's generalization capabilities.

Accuracy

Accuracy tells us how often the model makes correct predictions.

Formula:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

Example:

- A model is used to predict whether students will pass or fail.
- Results:
  - **TP (Pass and Correctly Predicted)**: 50
  - **TN (Fail and Correctly Predicted)**: 30
  - **FP (Fail but Predicted Pass)**: 10
  - **FN (Pass but Predicted Fail)**: 10

Accuracy:

$$\text{Accuracy} = \frac{50 + 30}{50 + 30 + 10 + 10} = \frac{80}{100} = 0.8\,(80\%)$$

Interpretation: 80% of the predictions are correct.

Precision

Precision measures the proportion of correctly predicted positive cases.

Formula:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

Example:

- Continuing the student example:
  - **TP**: 50
  - **FP**: 10
- Precision:

$$\text{Precision} = \frac{50}{50 + 10} = \frac{50}{60} = 0.833\,(83.3\%)$$

- **Interpretation**: 83.3% of students predicted to pass actually passed.

Recall (Sensitivity)

Recall measures the proportion of actual positives correctly identified.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Example:

- Using the same data:
  - **TP**: 50
  - **FN**: 10
- Recall:

$$\text{Recall} = \frac{50}{50 + 10} = \frac{50}{60} = 0.833\,(83.3\%)$$

- **Interpretation**: The model correctly identifies 83.3% of students who will pass.

F-Measure (F1 Score)

Combines precision and recall into a single metric using their harmonic mean.

Formula:

$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Example:

- Precision: 83.3% (0.833)
- Recall: 83.3% (0.833)
- F1 Score:

$$F1 = 2 \cdot \frac{0.833 \cdot 0.833}{0.833 + 0.833} = 0.833\,(83.3\%)$$

- **Interpretation**: The F1 Score balances precision and recall, useful for imbalanced data.

ROC Curve (Receiver Operating Characteristic Curve)

A graph showing the trade-off between true positive rate (Recall) and false positive rate (FPR) at different thresholds.

- True Positive Rate (TPR) = Recall
- False Positive Rate (FPR)

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

By adjusting the threshold of predictions, the TPR and FPR change.

Area Under the Curve (AUC):

Measures model performance.

- **AUC = 1: Perfect model.**
- **AUC = 0.5: Random guessing.**

Example

A medical test is used to detect whether a patient has a disease. There are 5 patients, and the test assigns a score to each patient based on the likelihood of disease. The higher the score, the more likely the patient has the disease.

Data:

Patient	Actual Class (Disease)	Predicted Score
A	1 (Positive)	0.9
B	0 (Negative)	0.8
C	1 (Positive)	0.7
D	0 (Negative)	0.4
E	1 (Positive)	0.3

Threshold Selection:

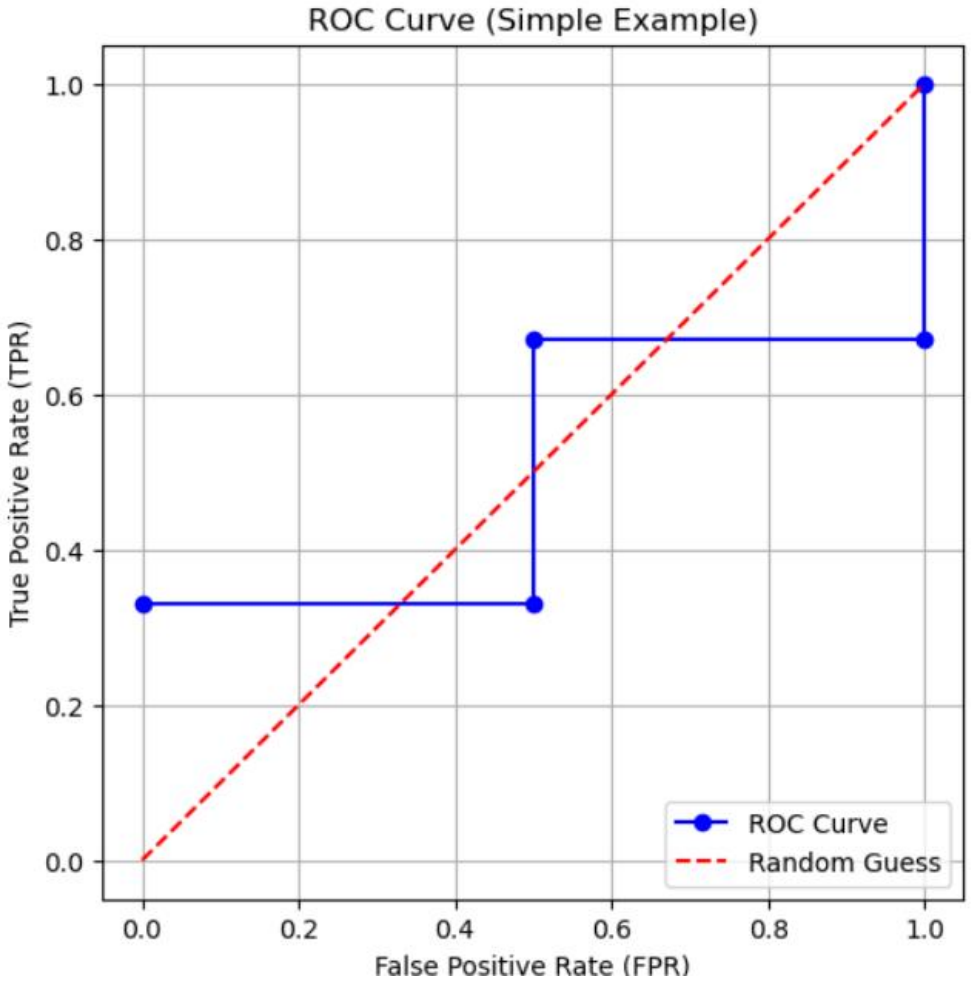
To compute the ROC curve, we vary the threshold to classify scores into positive and negative predictions.

Table of Results:

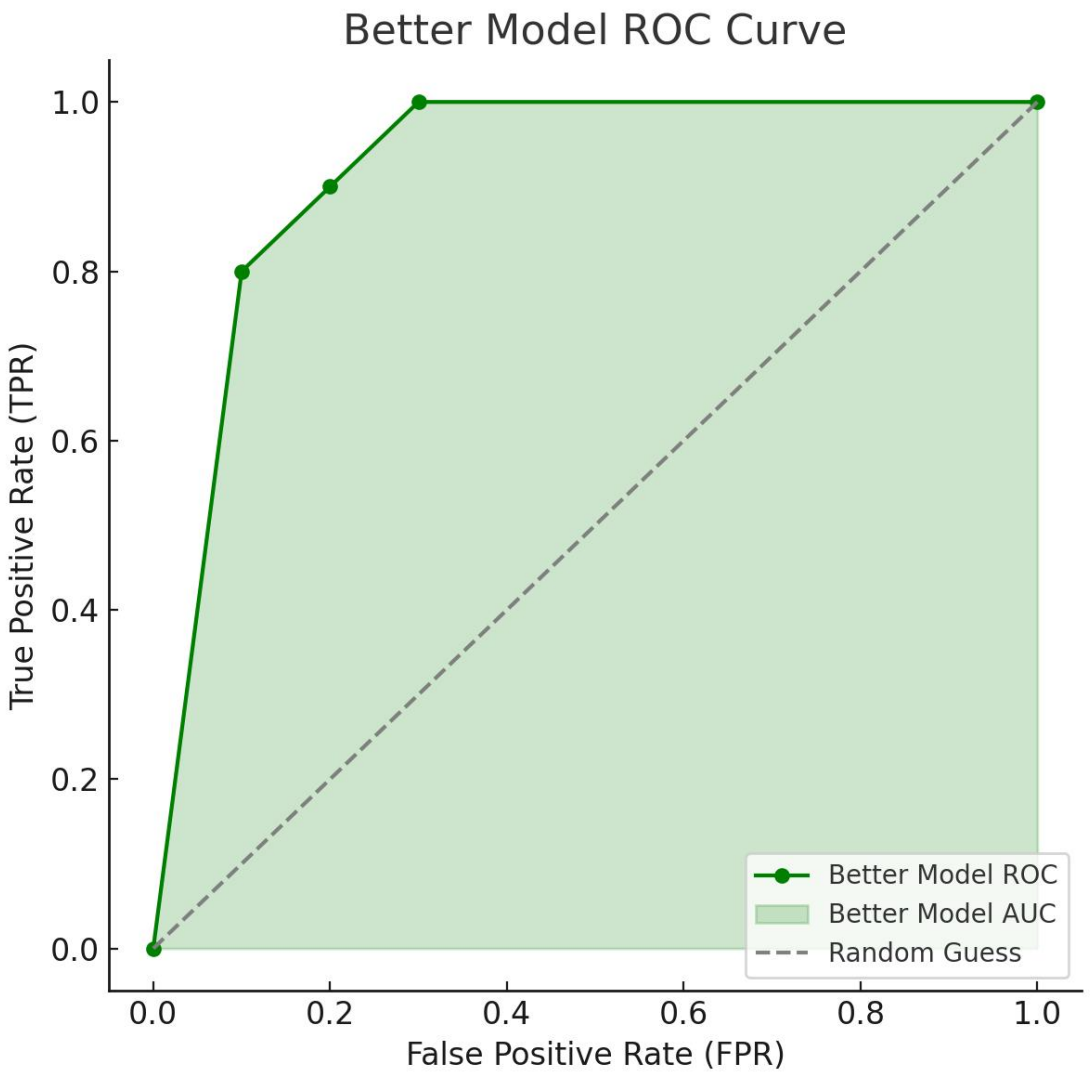
Threshold	TP	FP	TN	FN	TPR ($\frac{TP}{TP+FN}$)	FPR ($\frac{FP}{FP+TN}$)
0.9	1	0	2	2	$\frac{1}{3} = 0.33$	$\frac{0}{2} = 0.00$
0.8	1	1	1	2	$\frac{1}{3} = 0.33$	$\frac{1}{2} = 0.50$
0.7	2	1	1	1	$\frac{2}{3} = 0.67$	$\frac{1}{2} = 0.50$
0.4	2	2	0	1	$\frac{2}{3} = 0.67$	$\frac{2}{2} = 1.00$
0.3	3	2	0	0	$\frac{3}{3} = 1.00$	$\frac{2}{2} = 1.00$

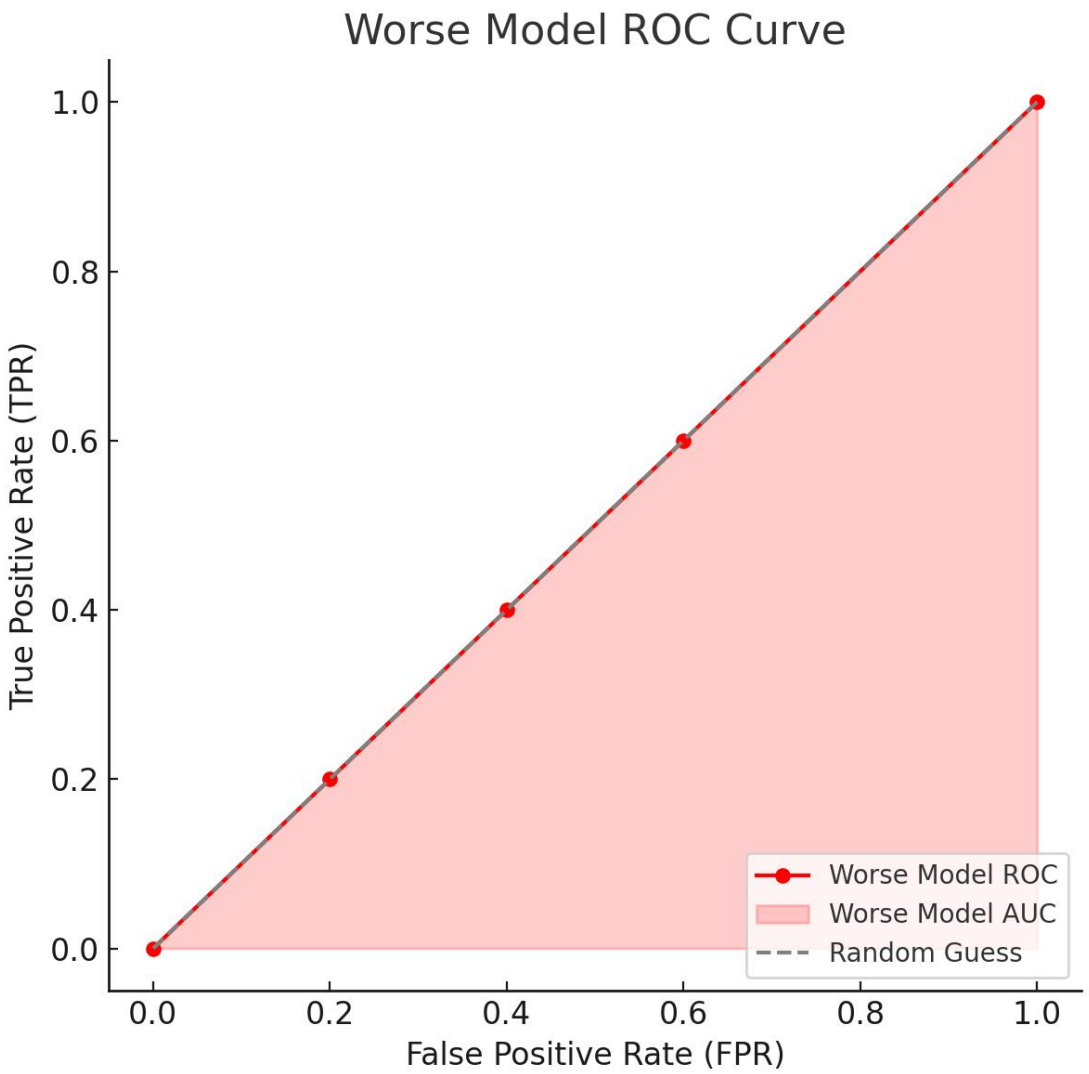
ROC Curve (Receiver Operating Characteristic Curve)

```
import matplotlib.pyplot as plt

fpr = [0.00, 0.50, 0.50, 1.00, 1.00]
tpr = [0.33, 0.33, 0.67, 0.67, 1.00]

plt.figure(figsize=(6, 6))
plt.plot(fpr, tpr, marker='o', color='blue', label='ROC Curve')
plt.plot([0, 1], [0, 1], linestyle='--', color='red', label='Random
Guess')
plt.title("ROC Curve (Simple Example)")
plt.xlabel("False Positive Rate (FPR)")
plt.ylabel("True Positive Rate (TPR)")
plt.legend(loc='lower right')
plt.grid()
plt.show()
```



ROC Curve (Receiver Operating Characteristic Curve)





A binary classifier predicts whether a tumor is malignant (1) or benign (0) based on a threshold applied to its probability scores. The actual labels and predicted probabilities are as follows:

Instance	True Label (Y)	Predicted Probability (P)
1	1	0.9
2	0	0.8
3	1	0.7
4	1	0.4
5	0	0.3
6	1	0.2
7	0	0.1

Threshold	Predictions	TP	FP	TPR	FPR
> 0.9	[1, 0, 0, 0, 0, 0, 0]	1	0	$1/4 = 0.25$	$0/3 = 0.00$
> 0.8	[1, 1, 0, 0, 0, 0, 0]	1	1	$1/4 = 0.25$	$1/3 = 0.33$
> 0.7	[1, 1, 1, 0, 0, 0, 0]	2	1	$2/4 = 0.50$	$1/3 = 0.33$
> 0.4	[1, 1, 1, 1, 0, 0, 0]	3	1	$3/4 = 0.75$	$1/3 = 0.33$
> 0.3	[1, 1, 1, 1, 1, 0, 0]	3	2	$3/4 = 0.75$	$2/3 = 0.67$
> 0.2	[1, 1, 1, 1, 1, 1, 0]	4	2	$4/4 = 1.00$	$2/3 = 0.67$
> 0.1	[1, 1, 1, 1, 1, 1, 1]	4	3	$4/4 = 1.00$	$3/3 = 1.00$

Confusion Matrix

A table summarizing the performance of a classification model.

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Example:

- In a test to detect a disease:
  - **TP**: 50 (Predicted sick, truly sick)
  - **TN**: 30 (Predicted healthy, truly healthy)
  - **FP**: 10 (Predicted sick, truly healthy)
  - **FN**: 10 (Predicted healthy, truly sick)

Confusion Matrix:

	Predicted Sick	Predicted Healthy
Actually Sick	50	10
Actually Healthy	10	30

Naïve Bayes Classifier Algorithm

- **Naïve Bayes algorithm** is a supervised learning algorithm, which is based on Bayes theorem and used for **solving classification problems**.
- It is mainly used in text classification that includes a high-dimensional training dataset.
- Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions.
- **It is a probabilistic classifier, which means it predicts on the basis of the probability of an object**.
- Some popular examples of Naïve Bayes Algorithm are **spam filtration, Sentimental analysis, and classifying articles etc.**

Why is it called Naïve Bayes?

Naïve: It is called Naïve because it assumes that the occurrence of a certain feature is independent of the occurrence of other features.

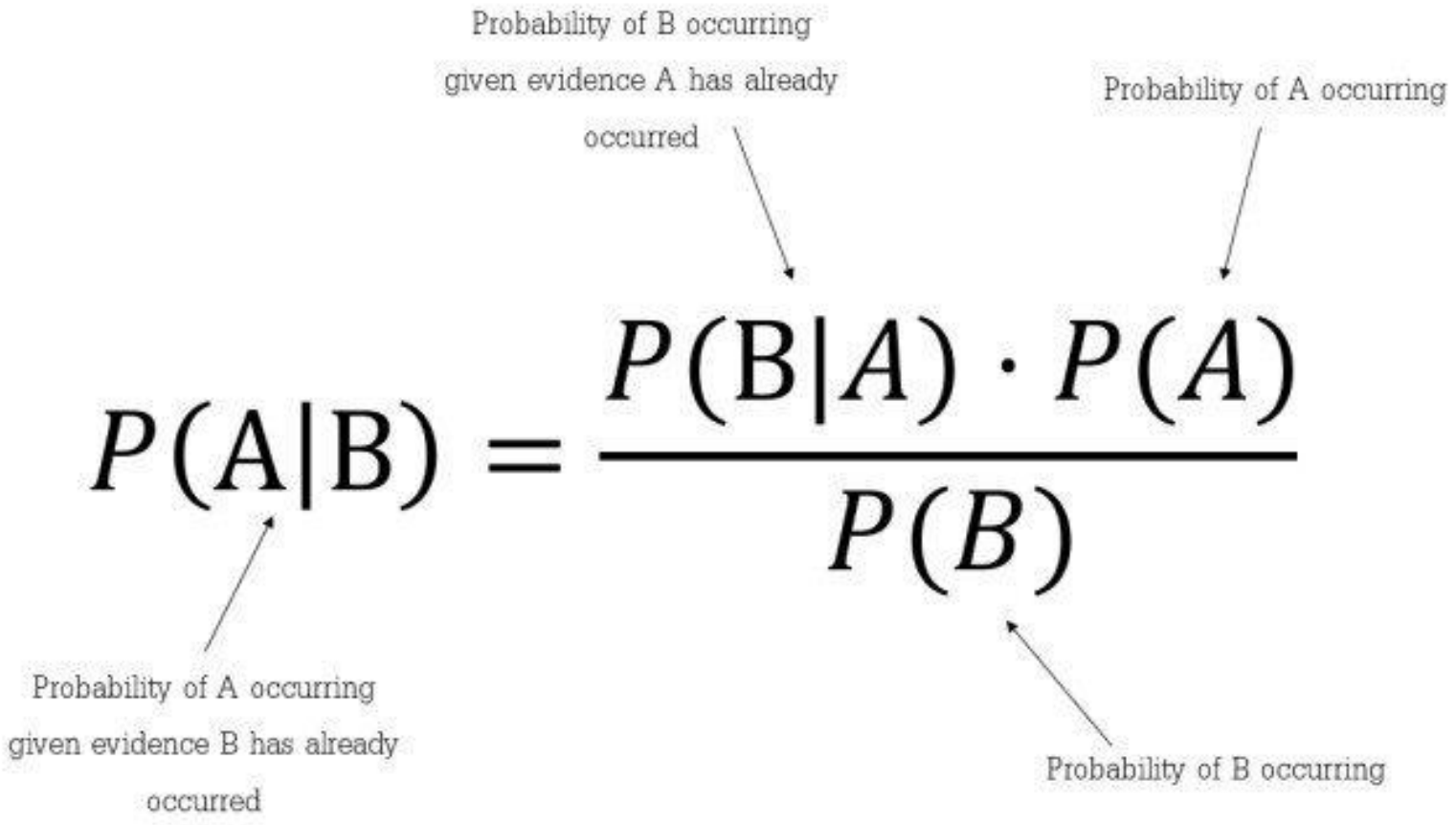
- Such as if the fruit is identified on the bases of **color, shape, and taste**, then red, spherical, and sweet fruit is recognized as an apple.

- Hence **each feature individually contributes** to identify that it is an apple without depending on each other.

Bayes: It is called Bayes because it depends on the principle of Bayes' Theorem.

Bayes Theorem(Conditional probability)

Conditional probability is a measure of the probability of an event occurring given that another event has (by assumption, presumption, assertion, or evidence) occurred.



How often A happens given that B happens

Bayes' Theorem:

Bayes' theorem is also known as Bayes' Rule or Bayes' law, which is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability.

The formula for Bayes' theorem

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

Where,

P(A|B) is **Posterior probability**: **Probability of hypothesis A on the observed event B.**

P(B|A) is **Likelihood probability**: **Probability of the evidence given that the probability of a hypothesis is true.**

P(A) is Prior Probability: Probability of hypothesis before observing the evidence.

P(B) is Marginal Probability: Probability of Evidence.

EXAMPLE- Suppose we have a dataset of weather conditions and corresponding target variable "Play". So using this dataset we need to decide that whether we should play or not on a particular day according to the weather conditions.

	Outlook	Play
0	Rainy	Yes
1	Sunny	Yes
2	Overcast	Yes
3	Overcast	Yes
4	Sunny	No
5	Rainy	Yes
6	Sunny	Yes
7	Overcast	Yes
8	Rainy	No
9	Sunny	No
10	Sunny	Yes
11	Rainy	No
12	Overcast	Yes
13	Overcast	Yes

Problem: If the weather is **sunny,** then the Player should **play or not**?

1. **Convert the given dataset into frequency tables.**
2. **Generate Likelihood table by finding the probabilities of given features.**
3. **Use Bayes theorem to calculate the posterior probability.**

Weather	Yes	No
Overcast	5	0
Rainy	2	2
Sunny	3	2
Total	**10**	**4**

Likelihood table weather condition:

Weather	No	Yes	
Overcast	0	5	5/14= 0.35
Rainy	2	2	4/14=0.29
Sunny	2	3	5/14=0.35
All	**4/14=0.29**	**10/14=0.71**	

Applying Bayes'theorem:

P(Yes|Sunny)= P(Sunny|Yes)*P(Yes)/P(Sunny)

P(Sunny|Yes)= 3/10= 0.3

P(Sunny)= 0.35

P(Yes)=0.71

So P(Yes|Sunny) = 0.3*0.71/0.35= 0.60

P(No|Sunny)= P(Sunny|No)*P(No)/P(Sunny)

P(Sunny|NO)= 2/4=0.5

P(No)= 0.29

P(Sunny)= 0.35

So P(No|Sunny)= 0.5*0.29/0.35 = 0.41

So as we can see from the above calculation that

P(Yes|Sunny) > P(No|Sunny)

Hence on a Sunny day, Player can play the game.

