PART 1

Naive Bayes Classifiers

A probabilistic approach to Machine Learning

Source: GeeksforGeeks

What is Naive Bayes?

Naive Bayes is a collection of classification algorithms based on **Bayes' Theorem**.

It is not a single algorithm but a family of algorithms where all of them share a common principle: every pair of features being classified is independent of each other.

- **Type:** Supervised Learning
- **Category:** Generative Model
- **Key Feature:** Probabilistic

Bayes' Theorem

Bayes' Theorem finds the probability of an event occurring given the probability of another event that has already occurred.

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

This formula allows us to reverse conditional probabilities.

Components of the Theorem

Posterior: P(A|B)

Probability of hypothesis A being true given evidence B.

Likelihood: P(B|A)

Probability of observing evidence B given hypothesis A is true.

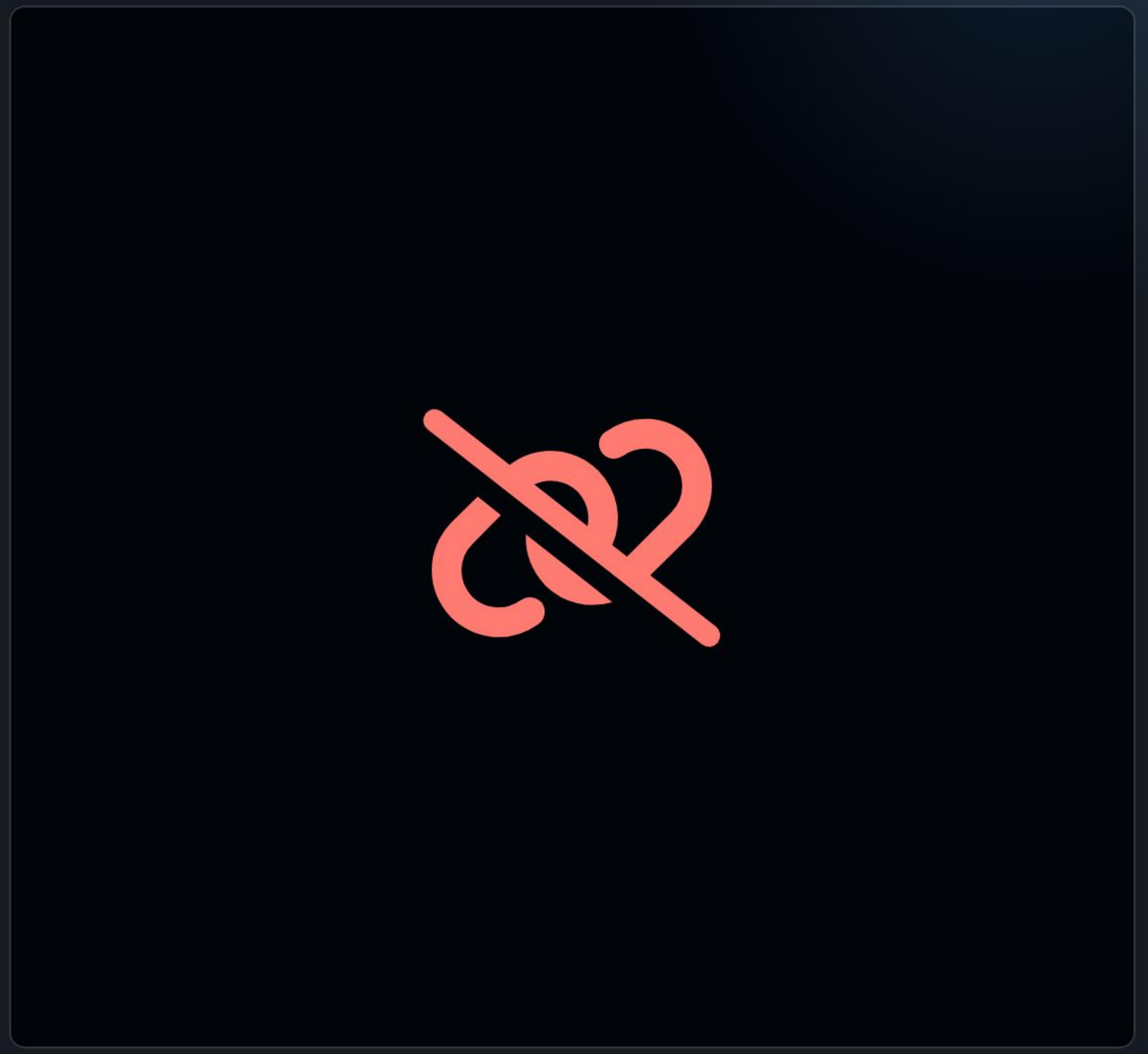
Prior: P(A)

Initial probability of hypothesis A being true (before evidence).

Why is it called "Naive"?

The algorithm is called "Naive" because it makes a strong assumption that the occurrence of a certain feature is **independent** of the occurrence of any other feature.

Example: A fruit is an apple if it is red, round, and about 3 inches. NB assumes "Red" is independent of "Round".



