**Assignment: Naïve Bayes Classification**

**Aim**

To implement and understand the working of the Naïve Bayes classification algorithm using Python and demonstrate its practical applications in predictive modeling.

## **Prerequisite**

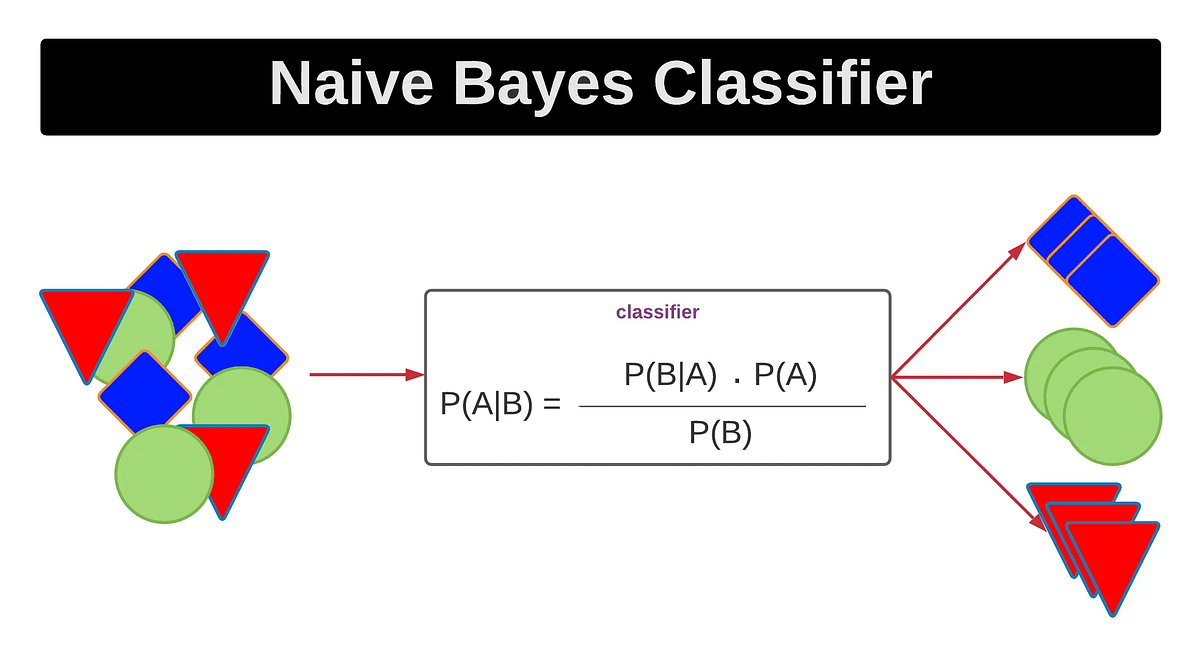
Basic proficiency in Python programming and understanding of probability theory.

### **Introduction to Naïve Bayes**

Naïve Bayes classifier is a probabilistic machine learning classifier model based on Bayes' Theorem and suitable for very high-dimensional, large datasets where there is independence of predictors assumed. While the independence assumption is indeed a simplification, in reality, this generally produces remarkably good results.