Advantages of Naïve Bayes Classifier:

- Naïve Bayes is one of the fast and easy ML algorithms to predict a class of datasets.
- It can be used for **Binary as well as Multi-class Classifications**.
- It performs well in **Multi-class predictions** as compared to the other Algorithms.
- It is the most popular choice for **text classification** problems.

Disadvantages of Naïve Bayes Classifier:

Naive Bayes assumes that all features are independent or unrelated, so it cannot learn the relationship between features.

Assumptions Made by Naïve Bayes

The fundamental Naïve Bayes assumption is that each feature makes an:

1. **independent**

2. **equal**

contribution to the outcome.

Assumptions Made by Naïve Bayes

Example No.	Color	Type	Origin	Stolen?
1	Red	Sports	Domestic	Yes
2	Red	Sports	Domestic	No
3	Red	Sports	Domestic	Yes
4	Yellow	Sports	Domestic	No
5	Yellow	Sports	Imported	Yes
6	Yellow	SUV	Imported	No
7	Yellow	SUV	Imported	Yes
8	Yellow	SUV	Domestic	No
9	Red	SUV	Imported	No
10	Red	Sports	Imported	Yes

Color	Type	Origin	Stolen
Red	SUV	Domestic	?

- **We assume that no pair of features are dependent. For example, the color being 'Red' has nothing to do with the Type or the Origin of the car. Hence, the features are assumed to be Independent.**

- **Secondly, each feature is given the same influence(or importance). For example, knowing the only Color and Type alone can't predict the outcome perfectly. So none of the attributes are irrelevant and assumed to be contributing Equally to the outcome.**

$$P(y|x_1,\ldots,x_n) = \frac{P(x_1|y)P(x_2|y)\ldots P(x_n|y)P(y)}{P(x_1)P(x_2)\ldots P(x_n)}$$

P(Yes | X) = P(Red | Yes) * P(SUV | Yes) * P(Domestic | Yes) * P(Yes)

= 3/5 * 1/5 * 2/5 * 1

= 0.048

P(No | X) = P(Red | No) * P(SUV | No) * P(Domestic | No) * P(No)

= 2/5 * 3/5 * 3/5 * 1

= 0.144

Since **0.144 > 0.048**, Which means given the features RED SUV and Domestic, our example gets classified as 'NO' the car is not stolen.

Example No.	Contains "Offer"?	Contains "Win"?	Contains "Click"?	Spam?
1	Yes	Yes	No	Yes
2	No	Yes	No	Yes
3	Yes	No	Yes	Yes
4	No	Yes	No	No
5	Yes	Yes	No	No
6	Yes	No	Yes	Yes
7	No	No	No	No
8	Yes	Yes	No	Yes

We want to classify an email with the following features:

Contains "Offer"? = Yes
Contains "Win"? = No
Contains "Click"? = Yes

Step 1: Prior Probabilities

- Total examples = 8
- **Spam examples = 5**
- **Not Spam examples = 3**

$$P(\text{Spam}) = \frac{5}{8} = 0.625$$

$$P(\text{Not Spam}) = \frac{3}{8} = 0.375$$

Step 2: Likelihoods

For **Spam**:

- $P(\text{Offer} = \text{Yes} \mid \text{Spam}) = \frac{\text{Spam examples with Offer} = \text{Yes}}{\text{Total Spam examples}}$
- $P(\text{Win} = \text{No} \mid \text{Spam}) = \frac{\text{Spam examples with Win} = \text{No}}{\text{Total Spam examples}} =$
- $P(\text{Click} = \text{Yes} \mid \text{Spam}) = \frac{\text{Spam examples with Click} = \text{Yes}}{\text{Total Spam examples}}$

For **Not Spam**:

- $P(\text{Offer} = \text{Yes} \mid \text{Not Spam}) = \frac{\text{Not Spam examples with Offer} = \text{Yes}}{\text{Total Not Spam examples}}$
- $P(\text{Win} = \text{No} \mid \text{Not Spam}) = \frac{\text{Not Spam examples with Win} = \text{No}}{\text{Total Not Spam examples}} =$
- $P(\text{Click} = \text{Yes} \mid \text{Not Spam}) = \frac{\text{Not Spam examples with Click} = \text{Yes}}{\text{Total Not Spam examples}}$

Step 3: Posterior Probabilities

For **Spam**:

$$P(\text{Spam} \mid \text{Features}) \propto P(\text{Spam}) \cdot P(\text{Offer} = \text{Yes} \mid \text{Spam}) \cdot P(\text{Win} = \text{No} \mid \text{Spam}) \cdot P(\text{Click} = \text{Yes} \mid \text{Spam})$$

For Not Spam:

$$P(\text{Not Spam} \mid \text{Features}) \propto P(\text{Not Spam}) \cdot P(\text{Offer} = \text{Yes} \mid \text{Not Spam}) \cdot P(\text{Win} = \text{No} \mid \text{Not Spam}) \cdot P(\text{Click} = \text{Yes} \mid$$

Problem Statement:

A company wants to classify emails as Spam or Not Spam based on the occurrence of the words "Free" and "Win". The dataset contains the following information:

SAMPLE PYTHON CODE

```
from sklearn.naive_bayes import GaussianNB

# Sample Dataset

data = [[1, 85], [1, 80], [2, 70], [3, 65], [3, 60]]
labels = [0, 0, 1, 1, 0]

# Create and train model

model = GaussianNB()
model.fit(data, labels)

# Predict for new data

new_data = [[2, 75]]
print(model.predict(new_data))
```

Decision Tree Classification Algorithm

- **Decision Tree** is a Supervised learning technique that can be used for both **classification and Regression problems**, but mostly it is preferred for solving **Classification problems**.

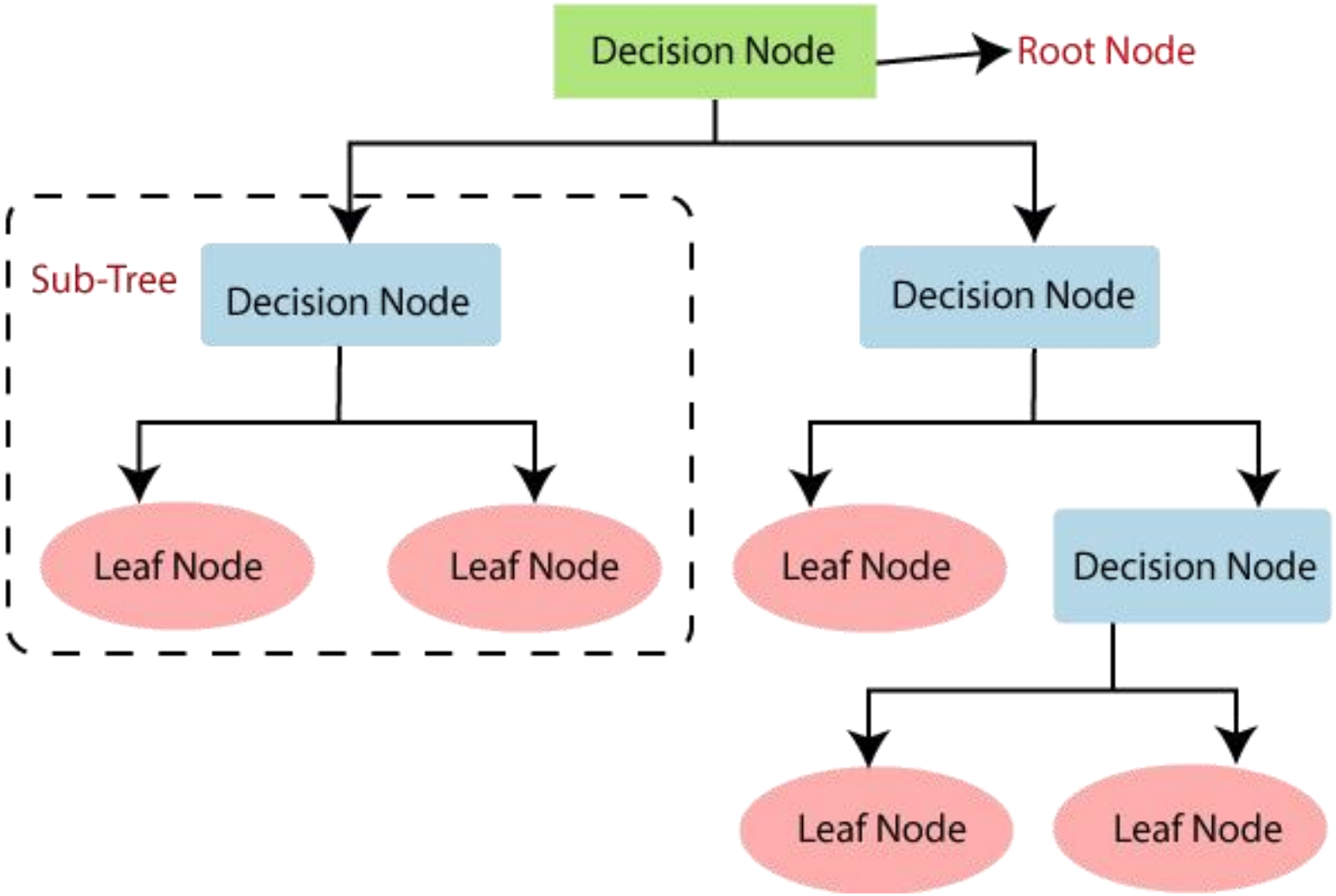
- It is a **tree-structured classifier**, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.

- In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.

- The decisions or the test are performed on the basis of features of the given dataset.

- **It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.**

Decision Tree Classification Algorithm



- It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.
- A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into subtrees.

Why use Decision Trees?

There are various algorithms in Machine learning, so choosing the best algorithm for the given dataset and problem is the main point to remember while creating a machine learning model.

- Decision Trees usually **mimic human thinking ability** while making a decision, so it is easy to understand.

- The logic behind the decision tree can be easily understood because it shows a tree-like structure.

Decision Tree Terminologies

Root Node: Root node is from where the decision tree starts. It represents the entire dataset, which further gets divided into two or more homogeneous sets.

Leaf Node: Leaf nodes are the final output node, and the tree cannot be segregated further after getting a leaf node.

Splitting: Splitting is the process of dividing the decision node/root node into sub-nodes according to the given conditions.

Branch/Sub Tree: A tree formed by splitting the tree.

Pruning: Pruning is the process of removing the unwanted branches from the tree.

Parent/Child node: The root node of the tree is called the parent node, and other nodes are called the child nodes.

How does the Decision Tree algorithm Work?

Step-1: Begin the tree with the root node, says S, which contains the complete dataset.

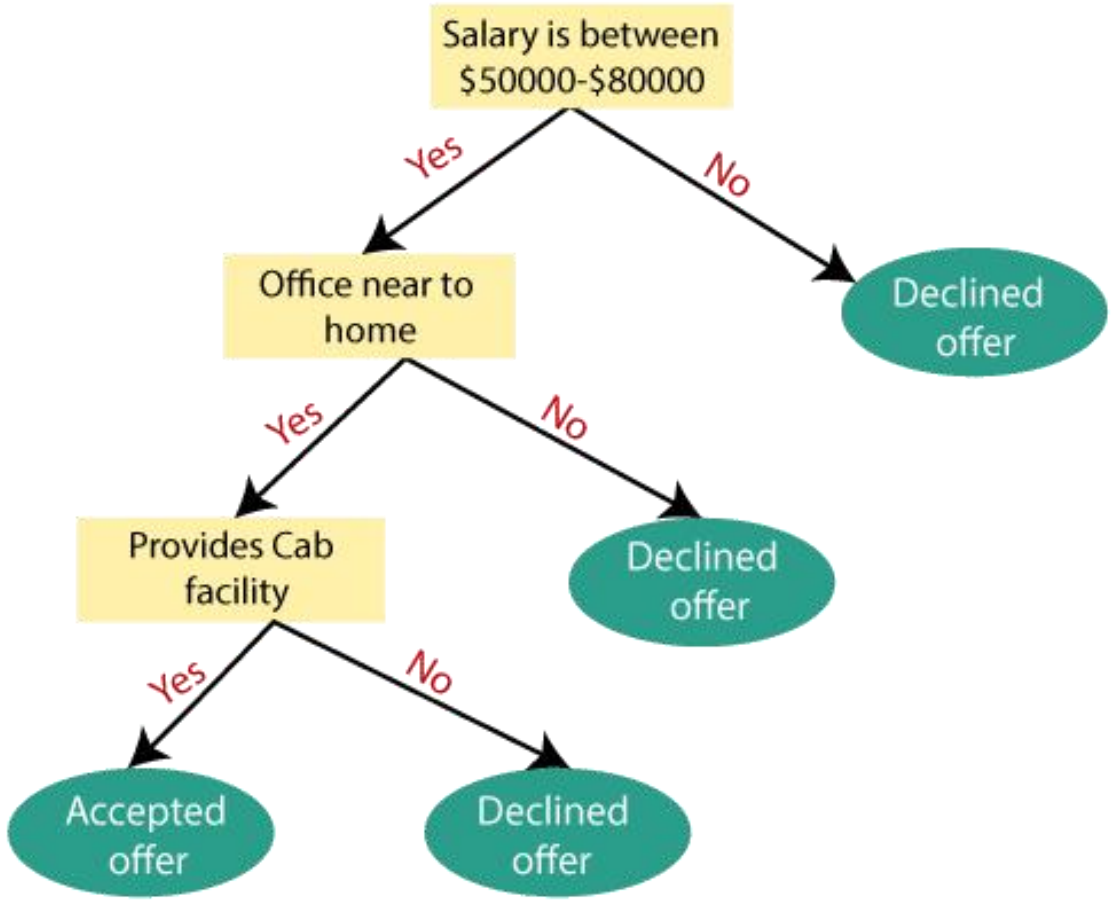
Step-2: Find the best attribute in the dataset using **Attribute Selection Measure (ASM)**.

Step-3: Divide the S into subsets that contains possible values for the best attributes.

Step-4: Generate the decision tree node, which contains the best attribute.

Step-5: Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

Example: Suppose there is a candidate who has a job offer and wants to decide whether he should accept the offer or Not. So, to solve this problem, the decision tree starts with the root node (Salary attribute by ASM). The root node splits further into the next decision node (distance from the office) and one leaf node based on the corresponding labels. The next decision node further gets split into one decision node (Cab facility) and one leaf node. Finally, the decision node splits into two leaf nodes **(Accepted offers and Declined offer)**.



Attribute Selection Measures

While implementing a Decision tree, the main issue arises that how to select the best attribute for the root node and for sub-nodes. So, to solve such problems there is a technique which is called as **Attribute selection measure or ASM**.

By this measurement, we can easily select the best attribute for the nodes of the tree. There are two popular techniques for **ASM**, which are:

- **Information Gain**
- **Gini Index**

Information Gain:

- Information gain is the measurement of changes in entropy after the segmentation of a dataset based on an attribute.
- It calculates how **much information a feature provides us about a class**.
- According to the value of information gain, we split the node and build the decision tree.
- A decision tree algorithm always tries to maximize the value of information gain, and a **node/attribute having the highest information gain is split first**.

Information Gain= Entropy(S)- [(Weighted Avg) *Entropy(each feature)

Entropy: Entropy is a metric to measure the impurity in a given attribute. It specifies randomness in data. Entropy can be calculated as:

Entropy(s)= -P(yes)log2 P(yes)- P(no) log2 P(no)

Information Gain:

*Information Gain= Entropy(S)- [(Weighted Avg) *Entropy(each feature)*

Entropy(s)= -P(yes)log2 P(yes)- P(no) log2 P(no)

Entropy: Entropy is a metric to measure the impurity in a given attribute. It specifies randomness in data.

S= Total number of samples

P(yes)= probability of yes

P(no)= probability of no

Decision Tree ID3 ALGORITHM

Day	Outlook	Temp	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

ID3 (Iterative Dichotomiser 3) is an algorithm invented by Ross Quinlan used to generate a decision tree from a dataset.

Day	Outlook	Temp	Humidity	Wind	Play Tennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Attribute: Outlook

$$Values\,(Outlook) = Sunny, Overcast, Rain$$

$$S = [9+, 5-] \qquad Entropy(S) = -\frac{9}{14}log_2\frac{9}{14} - \frac{5}{14}log_2\frac{5}{14} = 0.94$$

$$S_{Sunny} \leftarrow [2+, 3-] \qquad Entropy(S_{Sunny}) = -\frac{2}{5}log_2\frac{2}{5} - \frac{3}{5}log_2\frac{3}{5} = 0.971$$

$$S_{Overcast} \leftarrow [4+, 0-] \qquad Entropy(S_{Overcast}) = -\frac{4}{4}log_2\frac{4}{4} - \frac{0}{4}log_2\frac{0}{4} = 0$$

$$S_{Rain} \leftarrow [3+, 2-] \qquad Entropy(S_{Rain}) = -\frac{3}{5}log_2\frac{3}{5} - \frac{2}{5}log_2\frac{2}{5} = 0.971$$

$$Gain\,(S, Outlook) = Entropy(S) - \sum_{v \in \{Sunny, Overcast, Rain\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S, Outlook)$$

$$= Entropy(S) - \frac{5}{14}Entropy(S_{Sunny}) - \frac{4}{14}Entropy(S_{Overcast})$$

$$- \frac{5}{14}Entropy(S_{Rain})$$

$$Gain(S, Outlook) = 0.94 - \frac{5}{14}0.971 - \frac{4}{14}0 - \frac{5}{14}0.971 = 0.2464$$

Day	Outlook	Temp	Humidity	Wind	Play Tennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Attribute: Temp

$$Values\,(Temp) = Hot, Mild, Cool$$

$$S = [9+, 5-] \qquad Entropy(S) = -\frac{9}{14}log_2\frac{9}{14} - \frac{5}{14}log_2\frac{5}{14} = 0.94$$

$$S_{Hot} \leftarrow [2+, 2-] \qquad Entropy(S_{Hot}) = -\frac{2}{4}log_2\frac{2}{4} - \frac{2}{4}log_2\frac{2}{4} = 1.0$$

$$S_{Mild} \leftarrow [4+, 2-] \qquad Entropy(S_{Mild}) = -\frac{4}{6}log_2\frac{4}{6} - \frac{2}{6}log_2\frac{2}{6} = 0.9183$$

$$S_{Cool} \leftarrow [3+, 1-] \qquad Entropy(S_{Cool}) = -\frac{3}{4}log_2\frac{3}{4} - \frac{1}{4}log_2\frac{1}{4} = 0.8113$$

$$Gain\,(S, Temp) = Entropy(S) - \sum_{v \in \{Hot, Mild, Cool\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S, Temp)$$

$$= Entropy(S) - \frac{4}{14}Entropy(S_{Hot}) - \frac{6}{14}Entropy(S_{Mild})$$

$$- \frac{4}{14}Entropy(S_{Cool}) \quad = \quad 0.0289$$

Day	Outlook	Temp	Humidity	Wind	Play Tennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Attribute: Humidity

$$Values\,(Humidity) = High, Normal$$

$$S = [9+, 5-] \qquad Entropy(S) = -\frac{9}{14}log_2\frac{9}{14} - \frac{5}{14}log_2\frac{5}{14} = 0.94$$

$$S_{High} \leftarrow [3+, 4-] \qquad Entropy(S_{High}) = -\frac{3}{7}log_2\frac{3}{7} - \frac{4}{7}log_2\frac{4}{7} = 0.9852$$

$$S_{Normal} \leftarrow [6+, 1-] \qquad Entropy(S_{Normal}) = -\frac{6}{7}log_2\frac{6}{7} - \frac{1}{7}log_2\frac{1}{7} = 0.5916$$

$$Gain\,(S, Humidity) = Entropy(S) - \sum_{v \in \{High, Normal\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S, Humidity)$$

$$= Entropy(S) - \frac{7}{14}Entropy(S_{High}) - \frac{7}{14}Entropy(S_{Normal})$$

$$Gain(S, Humidity) = 0.94 - \frac{7}{14}0.9852 - \frac{7}{14}0.5916 =$$

$$= \mathbf{0.0516}$$

Day	Outlook	Temp	Humidity	Wind	Play Tennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Attribute: Wind

$Values\ (Wind) = Strong, Weak$

$S = [9+, 5-]$ $\quad Entropy(S) = -\frac{9}{14} log_2 \frac{9}{14} - \frac{5}{14} log_2 \frac{5}{14} = 0.94$

$S_{Strong} \leftarrow [3+, 3-]$ $\quad Entropy(S_{Strong}) = 1.0$

$S_{Weak} \leftarrow [6+, 2-]$ $\quad Entropy(S_{Weak}) = -\frac{6}{8} log_2 \frac{6}{8} - \frac{2}{8} log_2 \frac{2}{8} = 0.8113$

$$Gain\ (S, Wind) = Entropy(S) - \sum_{v \in \{Strong, Weak\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S, Wind) = Entropy(S) - \frac{6}{14} Entropy(S_{Strong}) - \frac{8}{14} Entropy(S_{Weak})$$

$$Gain(S, Wind) = 0.94 - \frac{6}{14}\ 1.0 - \frac{8}{14}\ 0.8113 :$$

$= 0.0478$

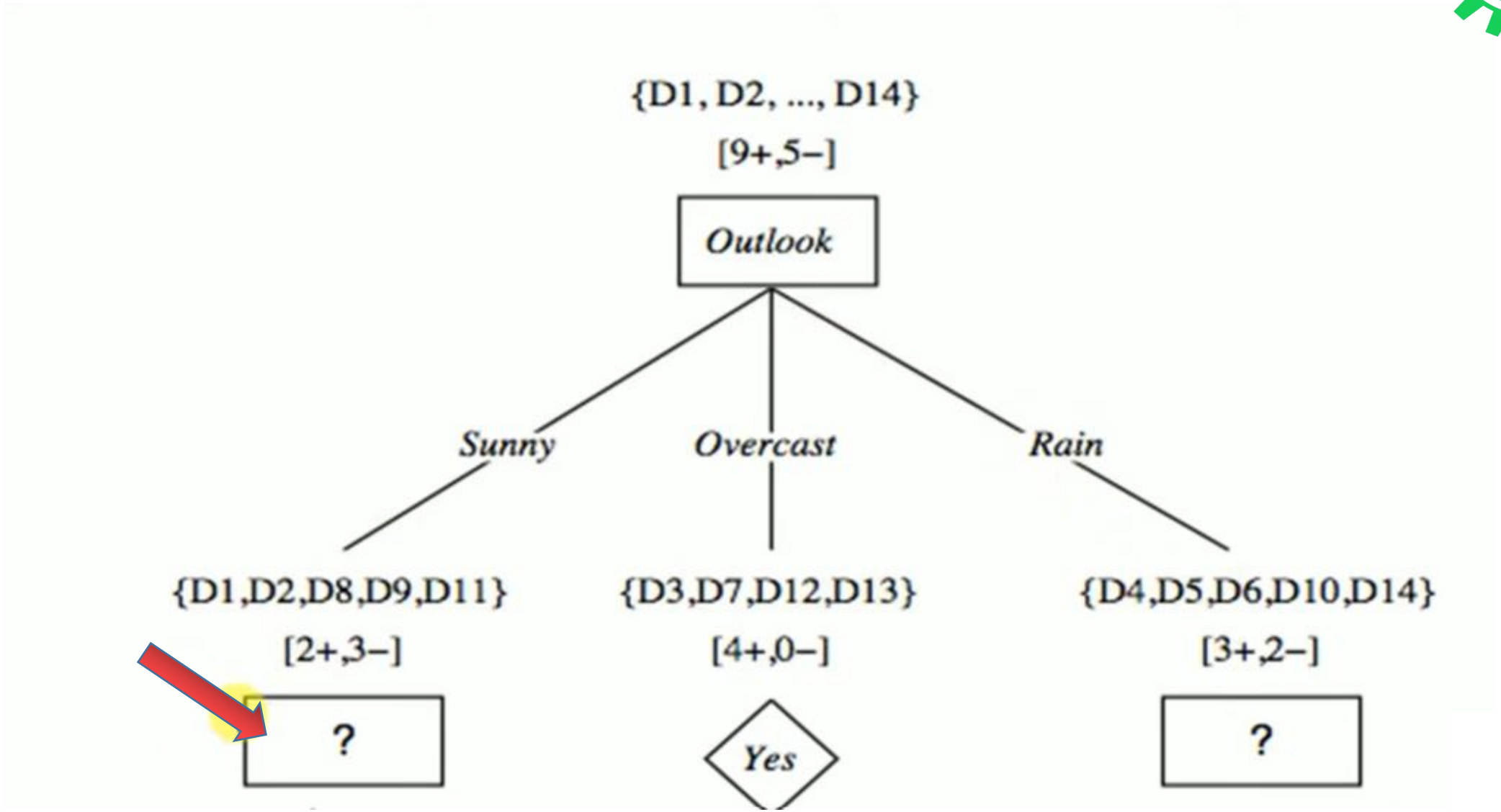
Day	Outlook	Temp	Humidity	Wind	Play Tennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

$$Gain(S, Outlook) = 0.2464$$

$$Gain(S, Temp) = 0.0289$$

$$Gain(S, Humidity) = 0.1516$$

$$Gain(S, Wind) = 0.0478$$


{D1, D2, ..., D14}
[9+,5−]
Outlook
Sunny
Overcast
Rain
{D1,D2,D8,D9,D11}
[2+,3−]
{D3,D7,D12,D13}
[4+,0−]
{D4,D5,D6,D10,D14}
[3+,2−]
?
Yes
?