The Classifier Equation

For a feature vector $X = (x_1, x_2, ..., x_n)$ and class y:

$$P(y \mid x_1, \ldots, x_n) \propto P(y) \prod_{i=1}^{n} P(x_i \mid y)$$

We pick the class y that maximizes this probability:

$$y = \text{argmax} P(y) \prod_{i=1}^{n} P(x_i \mid y)$$

Types of Classifiers

Depending on the distribution of the features, we use different types of NB classifiers:

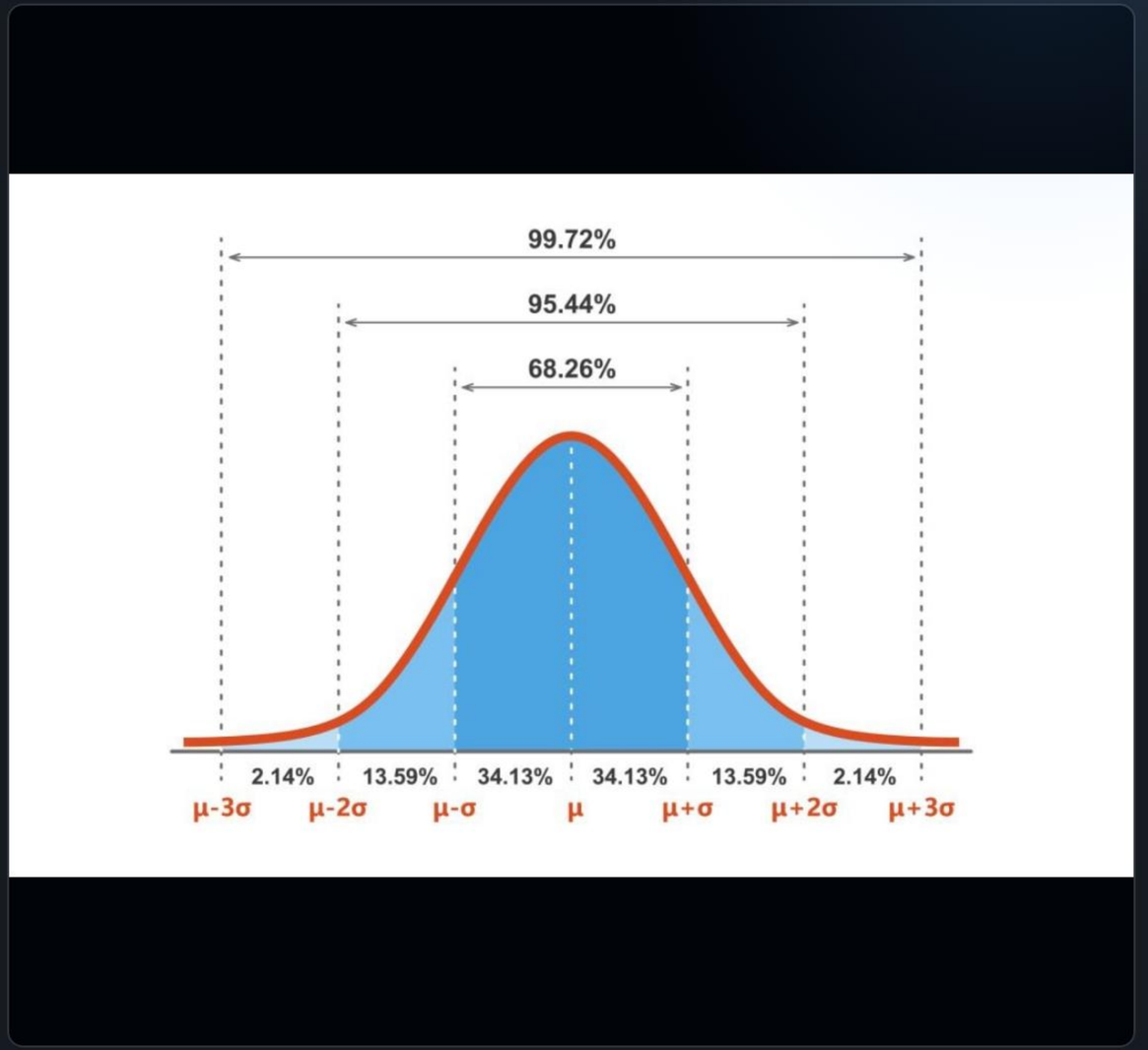
- **Gaussian Naive Bayes:** For continuous values (assumes normal distribution).
- **Multinomial Naive Bayes:** For feature counts (e.g., word frequency).
- **Bernoulli Naive Bayes:** For binary/boolean features.

Gaussian Naive Bayes

Used when features are continuous real values.

We assume the features follow a Gaussian (Normal) Distribution.

Useful for datasets like Iris (sepal length, petal width).



Gaussian Probability Density

To calculate the likelihood $P(x_i|y)$ for continuous data:

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp - \frac{(x_i - \mu_y)^2}{2\sigma_y^2}$$

- μ_y: Mean of feature x_i for class y.
- σ_y: Standard Deviation of feature x_i for class y.

Multinomial Naive Bayes

Designed for data that represents counts or frequencies.

Primary Use Case: Text Classification (e.g., Spam detection, Sentiment Analysis).