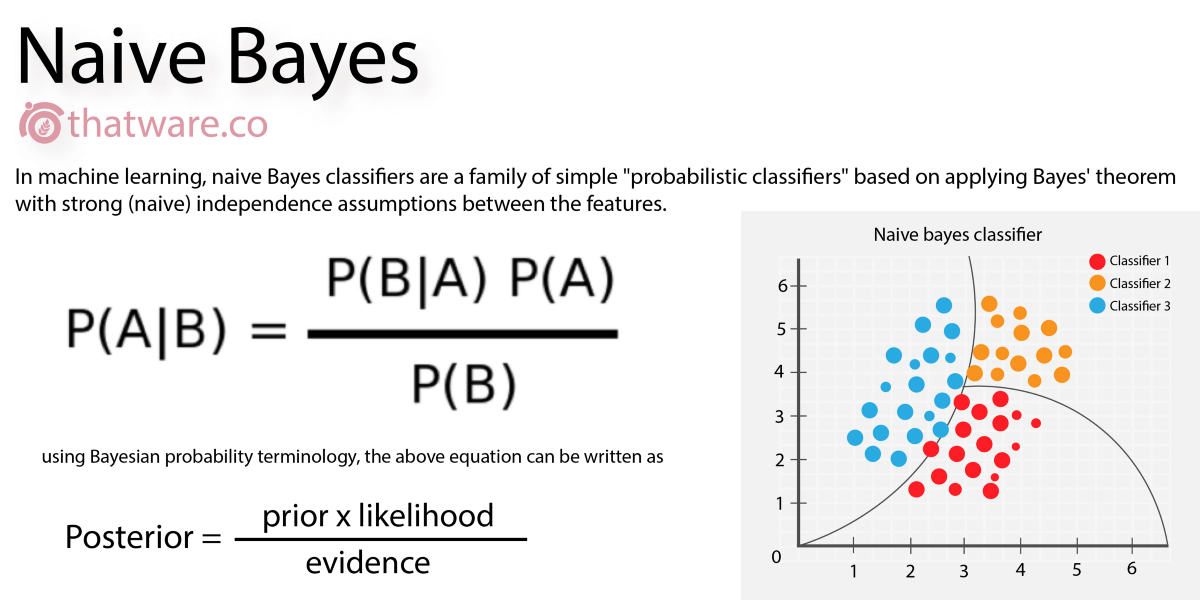
### **Bayes' Theorem**

Bayes' Theorem provides a way to update our prior beliefs with new evidence:



Where:

* P(A|B): Posterior probability of class A​ given feature vector B
* P(B): Prior probability of class B
* P(A|B): Likelihood of data A given class B
* P(B): Evidence or normalizing constant

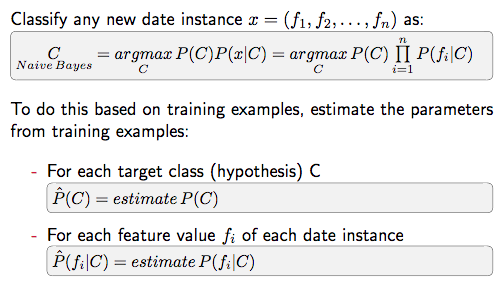


### **Types of Naive Bayes Classifiers:**

1. **Gaussian Naive Bayes** Assumes that the continuous values associated with each class are distributed according to a Gaussian (normal) distribution.
2. **Multinomial Naive Bayes** Useful for discrete features such as word counts in text classification problems (e.g., bag-of-words model).
3. **Bernoulli Naive Bayes** Designed for binary/boolean features — ideal when features represent yes/no type variables (e.g., word present or absent in a document).

### **Technical Perspective**

In practice, Naïve Bayes deals with both **categorical and continuous** variables. Given a feature set X={x1,x2,...,xd}X = {x\_1, x\_2, ...,xd}X={x1​,x2​,...,xd​}, and a class variable C={c1,c2,...,ck} , C = {c\_1, c\_2, ..., c\_k} ,C={c1​,c2​,...,ck​}, we compute:



Assuming **feature independence**, this drastically reduces computational complexity.

Basic Example

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score, classification\_report

# Load dataset

data = load\_iris()

X, y = data.data, data.target

# Train-test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Train Naive Bayes model

model = GaussianNB()

model.fit(X\_train, y\_train)

# Predict and evaluate

y\_pred = model.predict(X\_test)

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

print("Classification Report:\n", classification\_report(y\_test, y\_pred))

This demonstrates the Naïve Bayes classifier applied to the popular **Iris Dataset**, achieving high accuracy despite its simplicity.

## **Applications of Naïve Bayes**

* **Spam Detection:** Classifies emails as spam or non-spam.
* **Sentiment Analysis:** Categorizes reviews or tweets into positive/negative sentiments.
* **Medical Diagnosis:** Assesses patient symptoms to predict disease likelihood.
* **Document Classification:** Assigns articles to predefined categories.

## **Advantages**

* Simple and fast.
* Performs well with high-dimensional data.
* Robust to irrelevant features.
* Requires less training data.

## **Limitations**

* Strong assumption of feature independence.
* Struggles when predictors are highly correlated.
* Not ideal for complex decision boundaries.

## **Conclusion**

The **Naïve Bayes classifier**, despite its simplistic assumptions, remains a powerful and effective tool in the field of machine learning. It forms the basis of many real-world applications and often serves as a benchmark for evaluating other classifiers. Its ease of implementation and interpretation makes it an excellent starting point for beginners and a reliable model for experts.