Day	Temp	Humidity	Wind	Play Tennis
D1	Hot	High	Weak	No
D2	Hot	High	Strong	No
D8	Mild	High	Weak	No
D9	Cool	Normal	Weak	Yes
D11	Mild	Normal	Strong	Yes

Attribute: Temp

$$Values\,(Temp) = Hot, Mild, Cool$$

$S_{Sunny} = [2+, 3-]$ $\quad Entropy(S_{Sunny}) = -\frac{2}{5}log_2\frac{2}{5} - \frac{3}{5}log_2\frac{3}{5} = 0.97$

$S_{Hot} \leftarrow [0+, 2-]$ $\quad Entropy(S_{Hot}) = 0.0$

$S_{Mild} \leftarrow [1+, 1-]$ $\quad Entropy(S_{Mild}) =$ 1.0

$S_{Cool} \leftarrow [1+, 0-]$ $\quad Entropy(S_{Cool}) = 0.0$

$$Gain\,(S_{Sunny}, Temp) = Entropy(S) - \sum_{v \in \{Hot, Mild, Cool\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S_{Sunny}, Temp)$$

$$= Entropy(S) - \frac{2}{5}Entropy(S_{Hot}) - \frac{2}{5}Entropy(S_{Mild})$$

$$- \frac{1}{5}Entropy(S_{Cool})$$

$$Gain(S_{sunny}, Temp) = 0.97 - \frac{2}{5}0.0 - \frac{2}{5}1 - \frac{1}{5}0.0 = 0.570$$

Day	Temp	Humidity	Wind	Play Tennis
Dl	Hot	High	Weak	No
D2	Hot	High	Strong	No
D8	Mild	High	Weak	No
D9	Cool	Normal	Weak	Yes
Dl1	Mild	Normal	Strong	Yes

Attribute: Humidity

$$Values\,(Humidity) = High, Normal$$

$$S_{Sunny} = [2+, 3-] \qquad Entropy(S) = -\frac{2}{5}log_2\frac{2}{5} - \frac{3}{5}log_2\frac{3}{5} = 0.97$$

$$S_{high} \leftarrow [0+, 3-] \qquad Entropy(S_{High}) = 0.0$$

$$S_{Normal} \leftarrow [2+, 0-] \qquad Entropy(S_{Normal}) = 0.0$$

$$Gain\,(S_{Sunny}, Humidity) = Entropy(S) - \sum_{v \in \{High, Normal\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S_{Sunny}, Humidity) = Entropy(S) - \frac{3}{5}Entropy(S_{High}) - \frac{2}{5}Entropy(S_{Normal})$$

$$Gain(S_{Sunny}, Humidity) = 0.97 - \frac{3}{5}0.0 - \frac{2}{5}0.0 = 0.97$$

Day	Temp	Humidity	Wind	Play Tennis
Dl	Hot	High	Weak	No
D2	Hot	High	Strong	No
D8	Mild	High	Weak	No
D9	Cool	Normal	Weak	Yes
Dl1	Mild	Normal	Strong	Yes

Attribute: Wind

$Values\ (Wind) = Strong, Weak$

$S_{Sunny} = [2+, 3-]$ $\quad Entropy(S) = -\frac{2}{5}log_2\frac{2}{5} - \frac{3}{5}log_2\frac{3}{5} = 0.97$

$S_{Strong} \leftarrow [1+, 1-]$ $\quad Entropy(S_{Strong}) = 1.0$

$S_{Weak} \leftarrow [1+, 2-]$ $\quad Entropy(S_{Weak}) = -\frac{1}{3}log_2\frac{1}{3} - \frac{2}{3}log_2\frac{2}{3} = 0.9183$

$$Gain\ (S_{Sunny}, Wind) = Entropy(S) - \sum_{v \in \{Strong, Weak\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

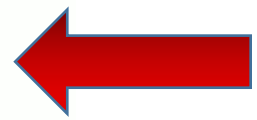
$$Gain(S_{Sunny}, Wind) = Entropy(S) - \frac{2}{5}Entropy(S_{Strong}) - \frac{3}{5}Entropy(S_{Weak})$$

$$Gain(S_{sunny}, Wind) = 0.97 - \frac{2}{5}1.0 - \frac{3}{5}\ 0.918 = 0.0192$$

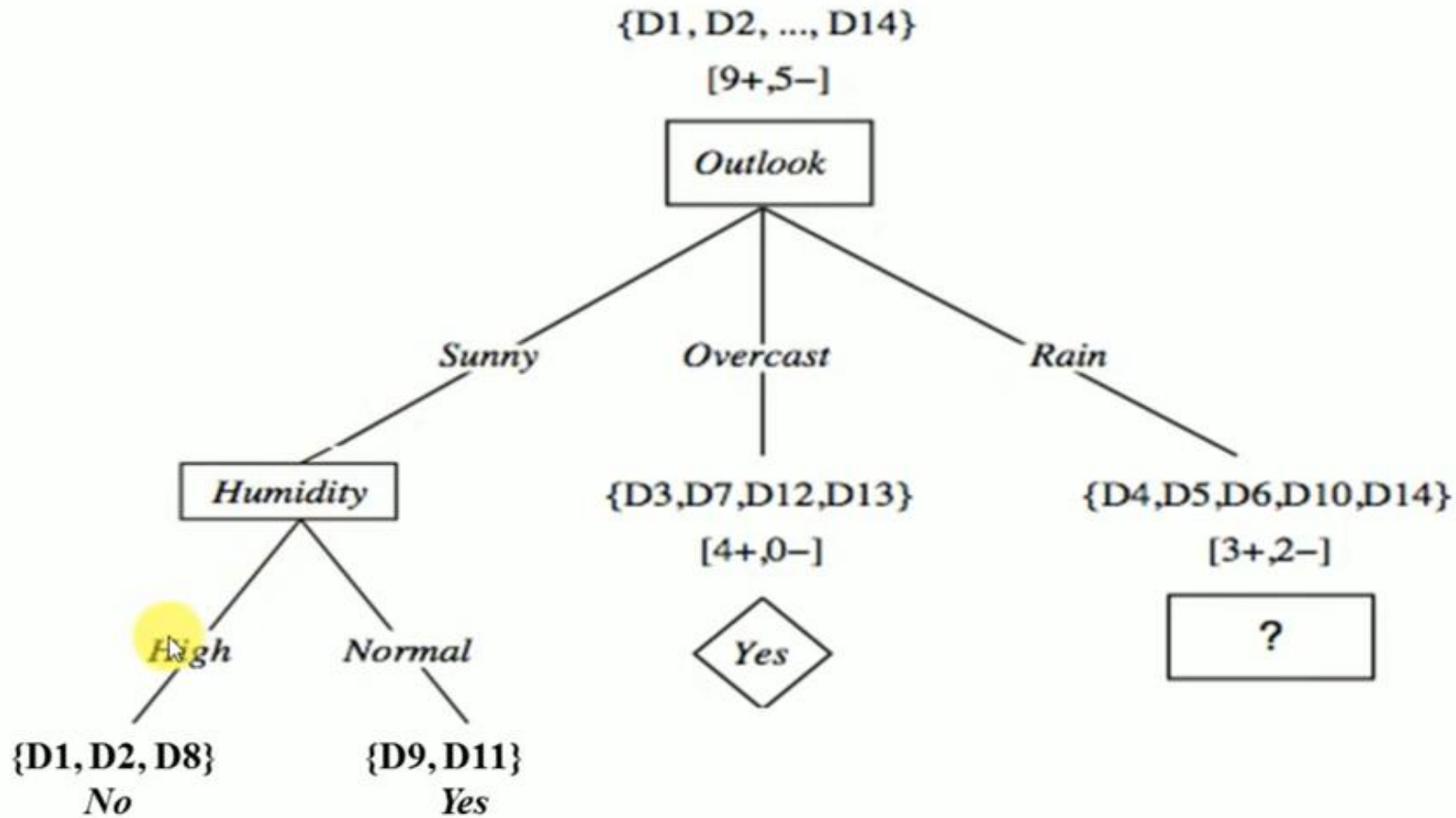
Day	Temp	Humidity	Wind	Play Tennis
D1	Hot	High	Weak	No
D2	Hot	High	Strong	No
D8	Mild	High	Weak	No
D9	Cool	Normal	Weak	Yes
D11	Mild	Normal	Strong	Yes

$$Gain(S_{sunny}, Temp) = 0.570$$

$$Gain(S_{sunny}, Humidity) = 0.97$$



$$Gain(S_{sunny}, Wind) = 0.0192$$


{D1, D2, ..., D14}
[9+,5−]
Outlook
Sunny
Overcast
Rain
Humidity
{D3,D7,D12,D13}
[4+,0−]
{D4,D5,D6,D10,D14}
[3+,2−]
High
Normal
Yes
?
{D1, D2, D8}
No
{D9, D11}
Yes

Day	Temp	Humidity	Wind	Play Tennis
D4	Mild	High	Weak	Yes
D5	Cool	Normal	Weak	Yes
D6	Cool	Normal	Strong	No
D10	Mild	Normal	Weak	Yes
D14	Mild	High	Strong	No

Attribute: Temp

$Values\ (Temp) = Hot, Mild, Cool$

$S_{Rain} = [3+, 2-]$

$S_{Hot} \leftarrow [0+, 0-]$

$S_{Mild} \leftarrow [2+, 1-]$

$S_{Cool} \leftarrow [1+, 1-]$

$$Entropy(S_{Sunny}) = -\frac{3}{5}log_2\frac{3}{5} - \frac{2}{5}log_2\frac{2}{5} = 0.97$$

$$Entropy(S_{Hot}) = 0.0$$

$$Entropy(S_{Mild}) = -\frac{2}{3}log_2\frac{2}{3} - \frac{1}{3}log_2\frac{1}{3} = 0.9183$$

$$Entropy(S_{Cool}) = 1.0$$

$$Gain\ (S_{Rain}, Temp) = Entropy(S) - \sum_{v \in \{Hot, Mild, Cool\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S_{Rain}, Temp)$$

$$= Entropy(S) - \frac{0}{5}Entropy(S_{Hot}) - \frac{3}{5}Entropy(S_{Mild})$$

$$- \frac{2}{5}Entropy(S_{Cool})$$

$$Gain(S_{Rain}, Temp) = 0.97 - \frac{0}{5}0.0 - \frac{3}{5}0.918 - \frac{2}{5}1.0 = 0.0192$$

Day	Temp	Humidity	Wind	Play Tennis
D4	Mild	High	Weak	Yes
D5	Cool	Normal	Weak	Yes
D6	Cool	Normal	Strong	No
Dl0	Mild	Normal	Weak	Yes
Dl4	Mild	High	Strong	No

Attribute: Humidity

$$Values\,(Humidity) = High, Normal$$

$S_{Rain} = [3+, 2-]$ $\quad Entropy(S_{Sunny}) = -\frac{3}{5}log_2\frac{3}{5} - \frac{2}{5}log_2\frac{2}{5} = 0.97$

$S_{High} \leftarrow [1+, 1-]$ $\quad Entropy(S_{High}) = 1.0$

$S_{Normal} \leftarrow [2+, 1-]$ $\quad Entropy(S_{Normal}) = -\frac{2}{3}log_2\frac{2}{3} - \frac{1}{3}log_2\frac{1}{3} = 0.9183$

$$Gain\,(S_{Rain}, Humidity) = Entropy(S) - \sum_{v \in \{High, Normal\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S_{Rain}, Humidity) = Entropy(S) - \frac{2}{5}Entropy(S_{High}) - \frac{3}{5}Entropy(S_{Normal})$$

$$Gain(S_{Rain}, Humidity) = 0.97 - \frac{2}{5}\,1.0 - \frac{3}{5}0.918 = 0.0192$$

Day	Temp	Humidity	Wind	Play Tennis
D4	Mild	High	Weak	Yes
D5	Cool	Normal	Weak	Yes
D6	Cool	Normal	Strong	No
Dl0	Mild	Normal	Weak	Yes
Dl4	Mild	High	Strong	No

Attribute: Wind

$Values\ (wind) = Strong, Weak$

$S_{Rain} = [3+, 2-]$

$S_{Strong} \leftarrow [0+, 2-]$

$S_{Weak} \leftarrow [3+, 0-]$

$$Entropy(S_{Sunny}) = -\frac{3}{5}log_2\frac{3}{5} - \frac{2}{5}log_2\frac{2}{5} = 0.97$$

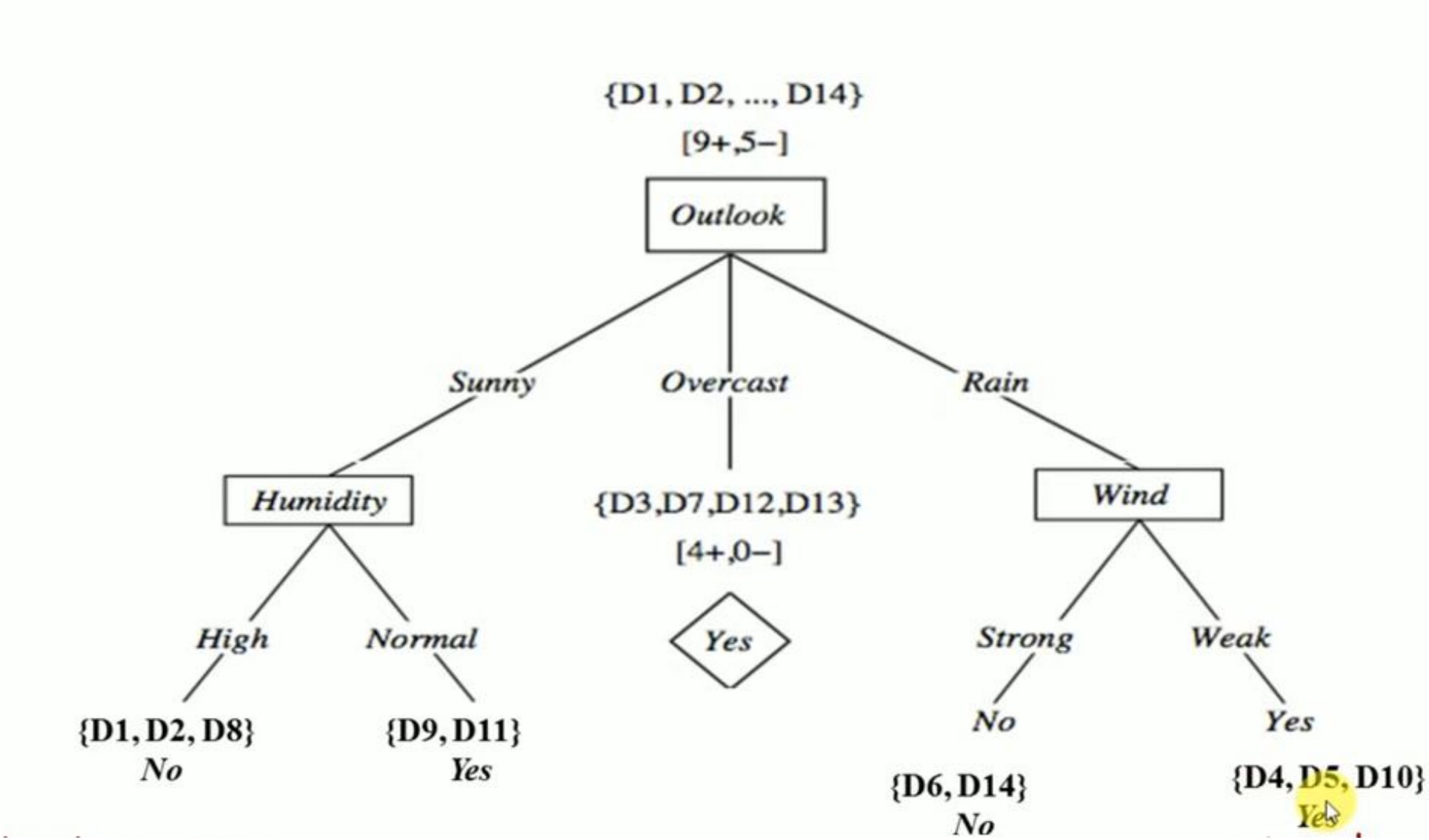
$$Entropy(S_{Strong}) = 0.0$$

$$Entropy(S_{weak}) = 0.0$$

$$Gain\ (S_{Rain}, Wind) = Entropy(S) - \sum_{v \in \{Strong, Weak\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S_{Rain}, Wind) = Entropy(S) - \frac{2}{5}Entropy(S_{Strong}) - \frac{3}{5}Entropy(S_{Weak})$$

$$Gain(S_{Rain}, Wind) = 0.97 - \frac{2}{5}\ 0.0 - \frac{3}{5}0.0 = 0.97$$


{D1, D2, ..., D14}
[9+,5−]
Outlook
Sunny
Overcast
Rain
Humidity
{D3,D7,D12,D13}
[4+,0−]
Wind
High
Normal
Yes
Strong
Weak
{D1, D2, D8}
No
{D9, D11}
Yes
No
Yes
{D6, D14}
No
{D4, D5, D10}
Yes

Advantages of the Decision Tree

- It is simple to understand as it follows the same process which a human follow while making any decision in real-life.
- It can be very useful for solving decision-related problems.
- It helps to think about all the possible outcomes for a problem.
- There is less requirement of data cleaning compared to other algorithms.

Disadvantages of the Decision Tree

- The decision tree contains lots of layers, which makes it complex.
- It may have an overfitting issue, which can be resolved using the Random Forest algorithm.
- For more class labels, the computational complexity of the decision tree may increase.