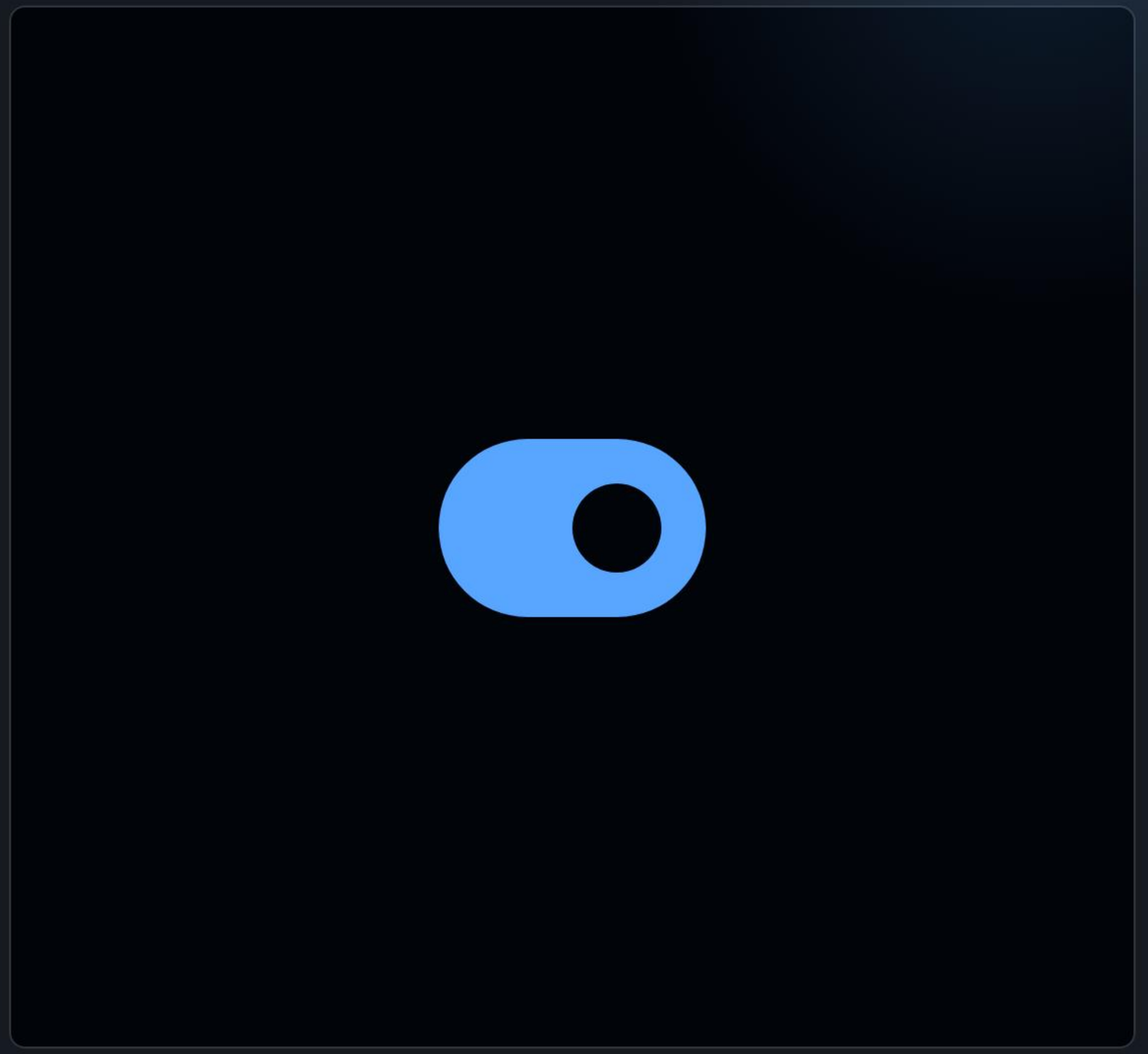
It considers the frequency of words in a document.

Bernoulli Naive Bayes

Similar to Multinomial but for boolean/binary features.

It checks if a word is **present** or **absent** (1 or 0), ignoring frequency.

Useful for short texts or when term existence is more important than count.



Zero Frequency Problem

If a categorical variable has a category in the test data set that was not observed in the training data set, the model assigns a 0 probability.

$$P(\text{word} \mid \text{class}) = 0$$

Since probabilities are multiplied, the entire prediction becomes 0.

Solution: Laplace Smoothing

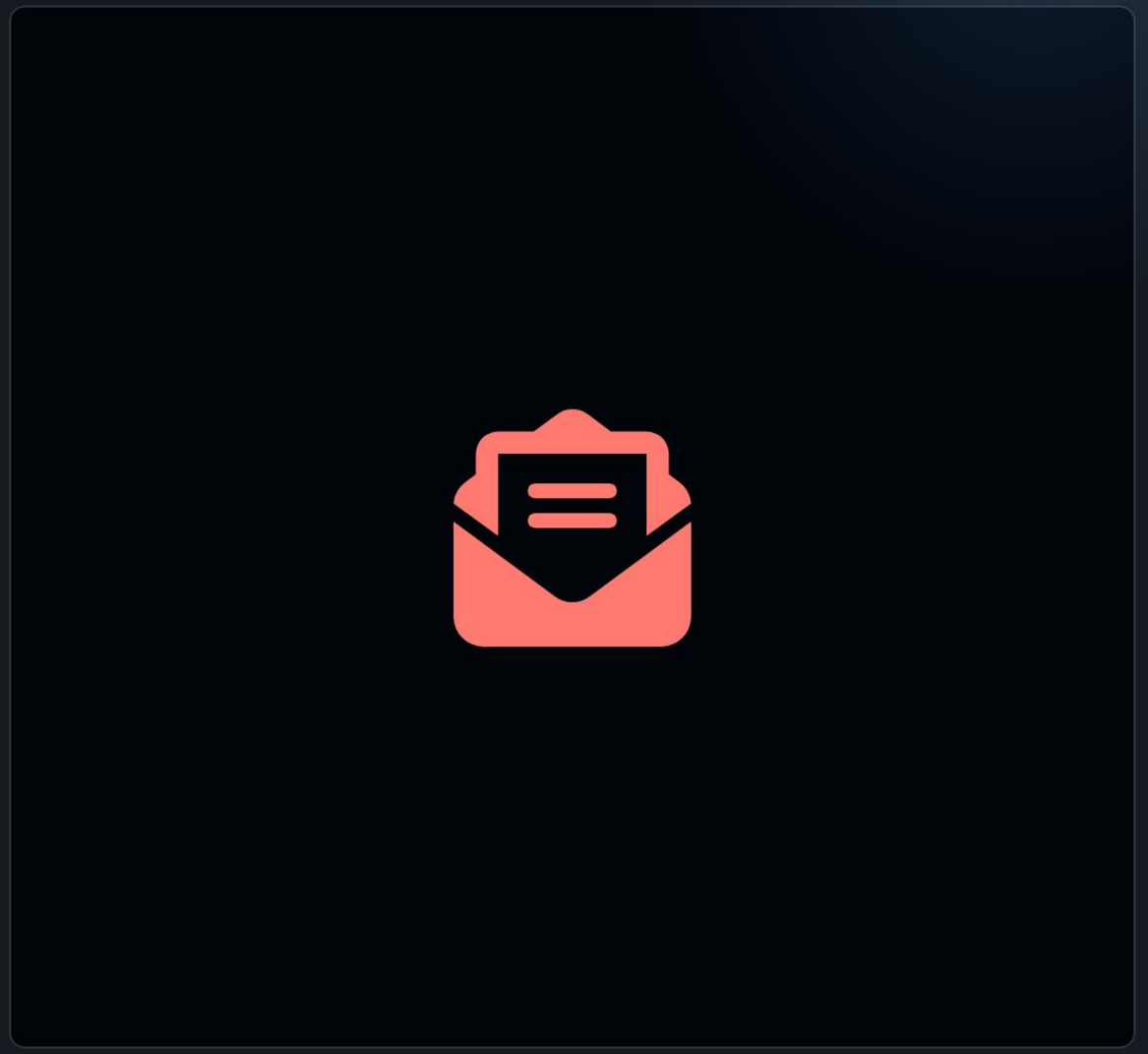
To solve the zero frequency problem, we add a small count (usually 1) to all variable counts.

$$P(x_i \mid y) = \frac{\text{count}_i + \alpha}{N + \alpha \cdot d}$$

- α: Smoothing parameter (usually 1).
- d: Number of distinct features (dimensions).

Use Case: Spam Filtering

1. **Training:** Calculate $P(word|Spam)$ and $P(word|Ham)$ for all words.
2. **Input:** "Win Free Money"
3. **Calc:** $P(Spam) \times P(Win|Spam) \times P(Free|Spam)...$
4. **Result:** Compare Spam vs Ham probability.



Advantages

Speed

Extremely fast for both training and prediction. Linear time complexity.

Small Data

Works well even with less training data compared to other models.

Multi-class

Handles multi-class prediction problems inherently well.

Disadvantages

- **Independence Assumption:** In real life, features are almost never independent. This is the main limitation.
- **Bad Estimator:** While predicted classes are often correct, the probability outputs are not to be taken too seriously (often overconfident).

Python: Setup & Data

```
from sklearn.datasets import load_iris from sklearn.model_selection import train_test_split from
sklearn.naive_bayes import GaussianNB # 1. Load the Iris Dataset X, y = load_iris(return_X_y=True) # 2. Split
into training and testing sets (80/20) X_train, X_test, y_train, y_test = train_test_split( X, y,
test_size=0.2, random_state=42 )
```

X, y: Features and Target arrays.
X_train, X_test: Arrays for model training and validation.

Python: Training GaussianNB

```
# 3. Initialize the Gaussian Naive Bayes Classifier gnb = GaussianNB() # 4. Train the model using the training sets gnb.fit(X_train, y_train) print("Model training complete.")
```

gnb: Instance of the GaussianNB class.
fit(): Method to calculate means and variances for each class.

Python: Making Predictions

```
17  # 5. Make predictions on the test set y_pred = gnb.predict(X_test) # Example prediction for a single sample
18  single_pred = gnb.predict([[5.1, 3.5, 1.4, 0.2]]) print(f"Prediction for sample: {single_pred[0]}")
19
20
21
22
```

y_pred: Array of predicted class labels for X_test.
single_pred: Prediction for a new, unseen flower measurement.

Python: Evaluation

```
23 from sklearn.metrics import accuracy_score # 6. Calculate Accuracy accuracy = accuracy_score(y_test, y_pred)
24 print(f"Gaussian Naive Bayes Accuracy: {accuracy * 100:.2f}%")
25
26
27
28
```

accuracy_score: Utility to compare true labels (y_test) vs predicted labels (y_pred).