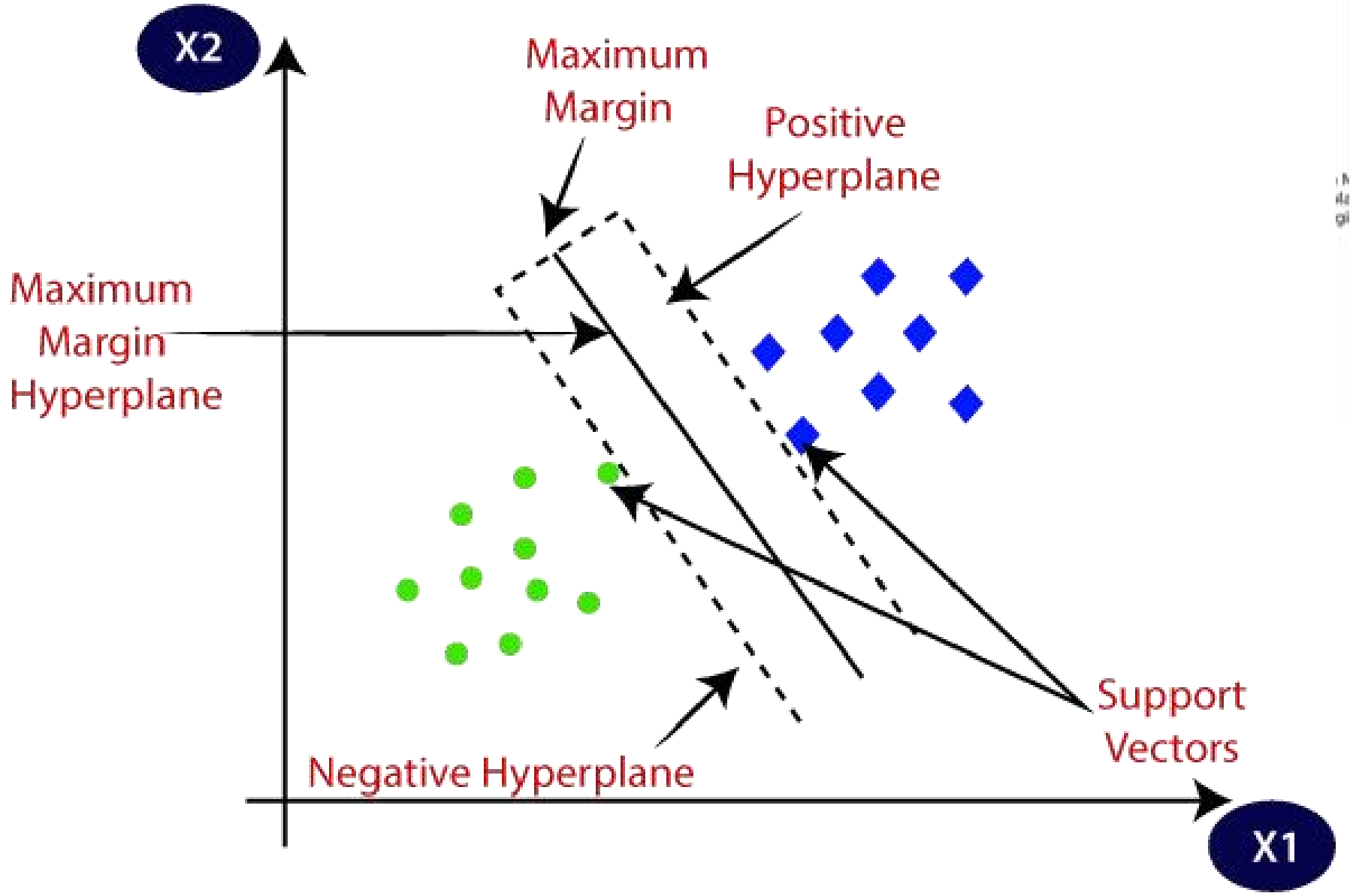
Support Vector Machine

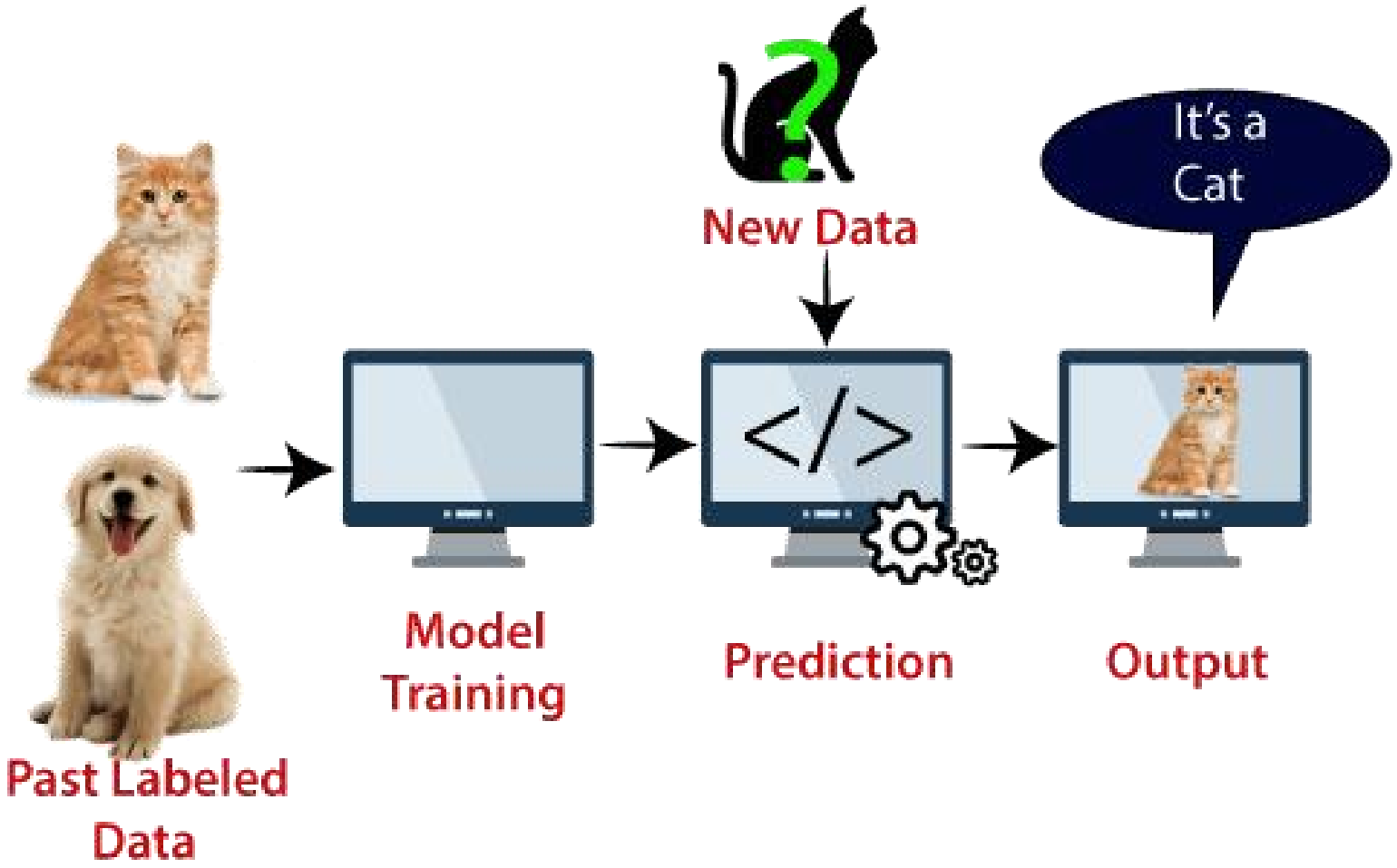
- **Support Vector Machine or SVM** is one of the most popular Supervised Learning algorithms, which is used for **Classification as well as Regression problems**. However, primarily, it is used for **Classification problems** in Machine Learning.

- The goal of the SVM algorithm is to create the **best line or decision boundary** that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This **best decision boundary is called a hyperplane**.

- SVM chooses the **extreme points/vectors** that help in creating the hyperplane.

- These extreme cases are called as support vectors, and hence algorithm is termed as **Support Vector Machine**.


X2
Maximum
Margin
Positive
Hyperplane
Maximum
Margin
Hyperplane
Support
Vectors
Negative Hyperplane
X1


New Data
It's a Cat
Past Labeled Data
Model Training
Prediction
Output

Types of SVM

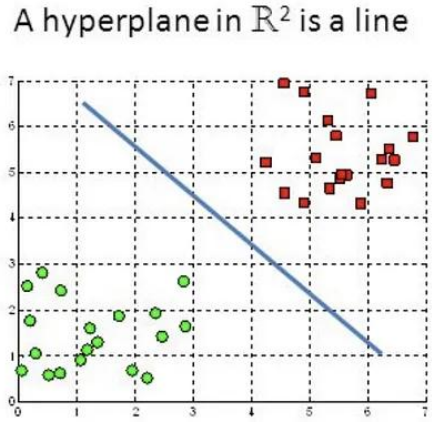
SVM can be of two types:

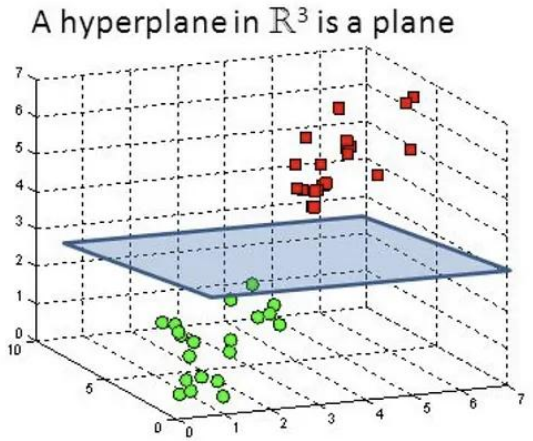
Linear SVM: Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.

Non-linear SVM: Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.

Hyperplane: There can be multiple lines/decision boundaries to segregate the classes in n-dimensional space, but we need to find out the **best decision boundary** that helps to classify the data points. This best boundary is known as the **hyperplane of SVM**.

The dimensions of the hyperplane depend on the features present in the dataset, which means if there are **2 features then hyperplane will be a straight line**. And if there are 3 features, then hyperplane will be a **2-dimension plane.**

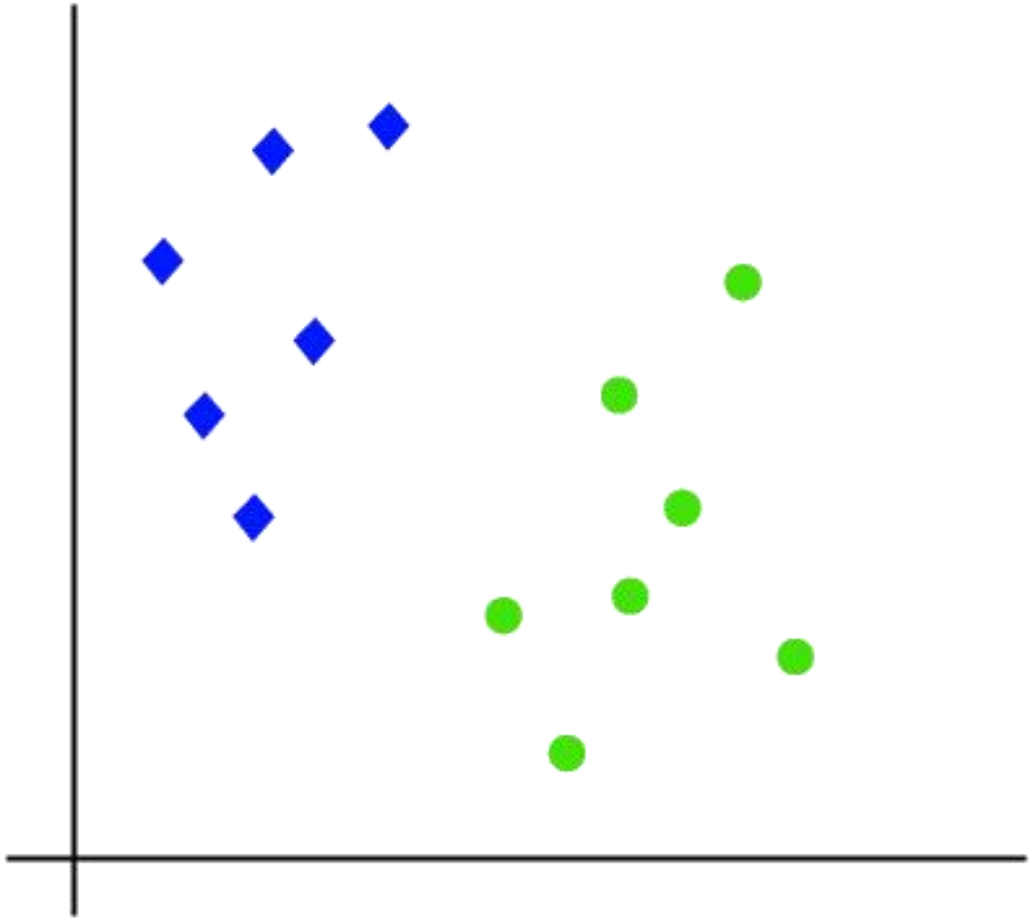


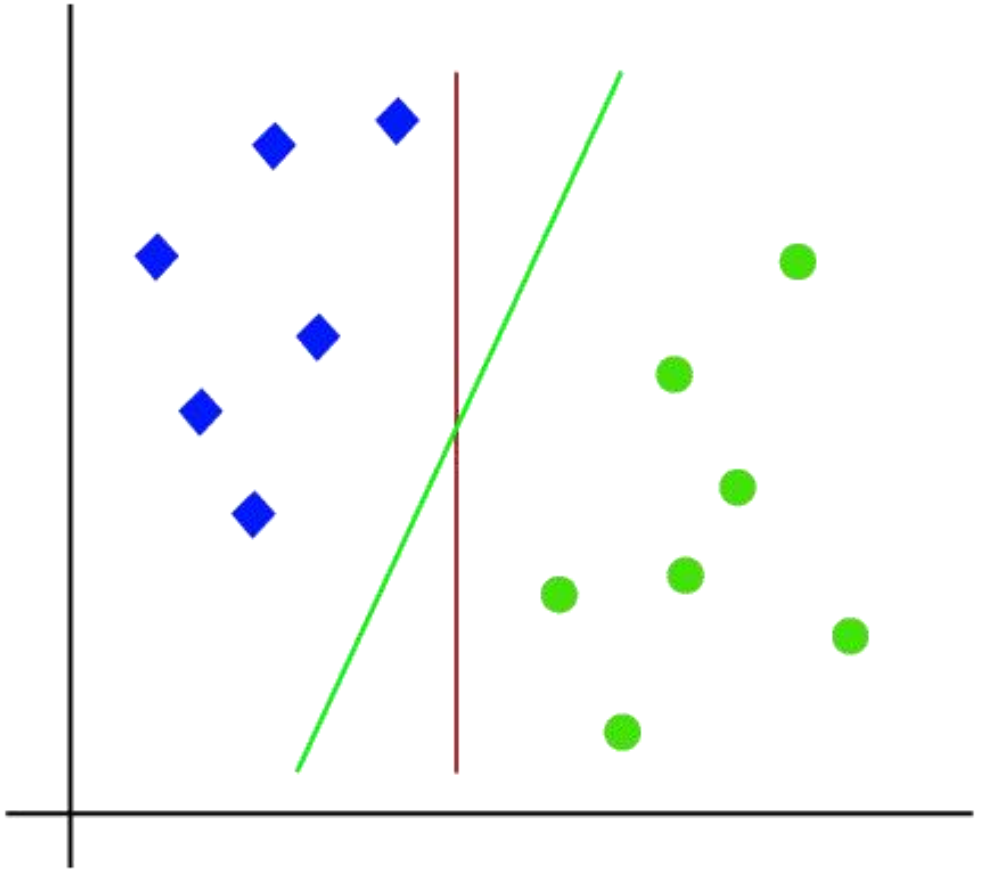


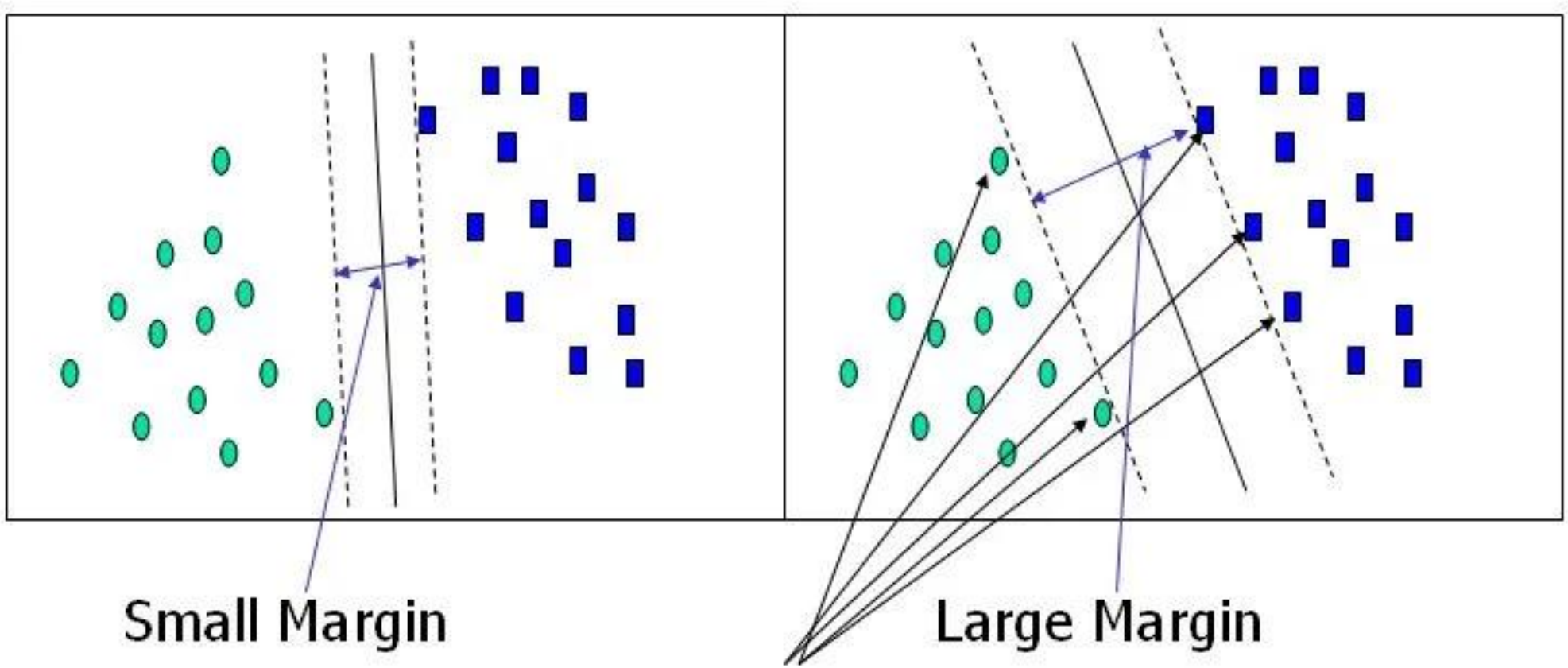
Support Vectors:

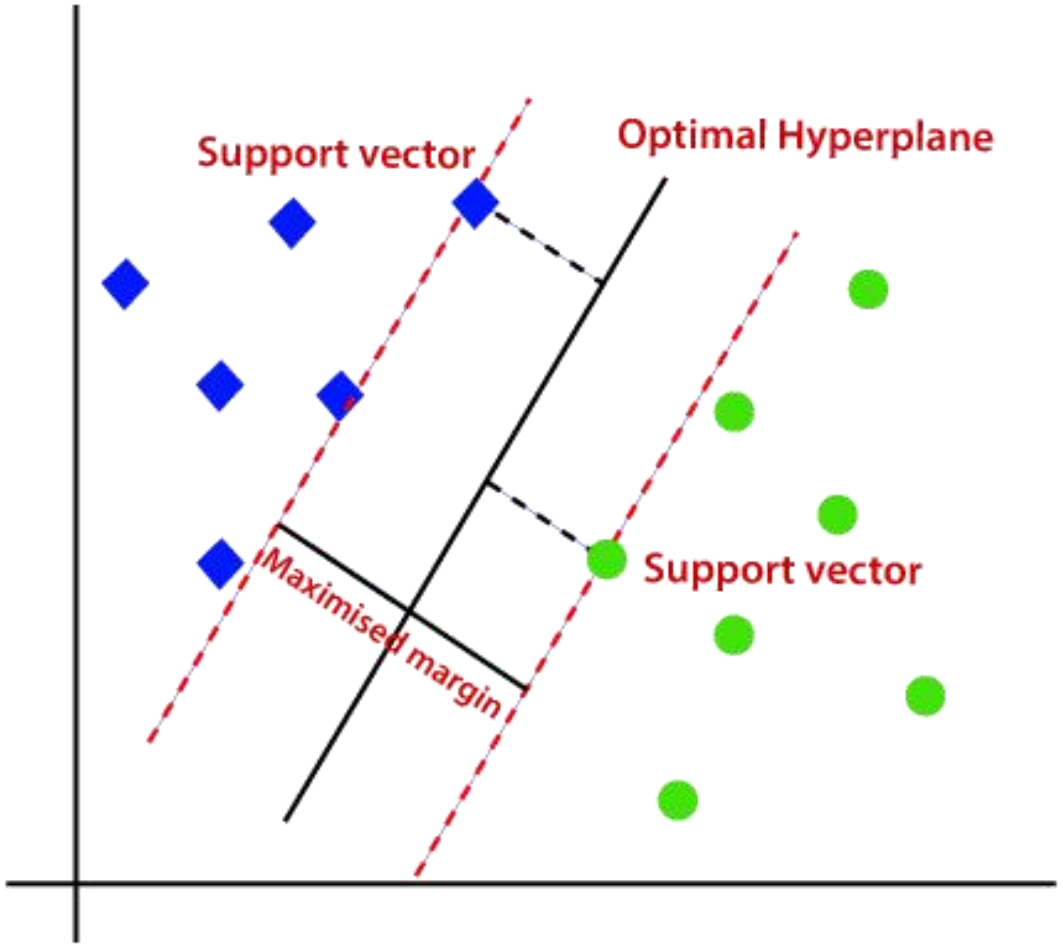
The data points or vectors that are the closest to the hyperplane and which affect the position of the hyperplane are **termed as Support Vector.** Since these vectors support the hyperplane, hence called a Support vector.

How does SVM works? (Linear SVM)






Small Margin
Large Margin
Support Vectors



- SVM algorithm helps to find the best line or decision boundary; this best boundary or region is called as a **hyperplane**.
- SVM algorithm finds the closest point of the lines from both the classes.
- These points are called **support vectors**.
- The distance between the vectors and the hyperplane is called as **margin**.
- The goal of SVM is to **maximize this margin**. **The hyperplane with maximum margin is called the optimal hyperplane.**

Mathematics Behind Support Vector Machine

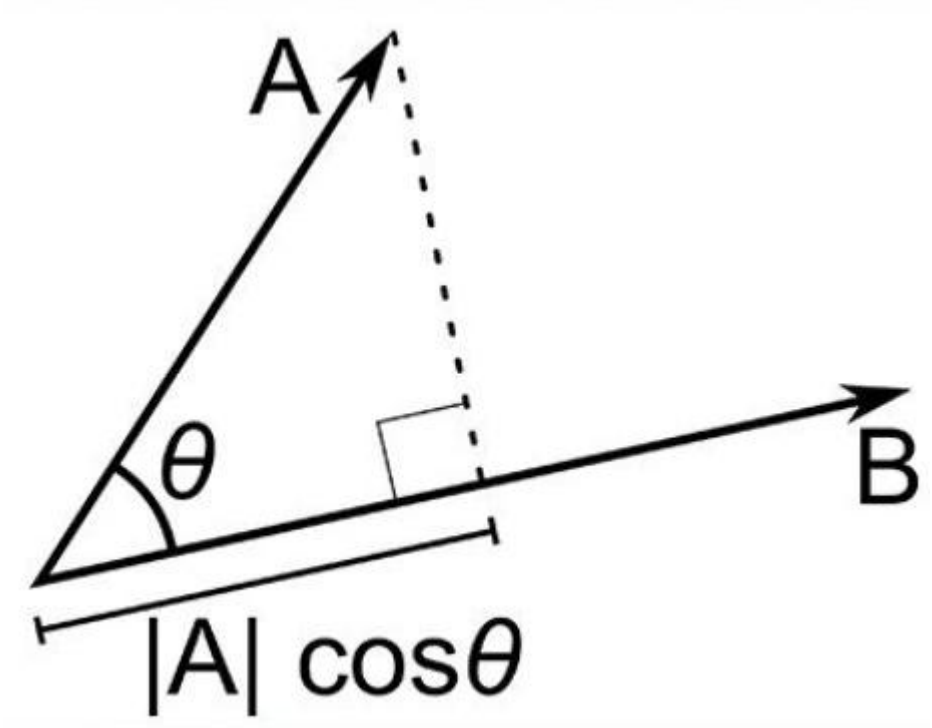
Understanding Dot-Product

We all know that a vector is a quantity that has magnitude as well as direction and just like numbers we can use mathematical operations such as addition, multiplication.

Dot product, and Cross product

The difference is only that the dot product is used to get a scalar value as a resultant whereas cross-product is used to obtain a vector again.