PART 2

Support Vector Machines

Finding the Optimal Hyperplane

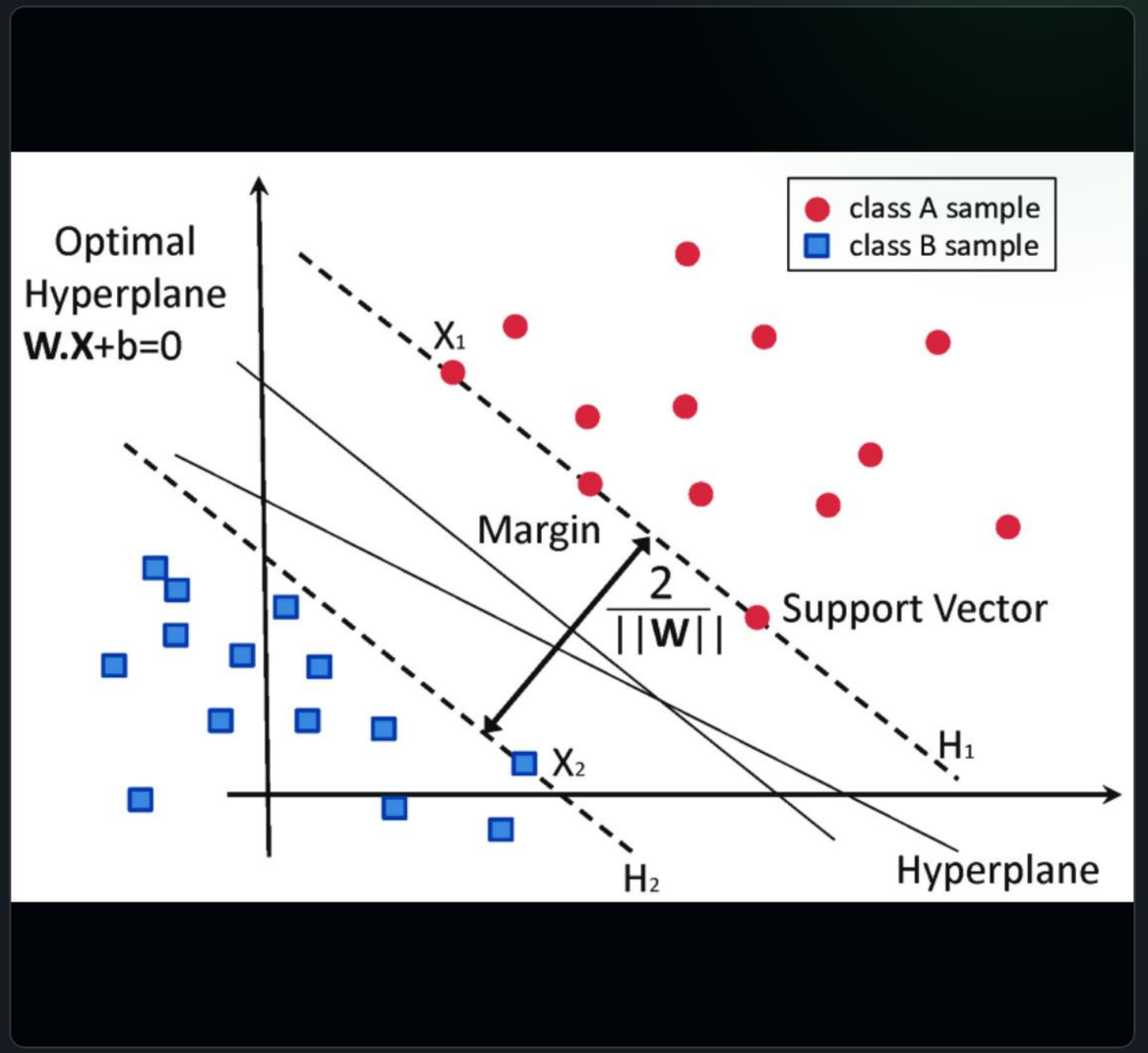
Source: GeeksforGeeks

What is SVM?

Support Vector Machine (SVM) is a supervised learning algorithm used for both classification and regression.

The objective is to find a **hyperplane** in an N-dimensional space that distinctively classifies the data points.

It is a *discriminative* classifier.

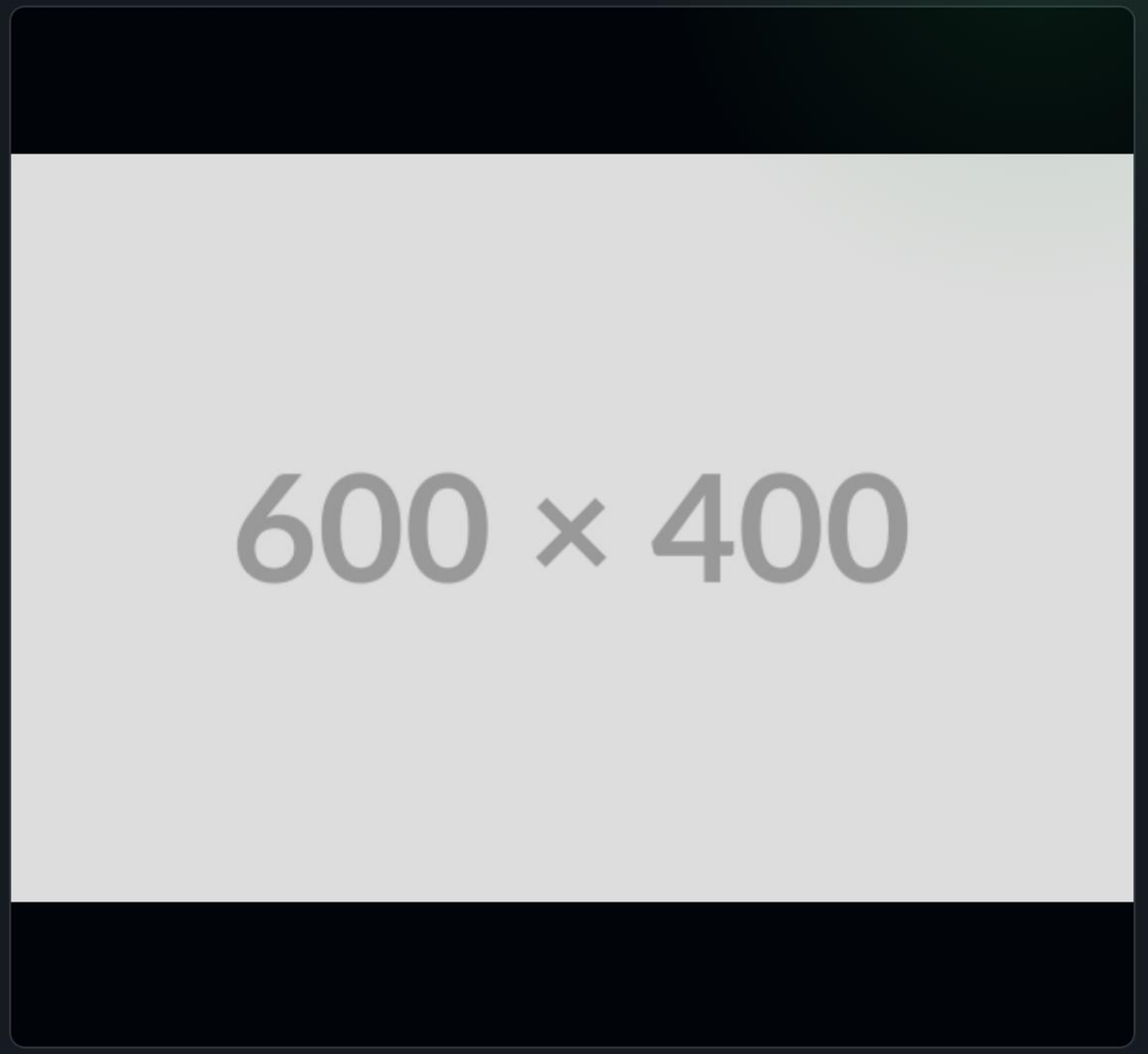


Geometric Intuition

Imagine points of two classes on a piece of paper.

SVM tries to draw a line (or plane in 3D) that separates them.

But not just any line: it wants the line with the **maximum distance** to the nearest points of both classes.



Key Terminology

Hyperplane

The decision boundary. Dimensions depend on features (Line in 2D, Plane in 3D).

Support Vectors

Data points closest to the hyperplane. They determine the line's position.

Margin

The distance between the hyperplane and the support vectors.

Maximizing the Margin

SVM solves an optimization problem.

It aims to maximize the margin distance d.

$$\text{Maximize} \quad \frac{2}{||w||}$$

A larger margin generally implies better generalization and less overfitting.

Hard vs Soft Margin

Hard Margin

Strictly forbids any misclassification. Only works if data is perfectly linearly separable.

Soft Margin

Allows some misclassification to happen to find a broader margin. Used in real-world noisy data.

Cost Function (Hinge Loss)

SVM uses Hinge Loss to penalize misclassifications.

$$c(x, y, f(x)) = \max(0, 1 - y \cdot f(x))$$

If the point is on the correct side of the margin (distance > 1), loss is 0. Otherwise, loss increases linearly.