Mathematics Behind Support Vector Machine



A . B = |A| cosθ * |B|

Where |A| cosθ is the projection of A on B

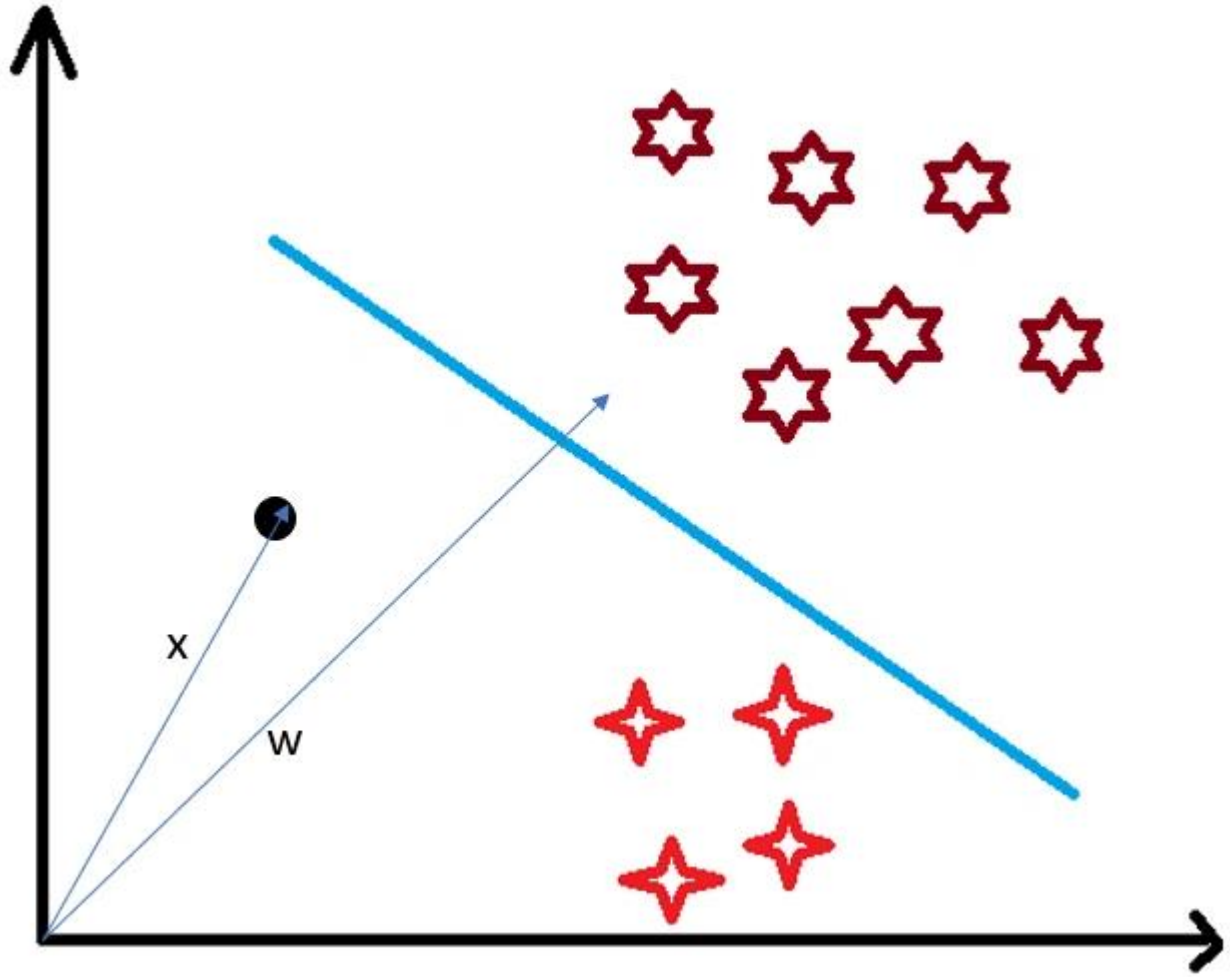
And |B| is the magnitude of vector B

Now in SVM we just need the **projection of A** not the magnitude of B.

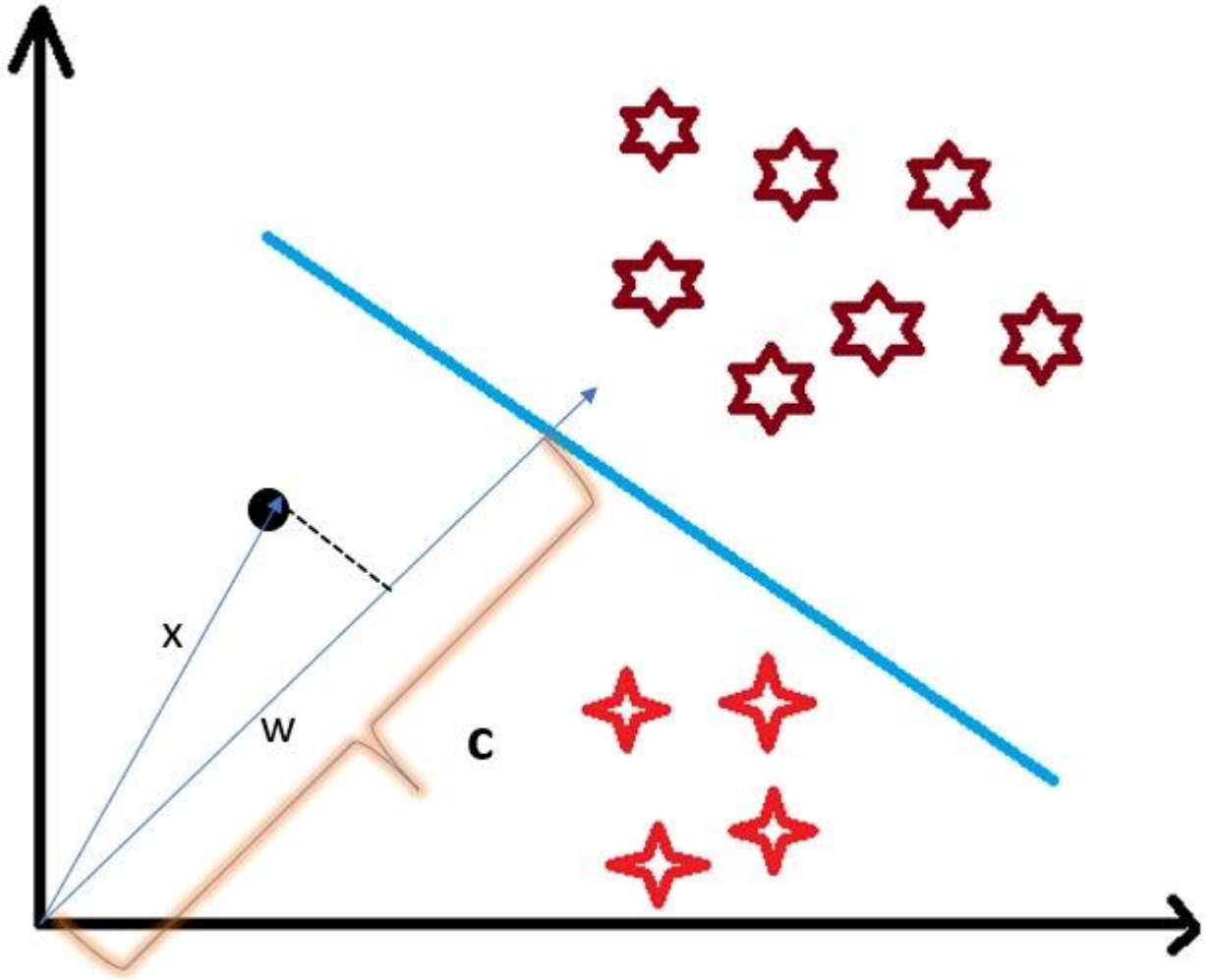
A.B = |A| cosθ * unit vector of B

Use of Dot Product in SVM

Consider a random **point X** and we want to know whether it lies on the right side of the plane or the left side of the plane (positive or negative).



Use of Dot Product in SVM



- To find this first we assume this point is a **vector (X)** .
- And then we make a **vector (w)** which is perpendicular to the hyperplane.
- Let's say the distance of vector w from origin to decision boundary is 'c'.
- Now we take the **projection of X vector on w.**

Use of Dot Product in SVM

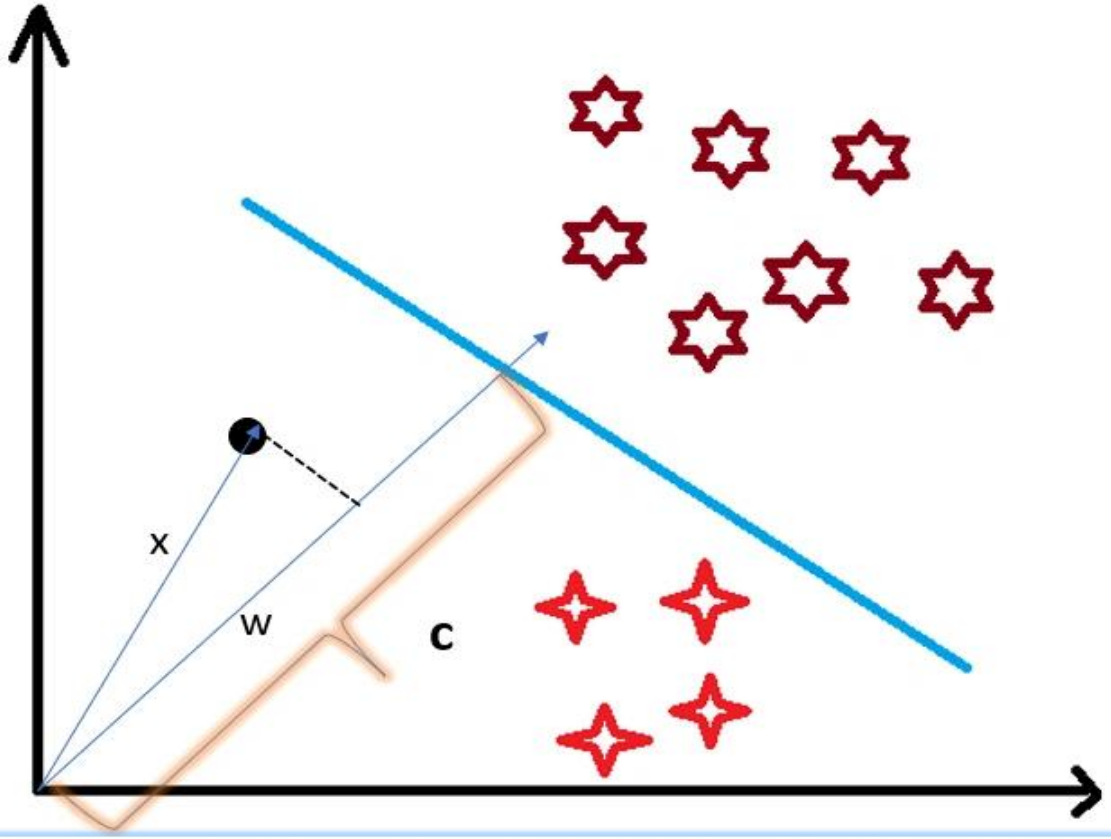
- We already know that projection of any vector or another vector is called dot-product.
- Hence, we take the **dot product of *x* and *w* vectors**.
- If the dot product is greater than ‘c’ then we can say that the point lies on the right side.
- If the dot product is less than ‘c’ then the point is on the left side
- if the dot product is equal to ‘c’ then the point lies on the decision boundary.

$$\vec{X}.\vec{w} = c \quad (the\ point\ lies\ on\ the\ decision\ boundary)$$

$$\vec{X}.\vec{w} > c\ (positive\ samples)$$

$$\vec{X}.\vec{w} < c\ (negative\ samples)$$

Use of Dot Product in SVM

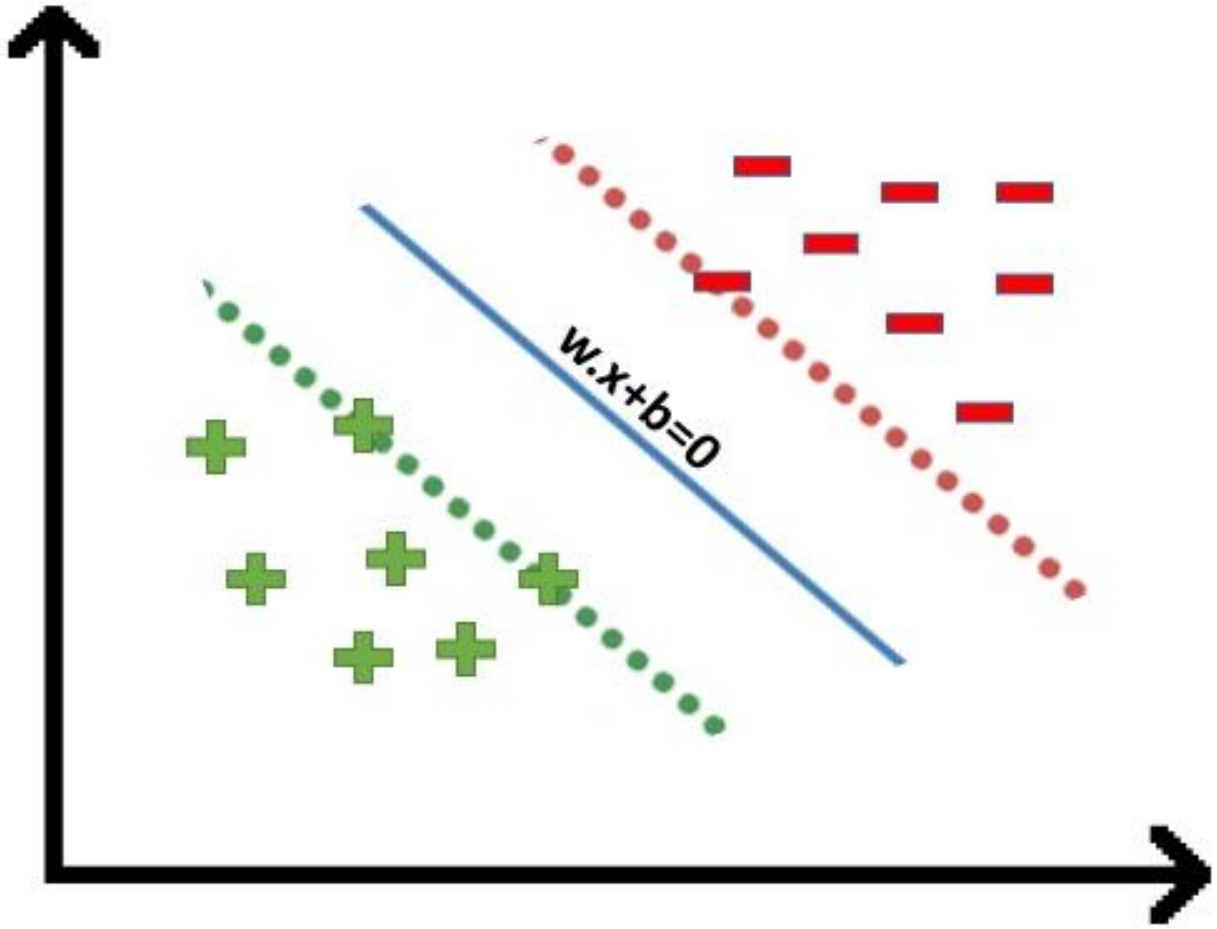


$$\vec{X}.\vec{w} = c \quad (the\ point\ lies\ on\ the\ decision\ boundary)$$

$$\vec{X}.\vec{w} > c \ (positive\ samples)$$

$$\vec{X}.\vec{w} < c \ (negative\ samples)$$

Margin in Support Vector Machine



To classify a point as negative or positive we need to define a decision rule. We can define decision rule as:

$$\vec{X}.\vec{w} - \mathrm{c} \geq 0$$

putting $-\mathrm{c}$ as b, we get

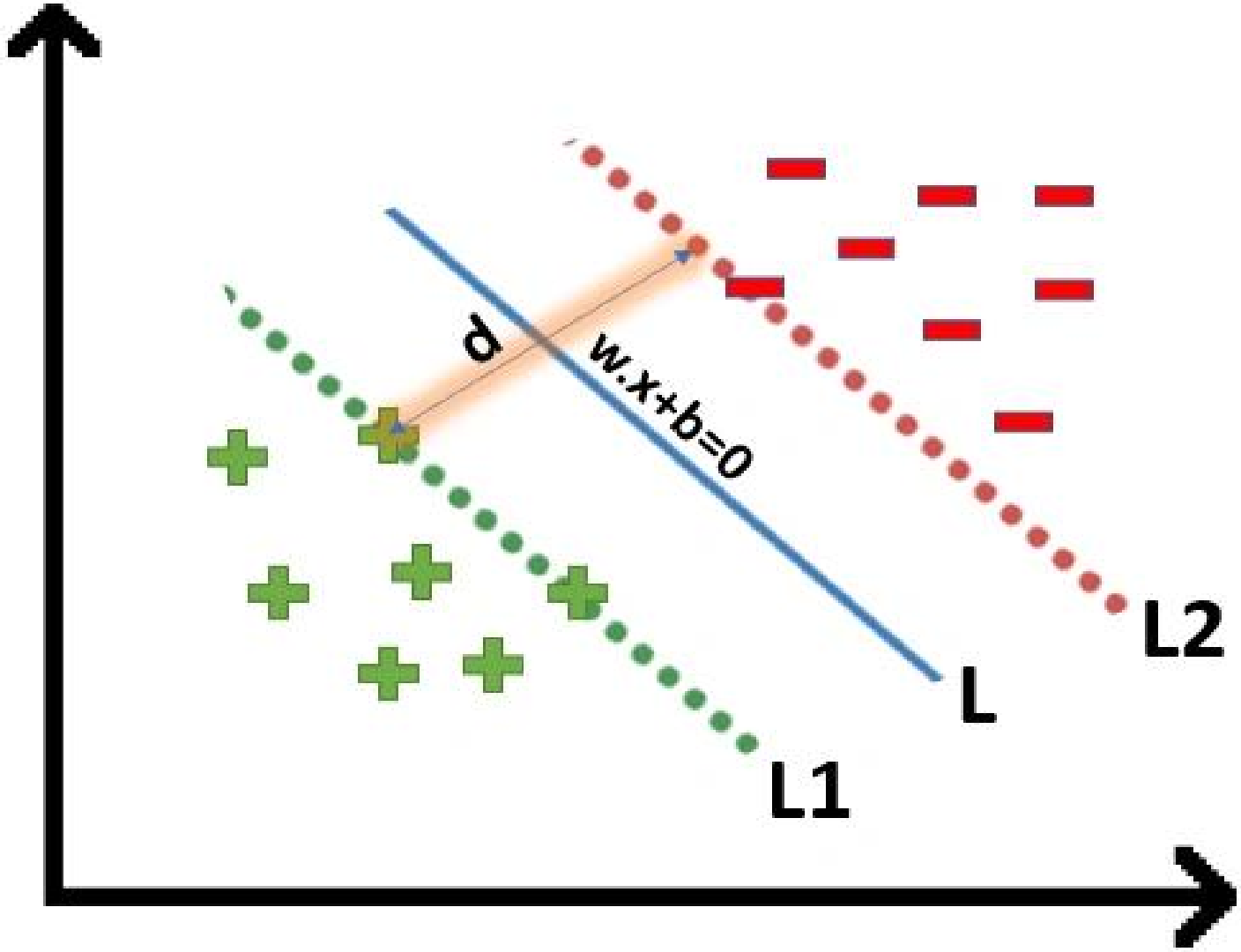
$$\vec{X}.\vec{w} + \mathrm{b} \geq 0$$

hence

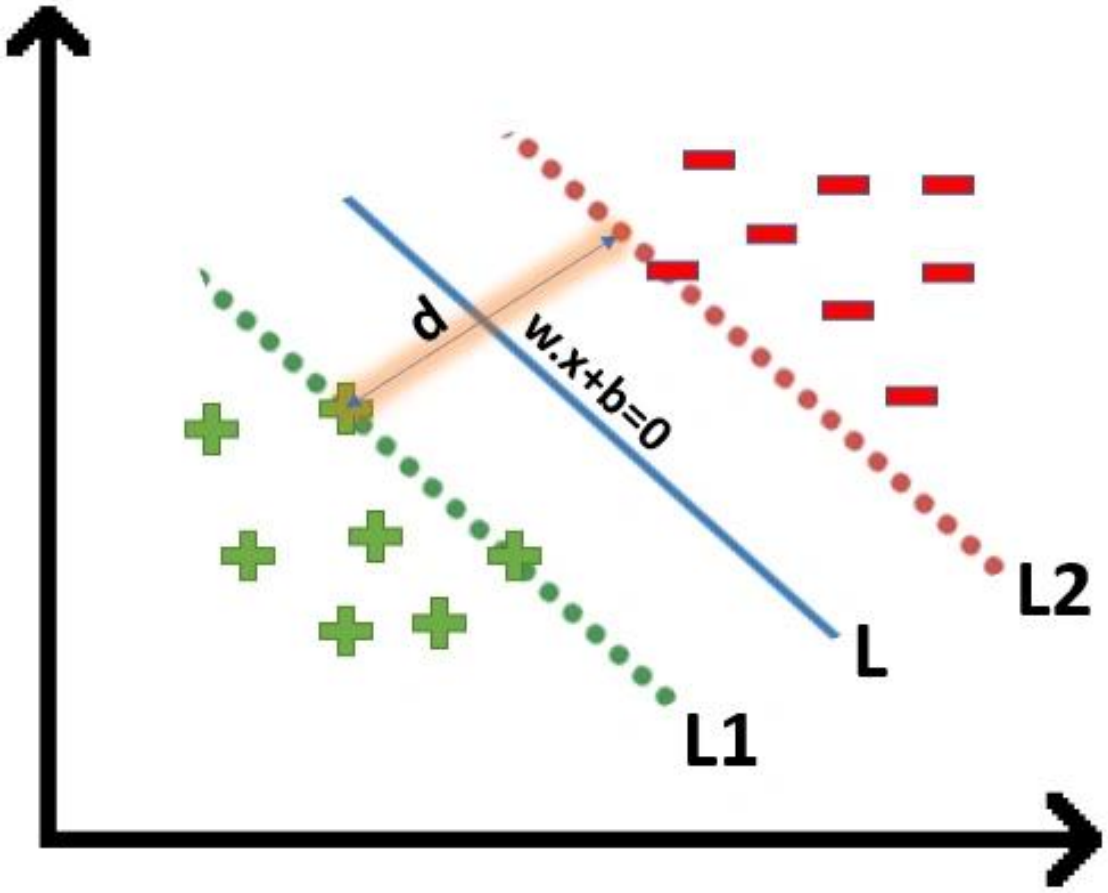
$$\mathrm{y} = \begin{cases} +1 & \text{if } \vec{X}.\vec{w} + \mathrm{b} \geq 0 \\ -1 & \text{if } \vec{X}.\vec{w} + \mathrm{b} < 0 \end{cases}$$

Margin in Support Vector Machine

Now we need (w,b) such that the margin has a maximum distance. Let's say this distance is 'd'.



Margin in Support Vector Machine



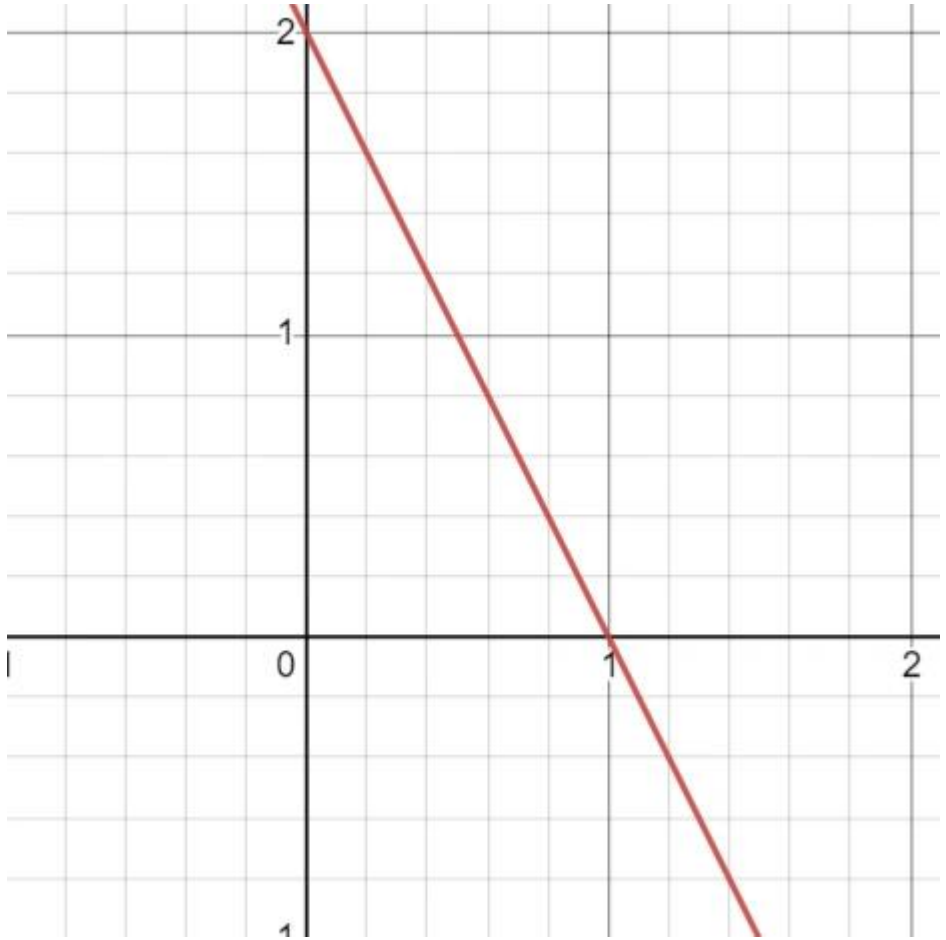
To **calculate 'd'** we need the equation of **L1 and L2**.

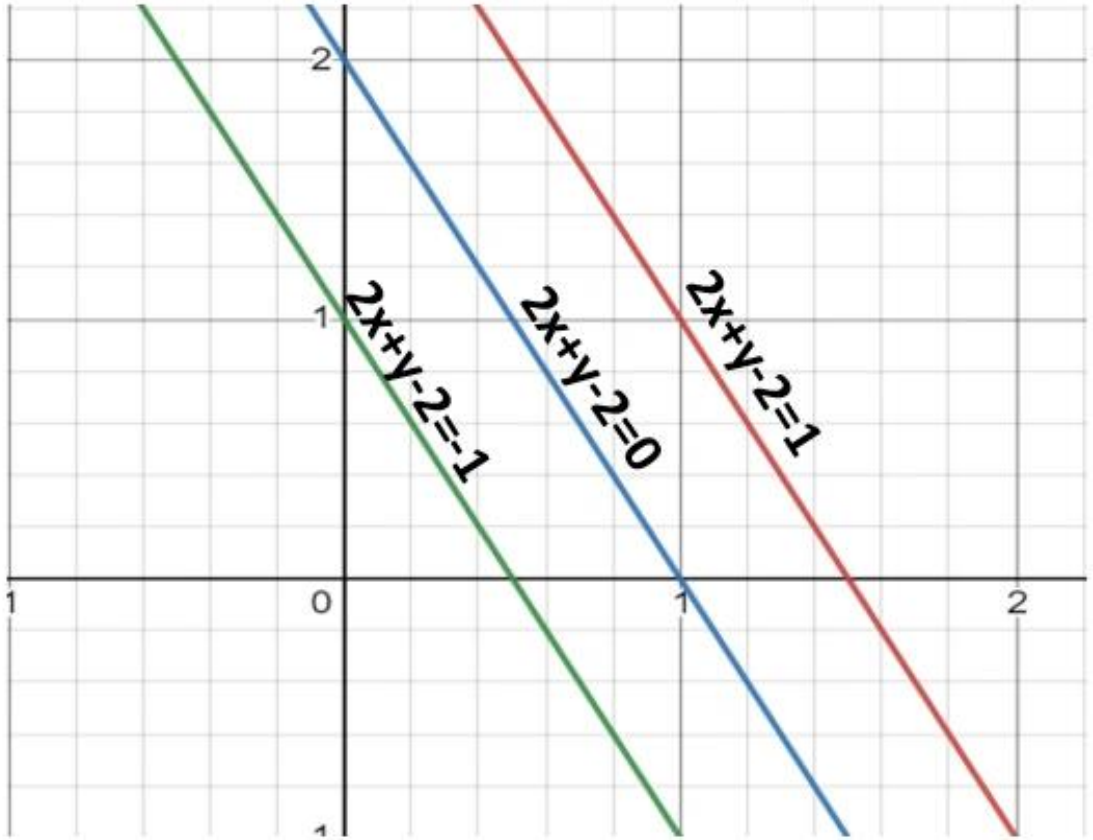
we will take few assumptions that the equation of

L1 is w.x+b=1

L2 it is w.x+b=-1

Suppose the equation of our hyperplane is 2x+y=2:





https://www.desmos.com/calculator/dvjo3vacyp

If you multiply these equations by 10, we will see that the parallel line (red and green) gets closer to our hyperplane.

link

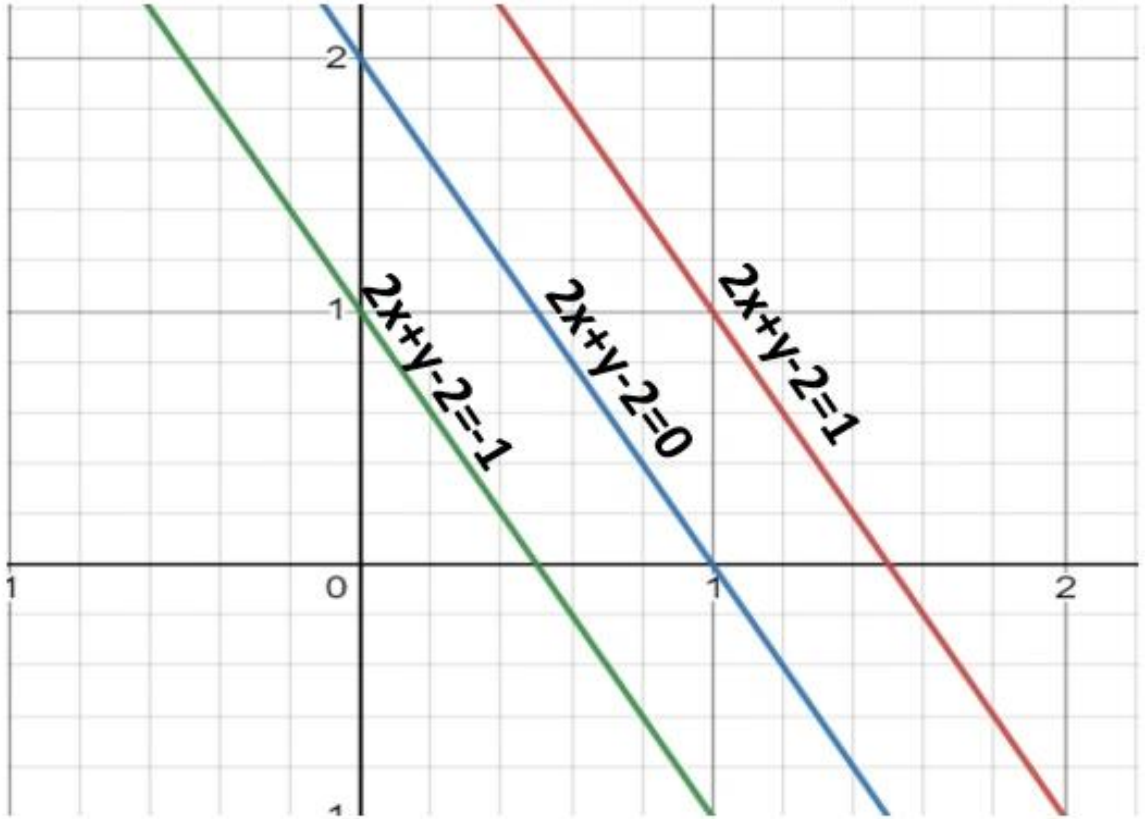
(https://www.desmos.com/calculator/dvjo3vacyp)

We also observe that if we divide this equation by 10 then these parallel lines get bigger.

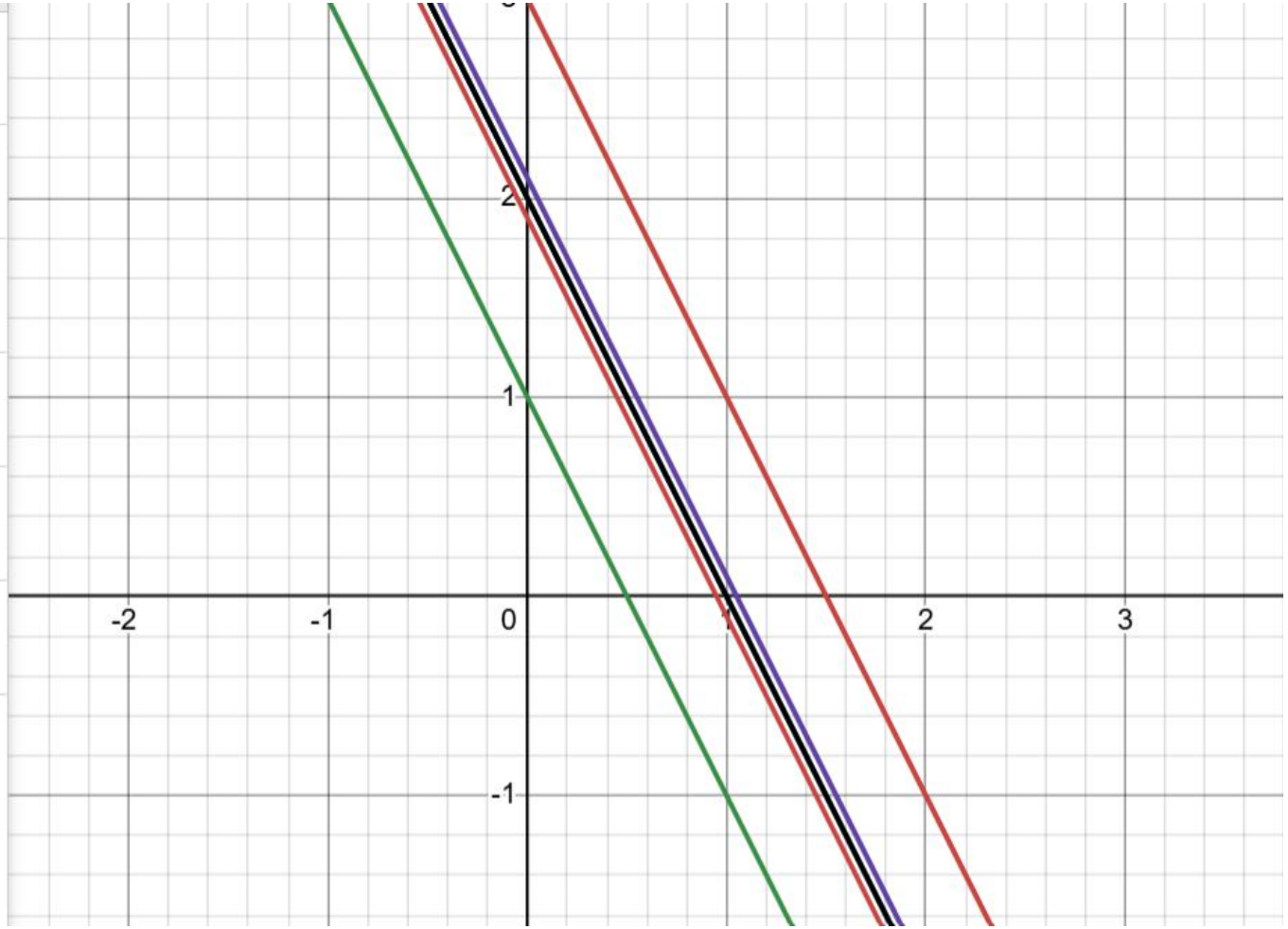
link

(https://www.desmos.com/calculator/15dbwehq9g).

- The parallel lines depend on (w,b) of our hyperplane, if we multiply the equation of hyperplane with a factor greater than 1 then the parallel lines will shrink and if we multiply with a factor less than 1, they expand.



1. $2x + y - 2 = 1$
2. $2x + y - 2 = 0$
3. $2x + y - 2 = -1$
4. $20x + 10y - 20 = 1$
5. $20x + 10y - 20 = 0$
6. $20x + 10y - 20 = -1$
7.



5 $20x + 10y - 20 = 1$

6 $20x + 10y - 20 = 0$

7 $20x + 10y - 20 = -1$

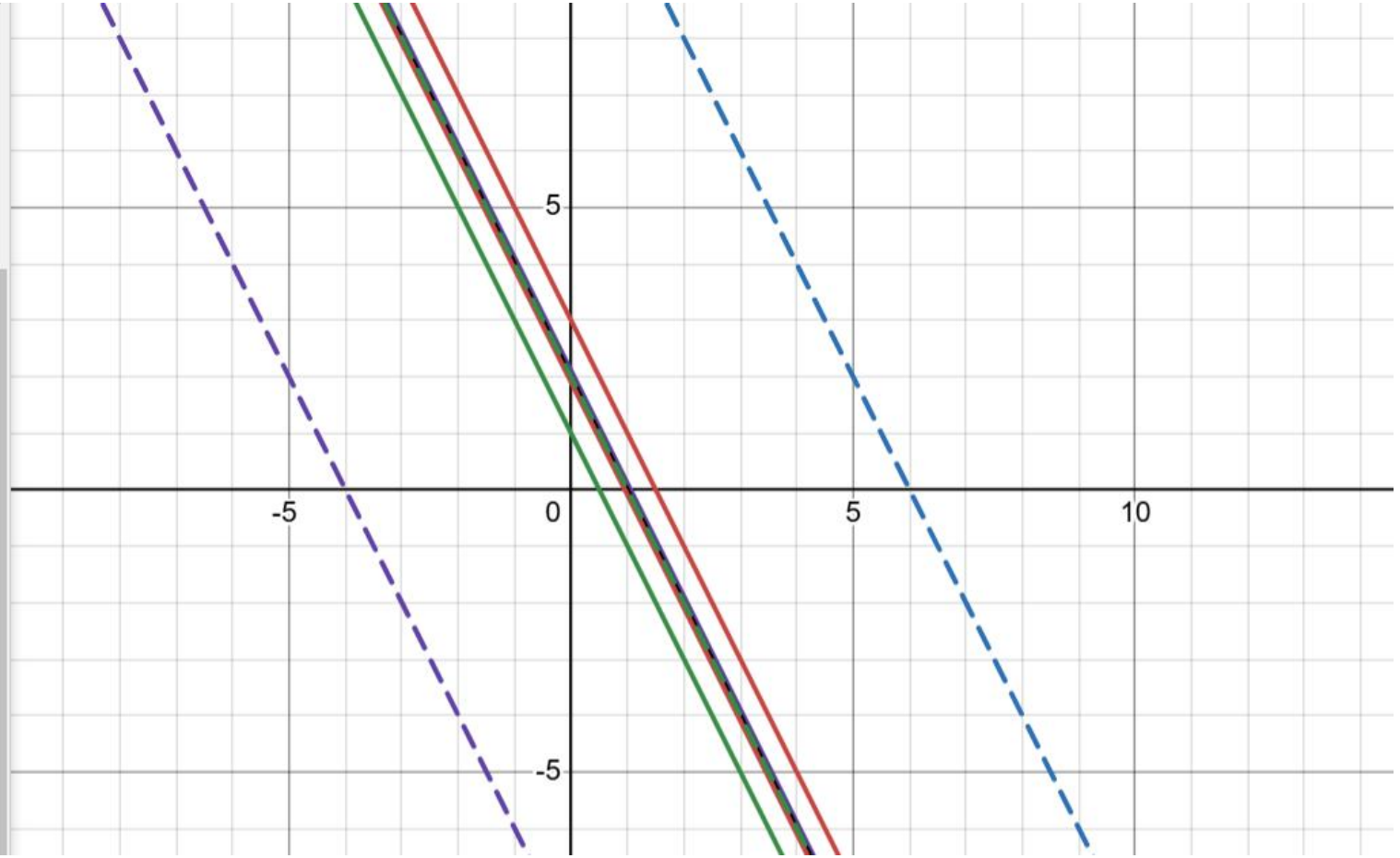
8

9 $0.2x + 0.1y - 0.2 = 1$

10 $0.2x + 0.1y - 0.2 = 0$

11 $0.2x + 0.1y - 0.2 = -1$

12



We can now say that these lines will move as **we do changes in (w,b)** and this is how this gets optimized.

But what is the optimization function?

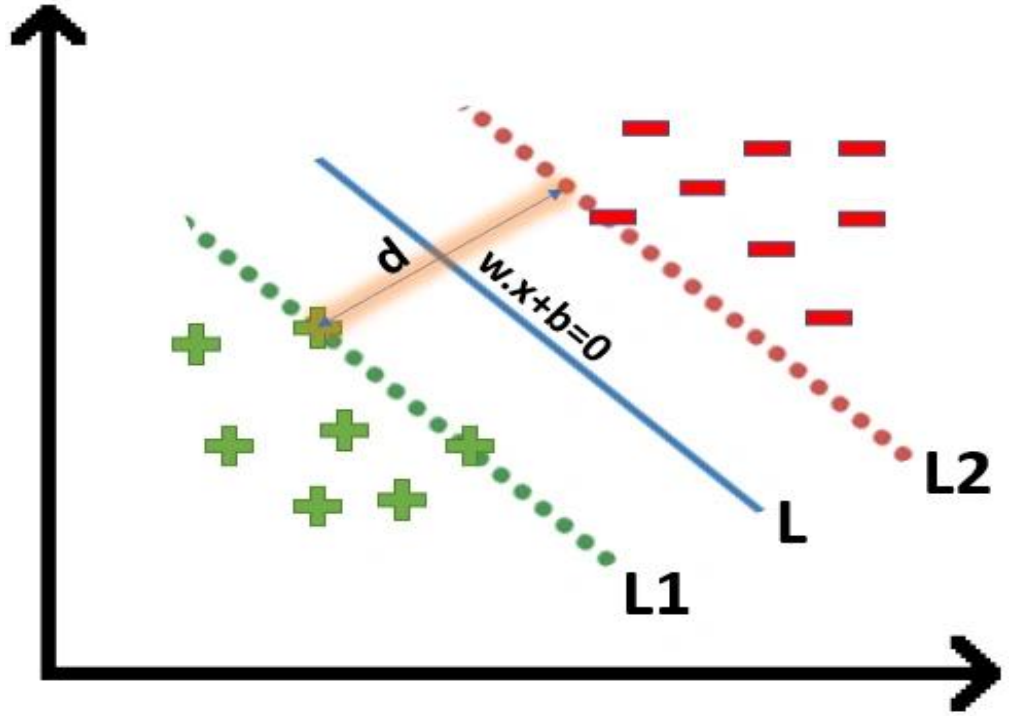
There are few constraints for this distance (d)

We'll calculate the **distance (d)** in such a way that no positive or negative point can cross the margin line".

Let's write these constraints mathematically:

$$For\ all\ the\ Red\ points\ \vec{w}.\vec{X}+b \leq -1$$

$$For\ all\ the\ Green\ points\ \vec{w}.\vec{X}+b \geq 1$$



Rather than taking 2 constraints forward, we'll now try to simplify these two constraints into 1.

We assume that **negative classes have y=-1 and positive classes have y=1**.

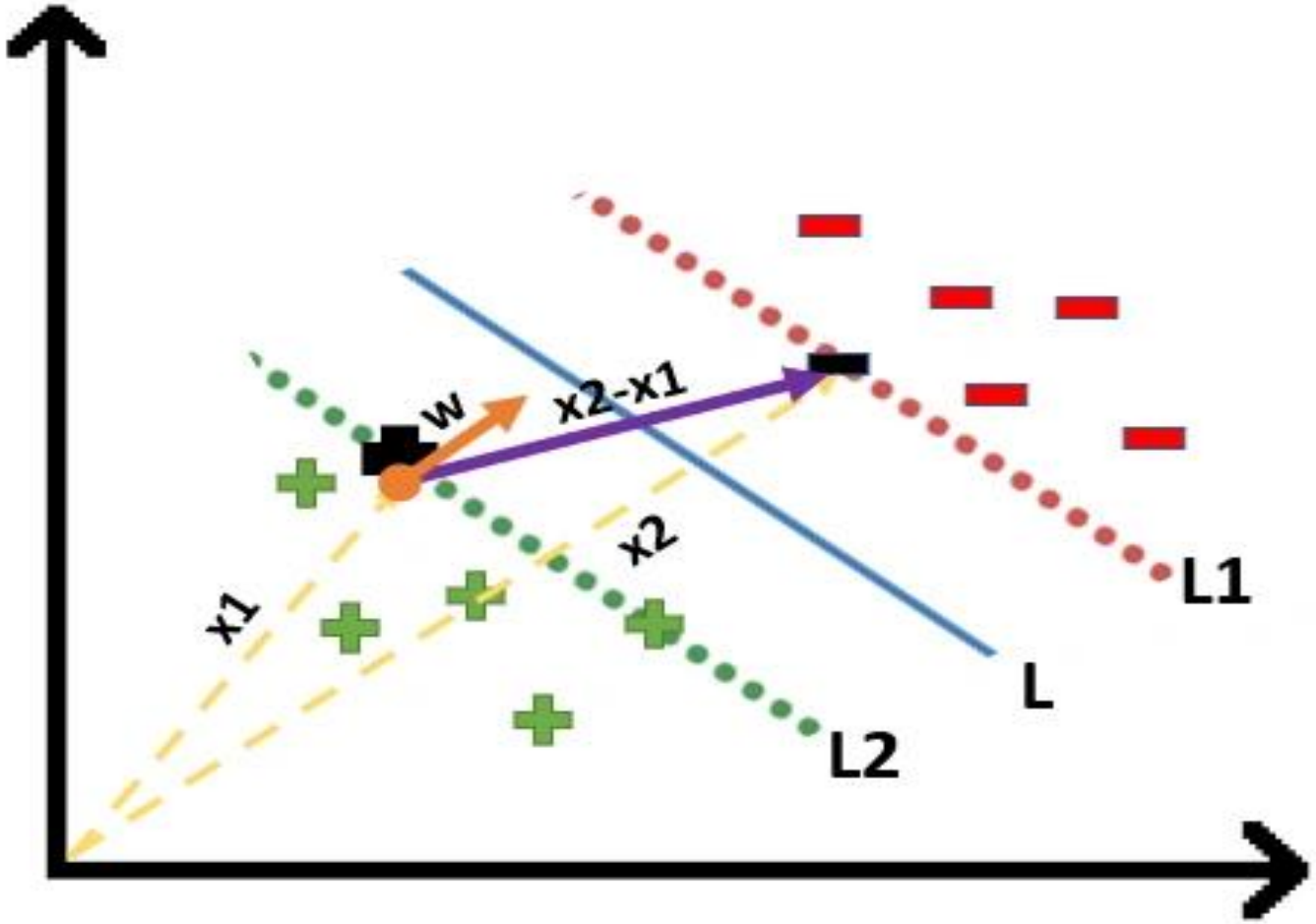
We can say that for every point to be correctly classified this condition should always be **true:**

$$y_i(\vec{w}.\vec{X}+b) \geq 1$$

Maximize (d) such that this constraint holds true

$$y_i(\vec{w}.\vec{X}+b) \geq 1$$

- Suppose a green point is correctly classified that means it will follow w.x+b>=1, if we multiply this with y=1 we get this same equation mentioned above.
- Similarly, if we do this with a red point with y=-1 we will again get this equation

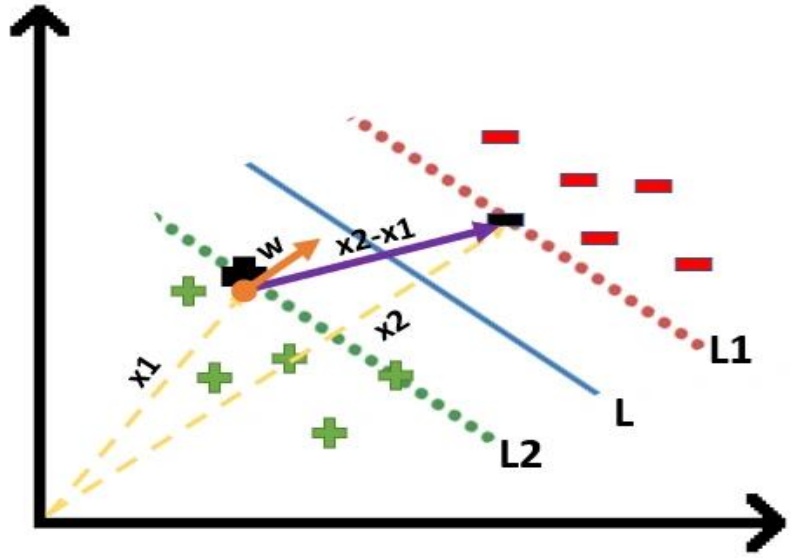


- We will take 2 support vectors, 1 from the negative class and 2nd from the positive class.
- The distance between these two vectors x1 and x2 will be (x2-x1) vector.
- What we need is, the shortest distance between these two points which can be found using a trick we used in the dot product.
- We take a vector ***'w'*** perpendicular to the hyperplane and then find the projection of **(x2-x1) vector on 'w**

Projection of a Vector on Another Vector Using Dot Product

$$\Rightarrow (x2 - x1) . \frac{\vec{w}}{\|w\|}$$

$$\Rightarrow \frac{x2 . \vec{w} - x1 . \vec{w}}{\|w\|} \quad ----(1)$$



This perpendicular vector should be a unit vector then only this will work. We already know To make this 'w' a unit vector we divide this with the norm of 'w'.