Regularization Parameter C

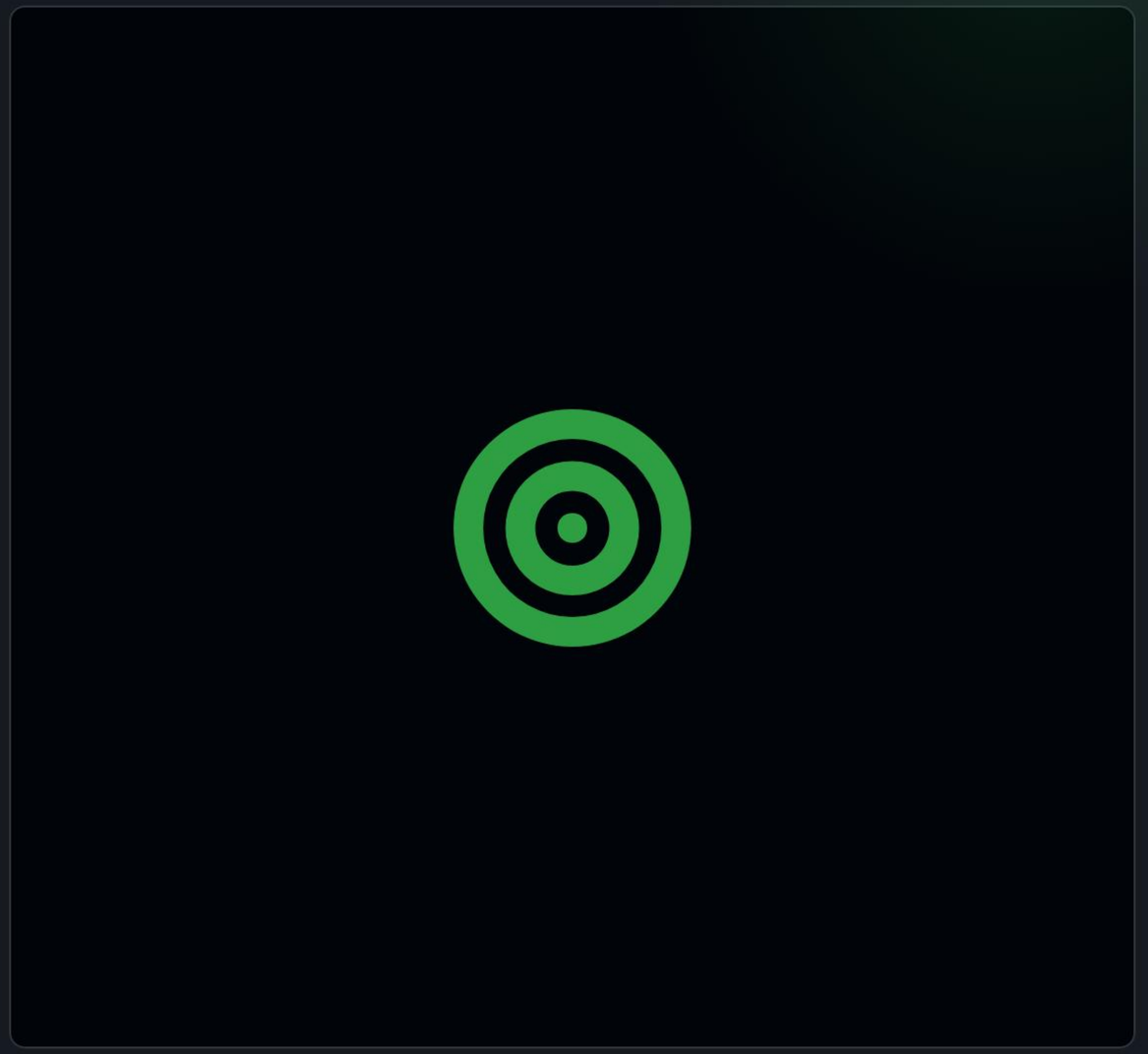
The parameter **C** tells the SVM optimization how much you want to avoid misclassifying each training example.

- **Large C:** Small margin, strict classification (Risk of Overfitting).
- **Small C:** Large margin, allows more errors (Risk of Underfitting).

Non-Linear Data

In many real-world scenarios, data is not linearly separable (e.g., concentric circles).

A straight line cannot separate red from blue here.

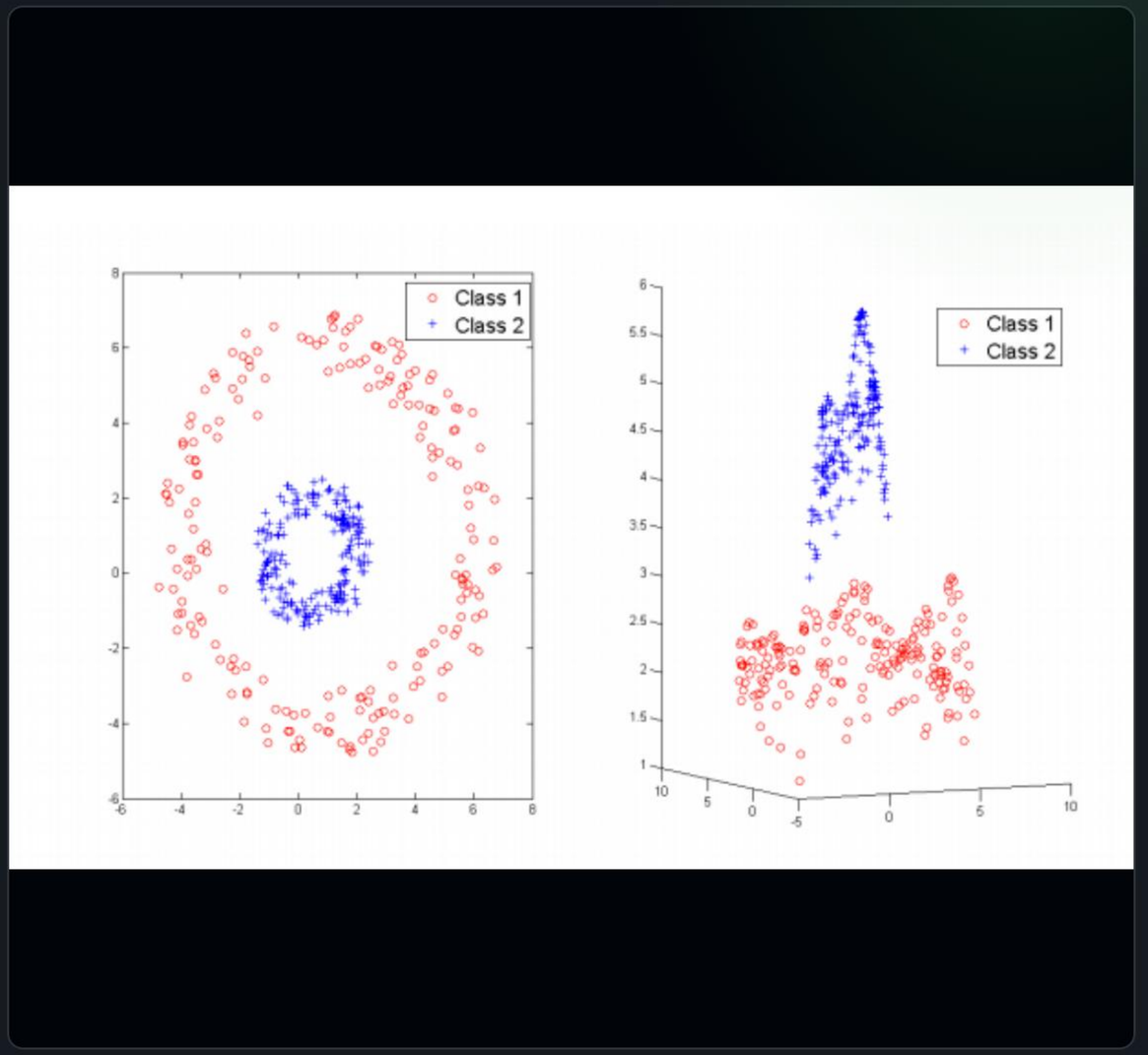


The Kernel Trick

SVM solves non-linear problems by mapping data to a **higher dimension**.

In 3D, complex 2D points might become separable by a flat plane.

The "Trick": It calculates high-dimensional relationships without actually transforming the data coordinates.



Types of Kernels

Linear

For simple, linearly separable data. Very fast.

Polynomial

Maps to polynomial dimensions. Good for image processing.

RBF (Gaussian)

Radial Basis Function. Maps to infinite dimensions. Most common.

RBF Kernel Explained

The Radial Basis Function kernel measures the distance between two vectors.

$$K(x, x') = \exp - \gamma \| x - x' \|^2$$

It creates non-linear decision boundaries like circles or curves.

Parameter Gamma (γ)

Used specifically in the RBF kernel.

- **High Gamma:** Influence of a single training example is close. Decision boundary becomes jagged (Overfitting).
- **Low Gamma:** Influence is far. Decision boundary is smooth (Underfitting).

Support Vector Regression (SVR)

SVM principles can be applied to regression.

Instead of a class margin, SVR tries to fit the error within a certain threshold (ϵ).

Points inside the ϵ-tube are ignored; points outside contribute to the cost.

Advantages of SVM

- Effective in **high-dimensional** spaces.
- Effective even when number of dimensions > number of samples.
- **Memory Efficient:** Uses a subset of training points (support vectors).
- **Versatile:** Different kernel functions can be specified.

Disadvantages of SVM

- **Training Time:** Not suitable for large datasets.
- **Noise Sensitivity:** Performs poorly if data has lots of noise or overlapping target classes.
- **No Probability:** Doesn't directly provide probability estimates (requires expensive cross-validation).

Python: Setup for SVM

```
from sklearn import datasets from sklearn.model_selection import train_test_split from sklearn.svm import SVC # 1. Load Data (Iris) iris = datasets.load_iris() X = iris.data y = iris.target # 2. Split Data X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.3 )
```

SVC: Support Vector Classification class.
iris.target: The classification labels (0, 1, 2).

Python: Training with RBF

```
# 3. Initialize with Kernel and Regularization model = SVC(kernel='rbf', C=1.0, gamma='scale') # 4. Train the Model model.fit(X_train, y_train) print("SVM Training Complete")
```

kernel='rbf': Use Radial Basis Function for non-linear boundaries.
C=1.0: Standard regularization.

Python: Predictions

```
# 5. Predict on Test Set y_pred = model.predict(X_test) # Predict single new sample new_flower = [[5.1, 3.5,
1.4, 0.2]] pred = model.predict(new_flower) print(f"Prediction: {iris.target_names[pred[0]]}")
```

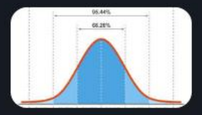
y_pred: The array of predicted classes.
iris.target_names: Maps 0,1,2 to 'setosa', 'versicolor', etc.

Python: Evaluation

```
23 from sklearn.metrics import accuracy_score # 6. Check Accuracy acc = accuracy_score(y_test, y_pred)
24 print(f"SVM Accuracy: {acc:.2f}")
25
26
27
```

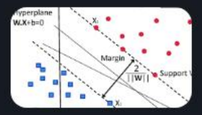
acc: Floating point accuracy (0.0 to 1.0).

Image Sources



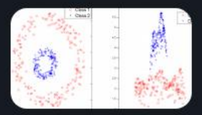
https://www.simplypsychology.org/wp-content/uploads/normal-distribution-1024×640.jpeg

Source: simplypsychology.org



https://miro.medium.com/v2/resize:fit:1400/0*5EsKRZqZuZEplh92.png

Source: medium.com



https://i.sstatic.net/7yM2K.png

Source: stats.stackexchange.com