Since x2 and x1 are support vectors and they lie on the hyperplane, hence they will follow : **yi* (w.x+b)=1**

so we can write it as:

$$\text{for positive point } y = 1$$

$$\Rightarrow 1 \times (\vec{w}.x1 + b) = 1$$

$$\Rightarrow \vec{w}.x1 = 1 - b \quad -----(2)$$

$$\text{Similarly for negative point } y = -1$$

$$\Rightarrow -1 \times \left(\vec{w}.x2 + b\right) = 1$$

$$\Rightarrow \vec{w}.x2 = -b - 1 \quad -----(3)$$

Putting equations (2) and (3) in equation (1) we get:

$$\Rightarrow (x2 - x1).\frac{\vec{w}}{\|w\|}$$

$$\Rightarrow \frac{x2.\vec{w} - x1.\vec{w}}{\|w\|} \quad -----(1)$$

Similarly for negative point $y = -1$

$$\Rightarrow -1 \times \left(\vec{w}.x2 + b\right) = 1$$

$$\Rightarrow \vec{w}.x2 = -b - 1 \quad ------(3)$$

for positive point $y = 1$

$$\Rightarrow 1 \times (\vec{w}.x1 + b) = 1$$

$$\Rightarrow \vec{w}.x1 = 1 - b \quad -----(2)$$

$$\Rightarrow \frac{(1-b)-(-b-1)}{\|w\|}$$

$$\Rightarrow \frac{1-b+b+1}{\|w\|} = \frac{2}{\|w\|} = d$$

Hence the equation which we have to maximize is:

$$\text{argmax}(w^*, b^*) \ \frac{2}{\|w\|} \text{ such that } y_i\left(\vec{w}.\vec{X} + b\right) \geq 1$$

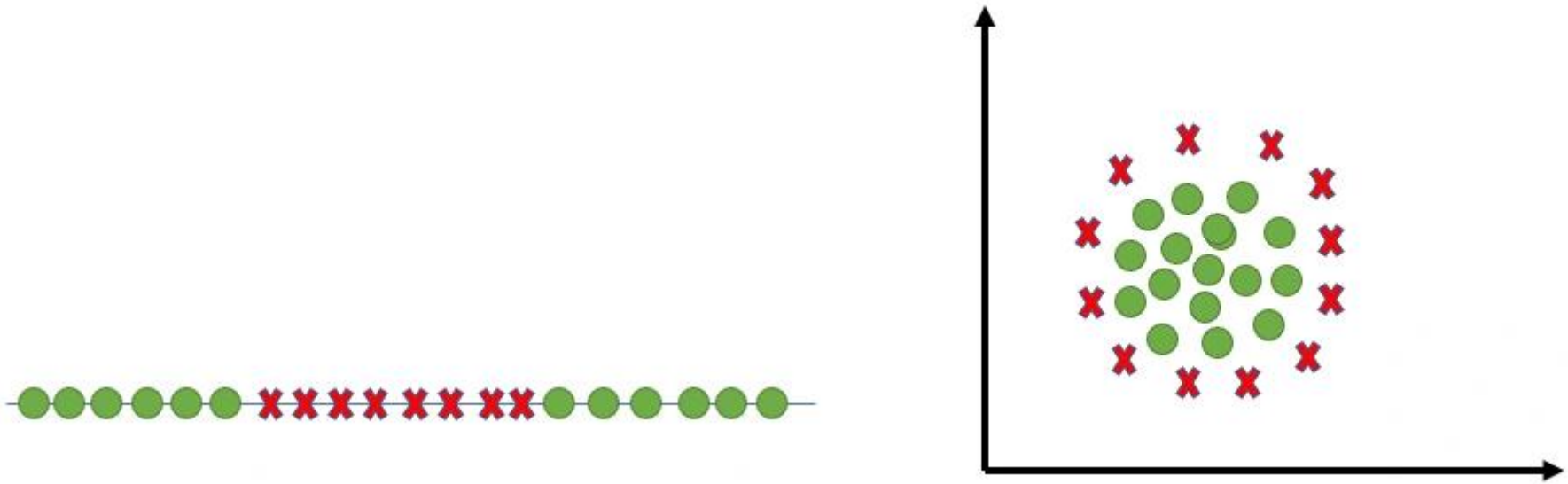
We don’t find this type of perfectly linearly separable data in the industry, there is hardly any case we get this type of data and hence we fail to use this condition we proved here.

The type of problem which we just studied is called **Hard Margin SVM** .

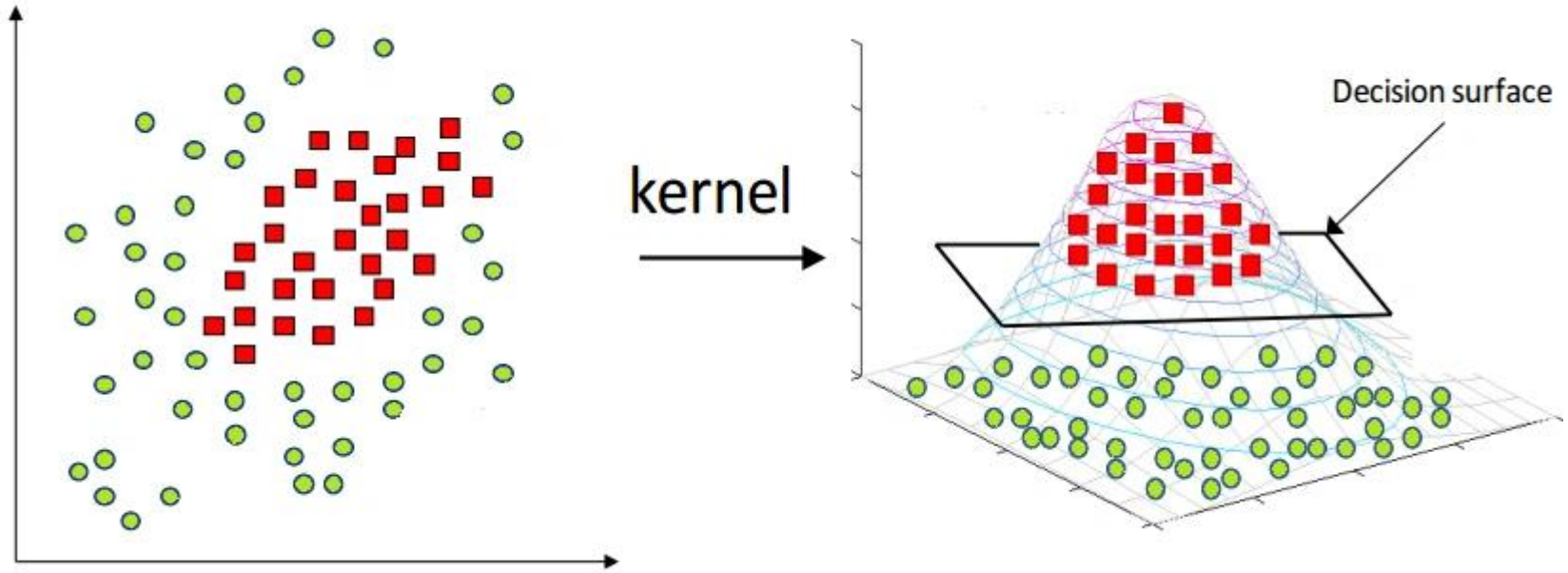
Soft margin which is similar to this but there are few more interesting tricks we use in **Soft Margin SVM.**

Kernels in Support Vector Machine

The most interesting feature of SVM is that it can even work with a non-linear dataset and for this, we use “Kernel Trick” which makes it easier to classifies the points



Kernels in Support Vector Machine



Different Kernel Functions

- Polynomial Kernel
- Sigmoid Kernel
- RBF Kernel
- Bessel function kernel. etc

The **kernel function allows SVM** to solve both linear and non-linear problems by mapping data into a **higher-dimensional space** where a hyperplane can separate the data.

Linear Kernel

The simplest kernel that assumes data is linearly separable. It computes the dot product between two vectors.

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

Example:

For two vectors $x = [1, 2]$ and $y = [3, 4]$,

$$K(x, y) = (1 \times 3) + (2 \times 4) = 3 + 8 = 11$$

Polynomial Kernel

Used for data that is not linearly separable. It introduces non-linearity by raising the dot product to a power.

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

c is a constant (controls influence of higher-order terms)

d is the degree of the polynomial.

Example:

For $x = [1, 2]$, $y = [3, 4]$, $c = 1$, and $d = 2$:

$$K(x, y) = ((1 \times 3) + (2 \times 4) + 1)^2 = (3 + 8 + 1)^2 = 12^2 = 144$$

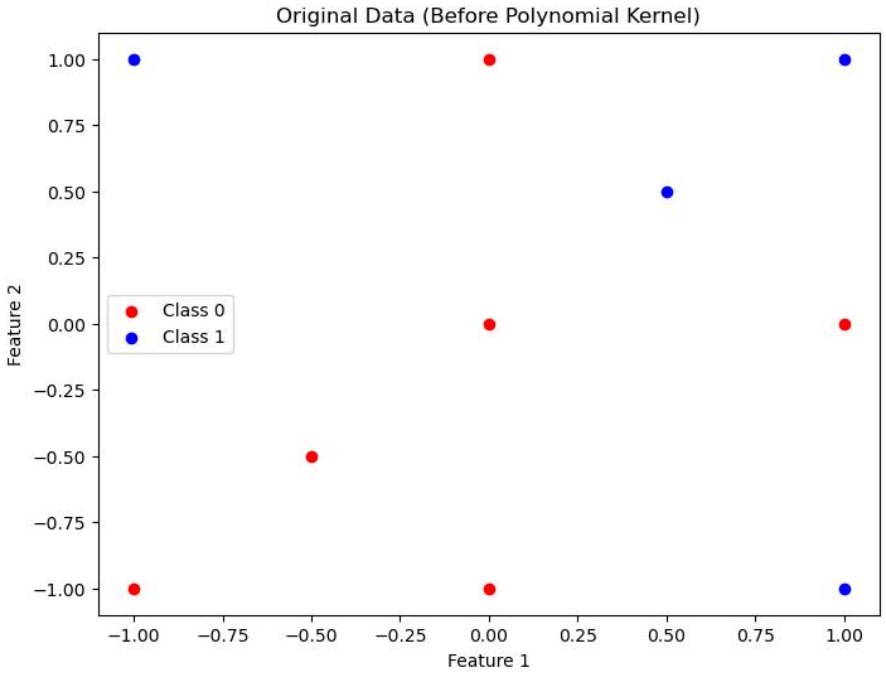
```
X = [
    [0, 0], [0, 1], [1, 0], [1, 1], [0.5, 0.5],
    [1, -1], [-1, 1], [-1, -1], [-0.5, -0.5], [0, -1]
]
y = [0, 0, 0, 1, 1, 1, 1, 0, 0, 0]
```

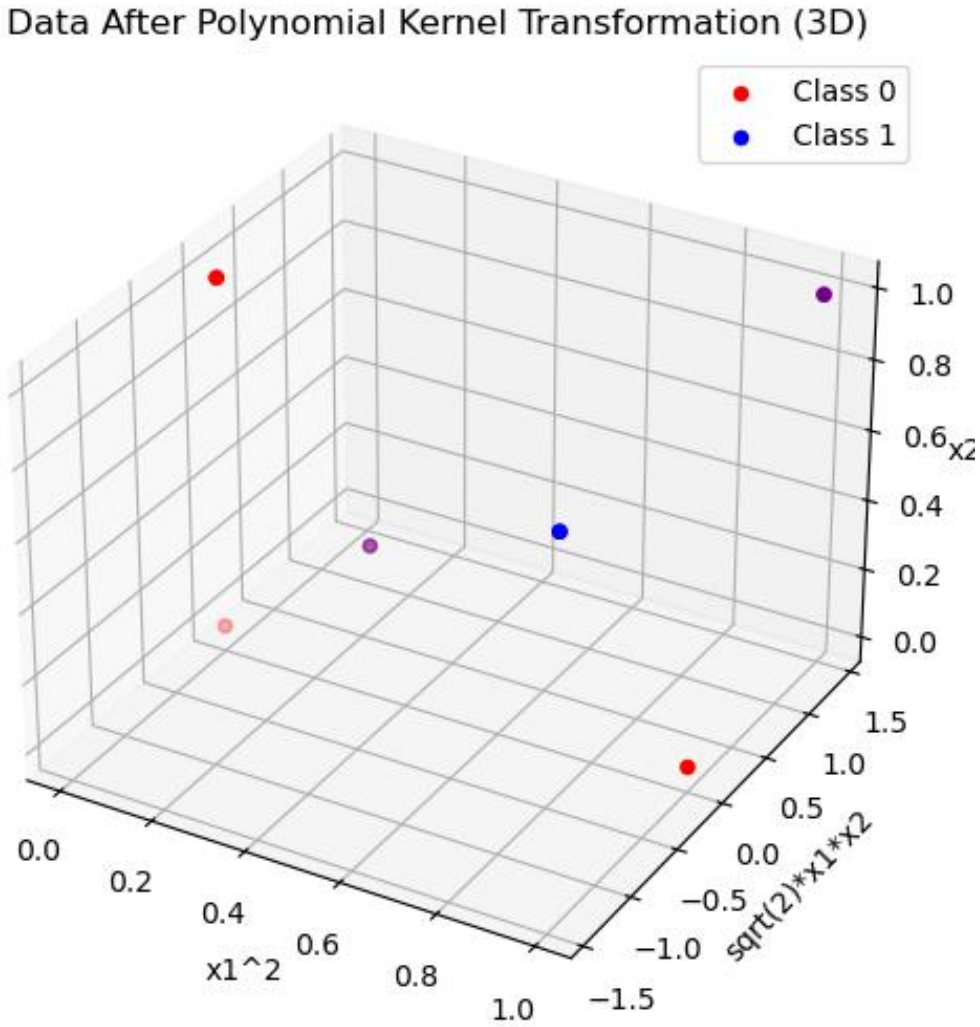
A **polynomial kernel transformation** of degree 2 maps each point $[x_1, x_2]$ in 2D space to a higher-dimensional 3D space:

$$\phi(x_1, x_2) = [x_1^2, \sqrt{2}x_1x_2, x_2^2]$$

Original (x1, x2)	-->	Transformed [x1^2, √2 * x1 * x2, x2^2]
[0, 0]	-->	[0, 0, 0]
[0, 1]	-->	[0, 0, 1]
[1, 0]	-->	[1, 0, 0]
[1, 1]	-->	[1, √2, 1]
[0.5, 0.5]	-->	[0.25, √2 * 0.25, 0.25]
[1, -1]	-->	[1, -√2, 1]
[-1, 1]	-->	[1, -√2, 1]
[-1, -1]	-->	[1, √2, 1]
[-0.5, -0.5]	-->	[0.25, √2 * 0.25, 0.25]
[0, -1]	-->	[0, 0, 1]

```
[[0.    0.    0.  ]
 [0.    0.    1.  ]
 [1.    0.    0.  ]
 [1.    1.41  1.  ]
 [0.25  0.71  0.25]
 [1.   -1.41  1.  ]
 [1.   -1.41  1.  ]
 [1.    1.41  1.  ]
 [0.25  0.71  0.25]
 [0.    0.    1.  ]]
```


Original Data (Before Polynomial Kernel)
Class 0
Class 1
Feature 1
Feature 2
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00


Data After Polynomial Kernel Transformation (3D)
Class 0
Class 1
x1^2
x2
sqrt(2)*x1*x2
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5

Radial Basis Function (RBF) Kernel (Gaussian Kernel)

The most commonly used kernel for non-linear problems. It maps data into infinite-dimensional space.

$$K(x, y) = \exp\left(-\frac{\|x - y\|^2}{2\sigma^2}\right)$$

Where:

- $\|x - y\|^2$ is the squared Euclidean distance between vectors x and y,
- σ is a parameter controlling the spread.

Example:

For $x = [1, 2]$, $y = [3, 4]$, and $\sigma = 1$:

$$K(x, y) = \exp\left(-\frac{(1-3)^2 + (2-4)^2}{2 \times 1^2}\right) = \exp\left(-\frac{4+4}{2}\right) = \exp(-4) \approx 0.018$$

Handwritten digit recognition (e.g., MNIST dataset).

Sigmoid Kernel

This is derived from the sigmoid activation function used in neural networks.

Mathematical Formula:

$$K(x, y) = \tanh(\alpha x^T y + c)$$

Where:

- α is a slope parameter,
- c is a constant.

Example:

For $x = [1, 2]$, $y = [3, 4]$, $\alpha = 0.5$, and $c = 1$:

$$K(x, y) = \tanh(0.5 \times ((1 \times 3) + (2 \times 4)) + 1) = \tanh(0.5 \times (3 + 8) + 1) = \tanh(6.5)$$

(The value of $\tanh(6.5)$ is approximately 0.9999.)

Multiclass SVM

Multiclass SVM is an extension of Support Vector Machines (SVMs) designed to handle classification problems with more than two classes.

SVM is inherently a binary classifier, but for multiclass classification, we use strategies like:

One-vs-One (OvO): Builds a classifier for every pair of classes. If there are n classes, it creates $\boldsymbol{n\ (n\ -1)\ /2}$ classifiers.

One-vs-All (OvA): Builds $\boldsymbol{n}$ **classifiers**, each distinguishing one class from all the others.

Training Phase:

- In OvO, each classifier separates one class from another.
- In OvA, each classifier separates one class from the rest.

Prediction Phase:

- For OvO, the class that wins the most pairwise comparisons is predicted.
- For OvA, the class with the highest confidence score is predicted.

Example: Handwriting Digit Classification

Classify handwritten digits (0, 1, 2) using Multiclass SVM.

Dataset:

Sample	Feature 1 (x1)	Feature 2 (x2)	Class
1	1.2	3.4	0
2	2.1	1.5	1
3	3.3	2.9	2

OvA Strategy:

Build 3 classifiers:

- Classifier 1: Distinguishes Class 0 from {1, 2}.
- Classifier 2: Distinguishes Class 1 from {0, 2}.
- Classifier 3: Distinguishes Class 2 from {0, 1}.

Decision Boundaries:

Each classifier generates a hyperplane. For a new data point, the classifier with the maximum confidence (distance from the hyperplane) predicts the class.

Example Prediction:

A new data point has features

(x 1, x 2)=(2.0, 2.5)

Classifier 1 predicts Not Class 0.

Classifier 2 predicts Class 1.

Classifier 3 predicts Not Class 2.

Final Prediction: Class 1.

Solve using One-vs-One Strategy

For n=3 (classes 0, 1, and 2), we need 3 classifiers:

- Classifier 1: Distinguishes between Class 0 and Class 1.
- Classifier 2: Distinguishes between Class 0 and Class 2.
- Classifier 3: Distinguishes between Class 1 and Class 2.

A new data point has features **(x 1, x 2)=(2.0, 2.5)**

Classifier 1 (Class 0 vs Class 1):

Decision boundary separates Class 0 and Class 1.
Prediction: Class 1 (data is closer to Class 1's region).

Classifier 2 (Class 0 vs Class 2):

Decision boundary separates Class 0 and Class 2.
Prediction: Class 2.

Classifier 3 (Class 1 vs Class 2):

Decision boundary separates Class 1 and Class 2
Prediction: Class 1.

Classifier Pair	Predicted Class
0 vs 1	1
0 vs 2	2
1 vs 2	1

Classifier Pair	Predicted Class
0 vs 1	1
0 vs 2	2
1 vs 2	1

Vote Count:

Class 0: 0 votes
Class 1: 2 votes
Class 2: 1 vote

Final Prediction:
The class with the majority vote is Class 1.

Principal Component Analysis (PCA)

- Principal Component Analysis, or PCA, is a fundamental technique in the realm of data analysis and machine learning.
- It plays a pivotal role in reducing the dimensionality of complex datasets, simplifying their interpretation, and enhancing computational efficiency.

What is Principal Component Analysis?

- Principal Component Analysis is a statistical method that transforms high-dimensional data into a lower-dimensional form while preserving the most important information.
- It accomplishes this by identifying new axes, called principal components, along which the data varies the most.
- These components are orthogonal to each other, meaning they are uncorrelated, making them a powerful tool for dimensionality reduction.

The Need for Dimensionality Reduction

Imagine you have a dataset with many features or variables. Each feature contributes to the overall complexity of the data, making it challenging to analyze, visualize, or build models.

- **Computational Complexity:**
- **Overfitting:**
- **Difficulty in Visualization:**
- **Redundancy:** Some features may be highly correlated, meaning they convey similar information. This redundancy can be eliminated without significant loss of information.

This is where PCA comes into play. It helps us reduce the dimensionality of the data while preserving its essential characteristics.

Principal Components:

- PCA identifies new axes (called principal components) in the data.
- These components are **linear combinations of the original features** and are orthogonal (uncorrelated).
- The **first principal component captures the maximum variance**, the second captures the next highest variance (orthogonal to the first), and so on.
- PCA prioritizes features with the highest variance, as variance is often associated with meaningful information.

Steps in PCA

Standardization: Center and scale the data so that all features have a mean of 0 and variance of 1.

Covariance Matrix: Compute the covariance matrix of the data to understand how features are correlated.

Eigenvectors and Eigenvalues: Compute the eigenvectors (directions) and eigenvalues (amount of variance) of the covariance matrix.

Eigenvectors define the new axes, while eigenvalues indicate the importance (variance) captured by each axis.

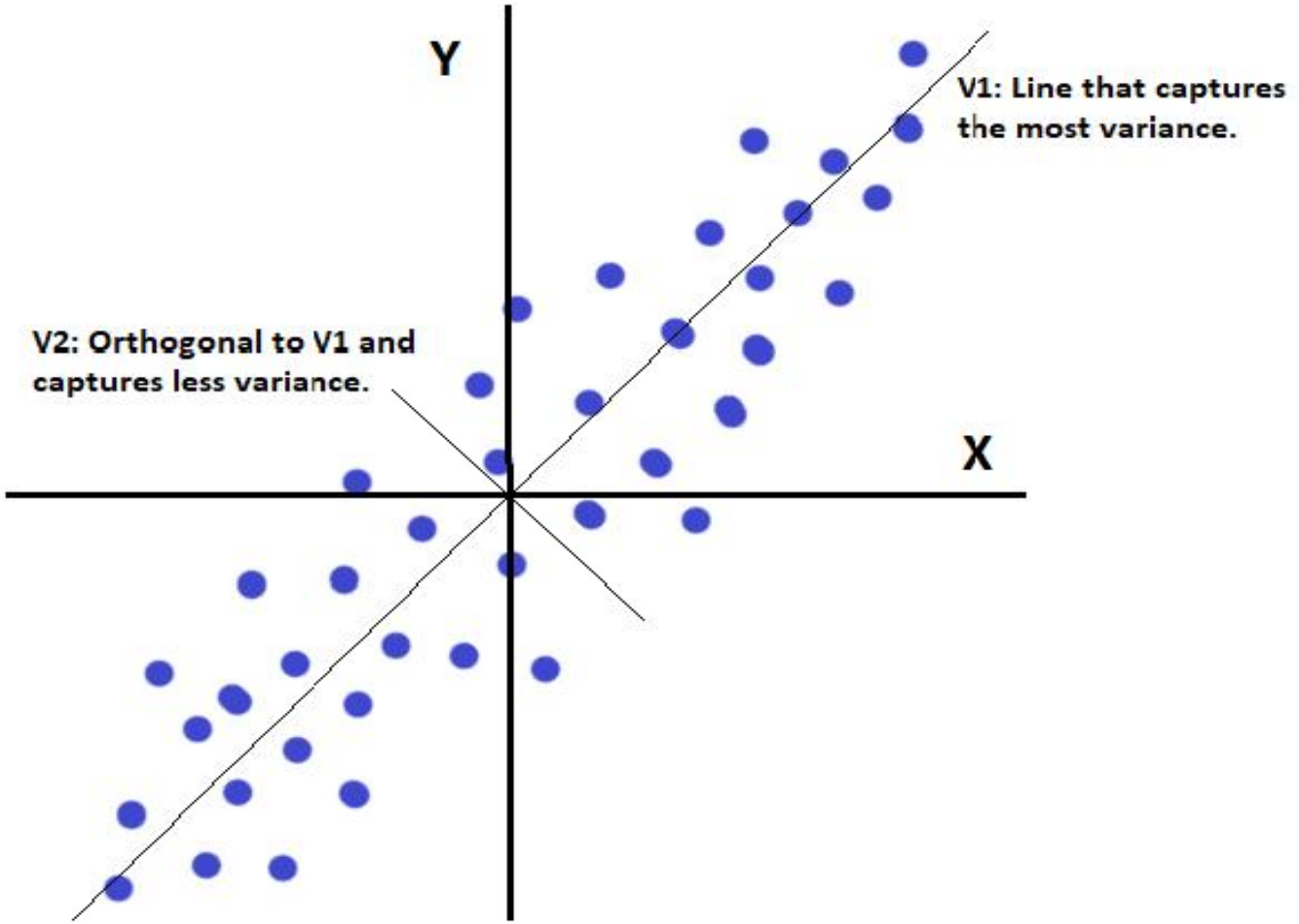
Projection: Project the original data onto the top k principal components (axes with the highest eigenvalues).

Real-World Example

- Imagine a dataset with images of handwritten digits (e.g., the MNIST dataset). Each image has 784 features (28x28 pixels).

- PCA can reduce this high-dimensional data to 2D or 3D while retaining most of the information, enabling easy visualization and faster processing.

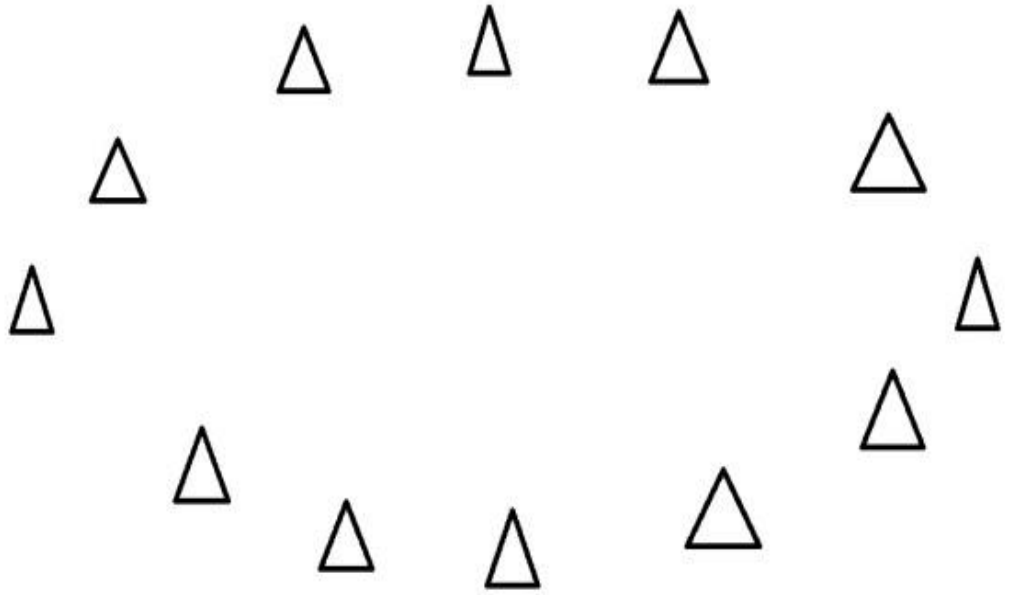
Principal Components



Principal Components

- It is often useful to measure data in terms of its principal components rather than on a normal x-y axis. So what are principal components then?

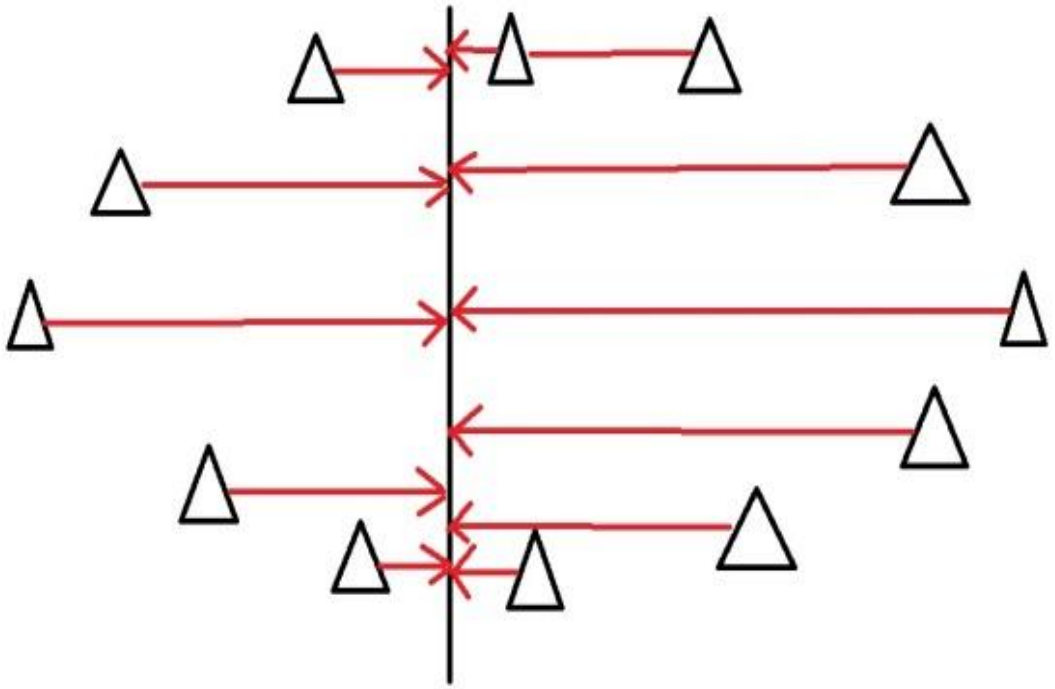
- They’re the underlying structure in the data. They are the directions where there is the most variance, the directions where the data is most spread out.



Principal Components

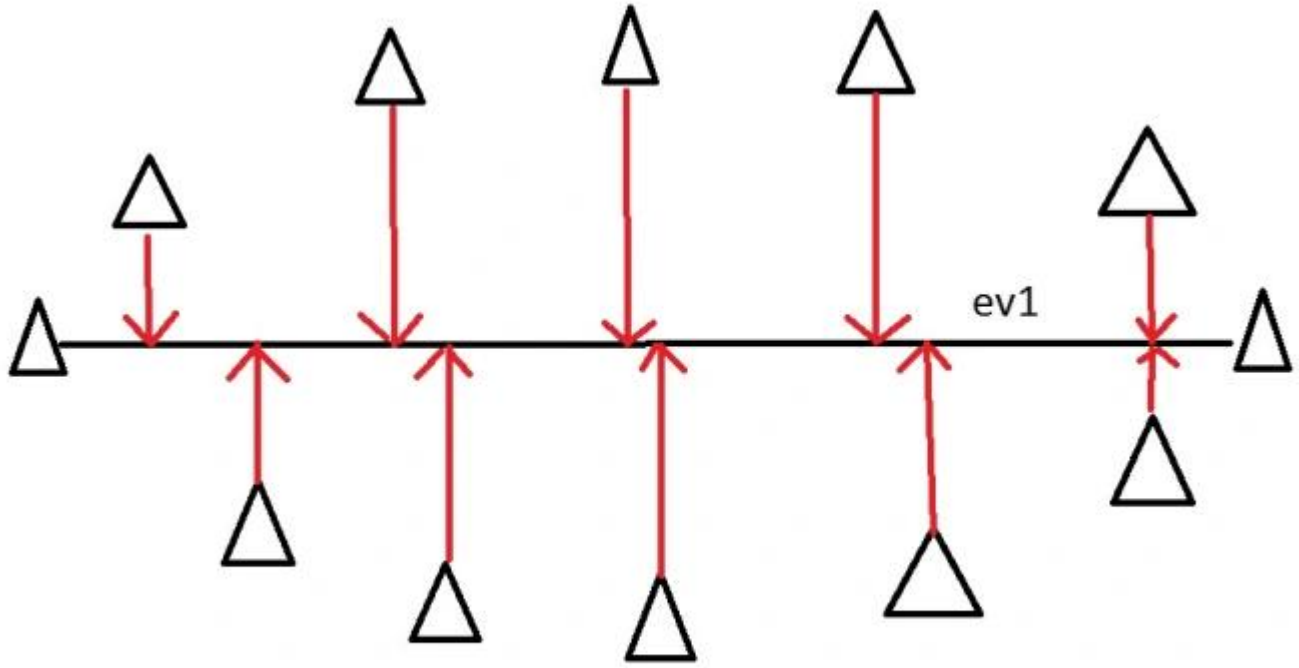
Imagine that the triangles are points of data. To find the direction where there is most variance, find the straight line where the data is most spread out when projected onto it.

A vertical straight line with the points projected on to it will look like this



Principal Components

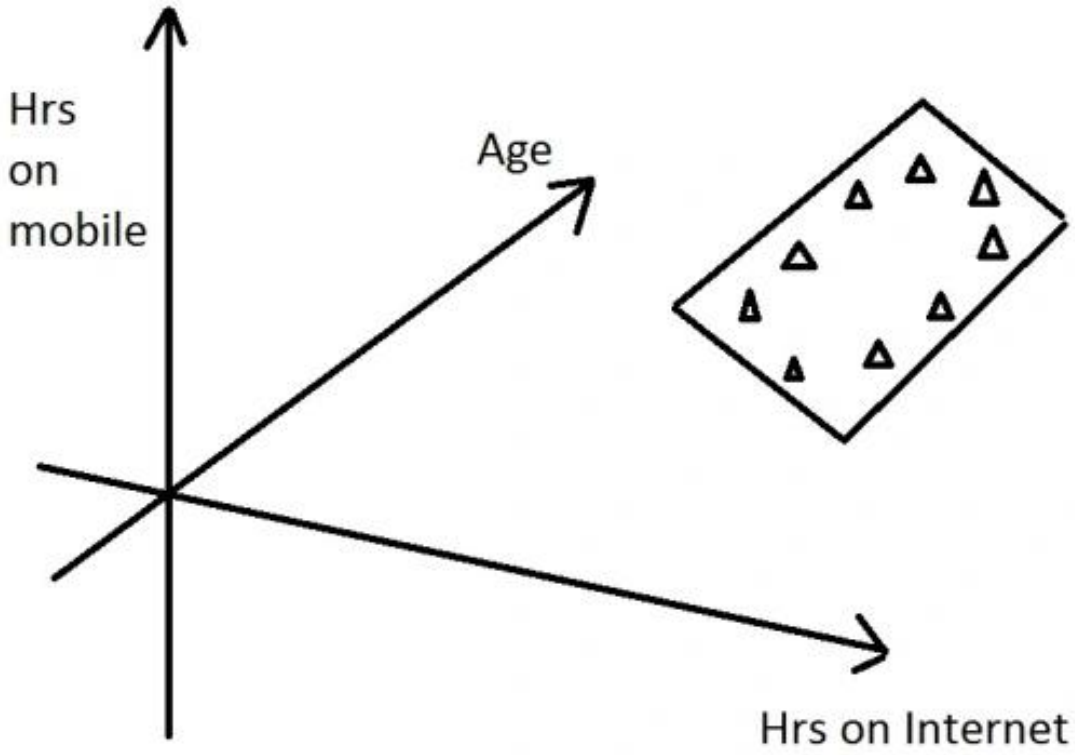
A horizontal line are with lines projected on will look like this



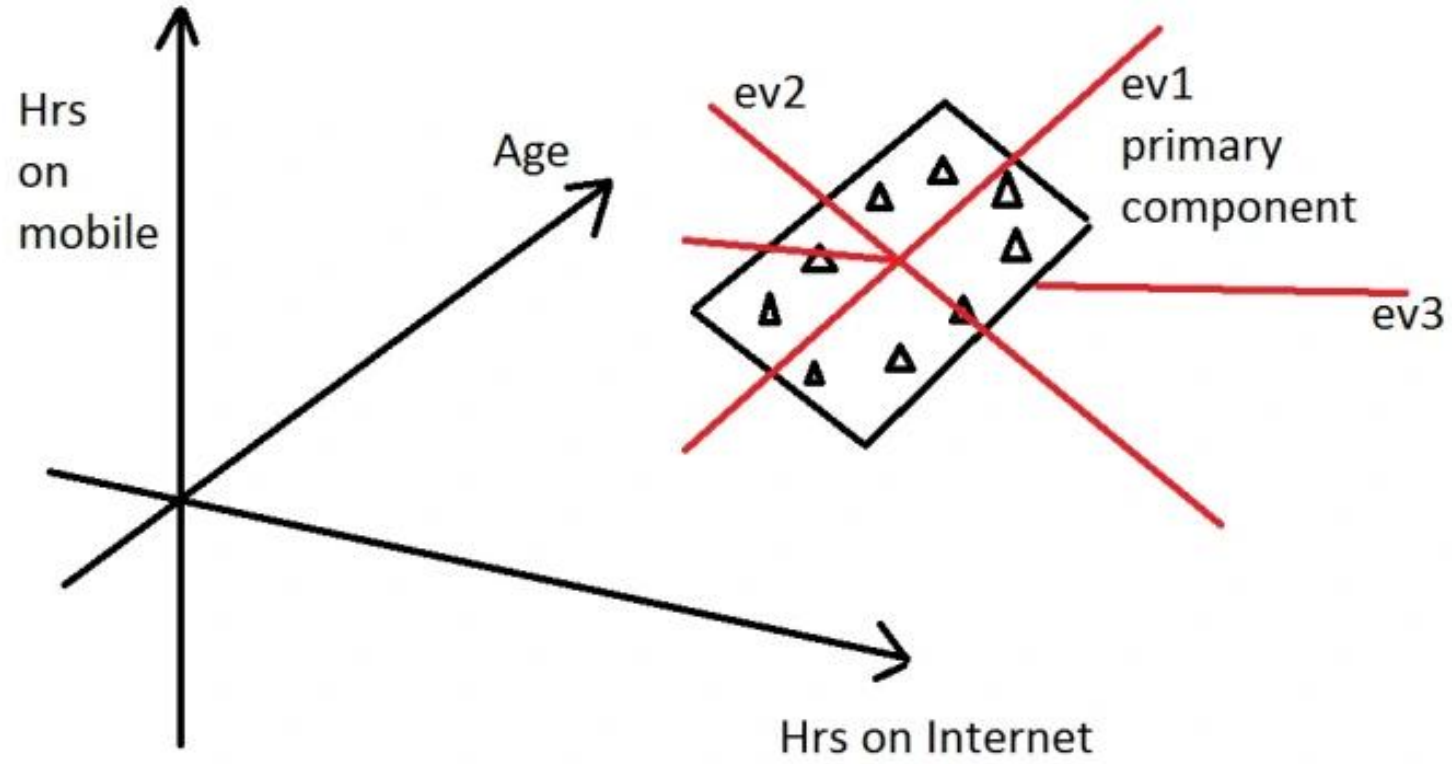
- On this line the data is way more spread out, it has a large variance. In fact there isn't a straight line you can draw that has a larger variance than a horizontal one.

- **A horizontal line is therefore the principal component in this example**

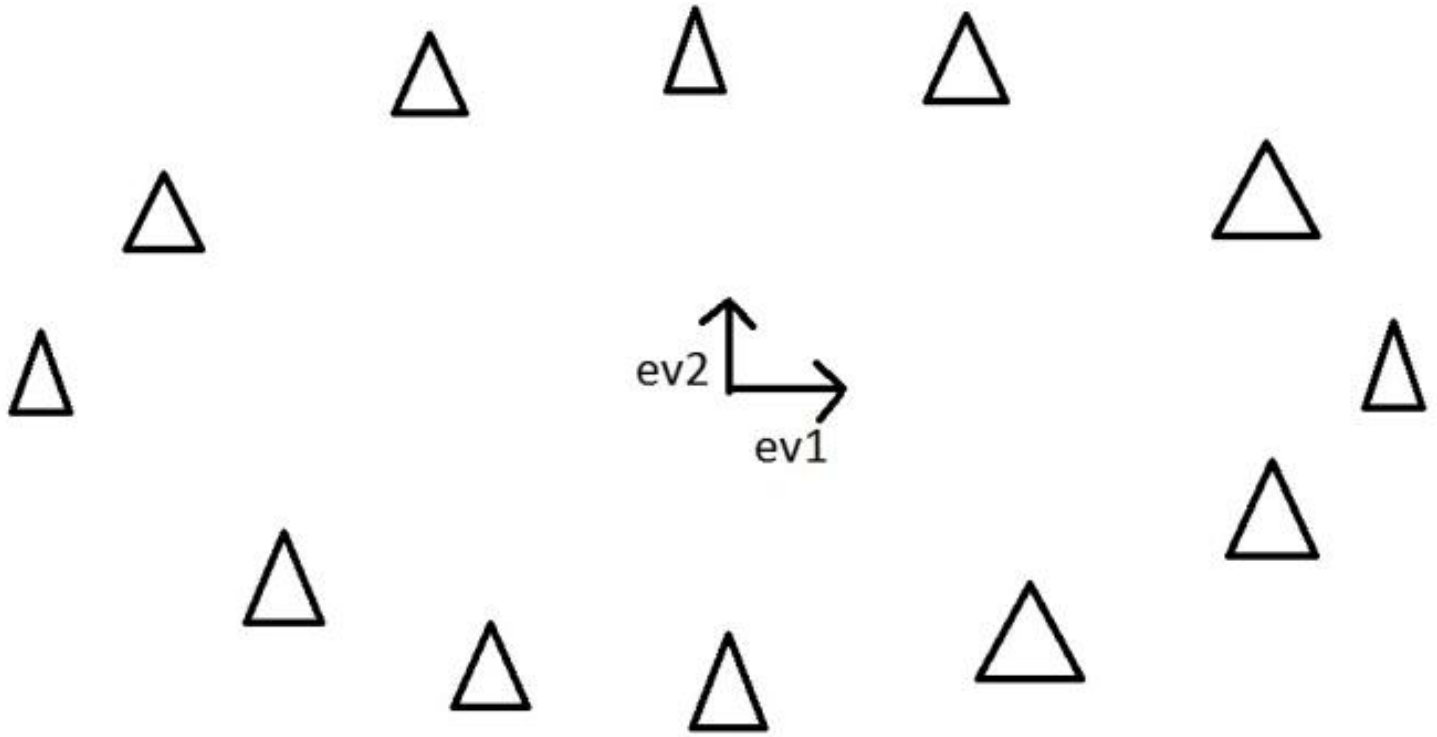
Principal Components



Principal Components



Principal Components



This is dimension reduction. We have reduced the problem from a 3D to a 2D problem, getting rid of a dimension. Reducing dimensions helps to simplify the data and makes it easier to visualise.

Principal Components

Consider a dataset with two features (X1 and X2) for 4 data points:

Data Point	X1	X2
1	2	3
2	3	4
3	4	5
4	5	6

We will apply PCA to reduce this 2-dimensional dataset into 1 dimension.

Principal Components

Step 1: Mean centering the data

We first calculate the mean of each feature:

- Mean of X1 = (2 + 3 + 4 + 5) / 4 = 3.5
- Mean of X2 = (3 + 4 + 5 + 6) / 4 = 4.5

Principal Components

Now, we subtract the mean from each data point to center the data around the origin:

Data Point	X1 (Centered)	X2 (Centered)
1	2 - 3.5 = -1.5	3 - 4.5 = -1.5
2	3 - 3.5 = -0.5	4 - 4.5 = -0.5
3	4 - 3.5 = 0.5	5 - 4.5 = 0.5
4	5 - 3.5 = 1.5	6 - 4.5 = 1.5

Principal Components

So the **centered data matrix** is:

$$X_{\text{centered}} = \begin{bmatrix} -1.5 & -1.5 \\ -0.5 & -0.5 \\ 0.5 & 0.5 \\ 1.5 & 1.5 \end{bmatrix}$$

Transpose of X_{centered}:

$$X_{\text{centered}}^T = \begin{bmatrix} -1.5 & -0.5 & 0.5 & 1.5 \\ -1.5 & -0.5 & 0.5 & 1.5 \end{bmatrix}$$

Principal Components

Step 4: Covariance Matrix Formula

The formula for the covariance matrix is:

$$\text{Cov}(X) = \frac{1}{n-1} X_{\text{centered}}^T X_{\text{centered}}$$

Where n is the number of data points, which in this case is 4, so $n - 1 = 3$.