# Naive Bayes Classifier

**Definition**

**Naive Bayes** is a **probabilistic classification algorithm** based on **Bayes’ Theorem**, assuming that all features are **independent** of each other.

In simple words:

It predicts the class of an example by calculating **the probability** that it belongs to each class, and then picking the class with the **highest probability**.

**Mathematical Foundation: Bayes’ Theorem**

Where:

* = Probability of class **Y** given the input **X** (posterior)
* = Probability of input features given class (likelihood)
* = Probability of the class (prior)
* = Probability of the input data (evidence)

**Why “Naive”?**

Because it **assumes all features are independent**, which is rarely true in reality but surprisingly, it still works very well!

Example:  
In spam detection, even though words in an email are related, Naive Bayes assumes they’re independent.

**Types of Naive Bayes**

|  |  |  |
| --- | --- | --- |
| **Type** | **When to Use** | **Example** |
| **Gaussian Naive Bayes** | When features are **continuous** and follow a **normal distribution** | Predicting diseases using lab values |
| **Multinomial Naive Bayes** | For **discrete counts**, e.g., word counts in text | Text classification, spam detection |
| **Bernoulli Naive Bayes** | For **binary features (0/1)** | Document has a word or not |

**Working Principle (Step-by-Step Example)**

**Example: Spam Email Detection**

|  |  |  |
| --- | --- | --- |
| **Word** | **Spam (Yes)** | **Not Spam (No)** |
| “free” | 8 | 1 |
| “win” | 7 | 2 |
| “meeting” | 1 | 9 |

Suppose an email has words **[“free”, “win”]**.

We calculate:

and

Whichever is higher → model classifies the email as **Spam** or **Not Spam**.

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

X, y = load\_iris(return\_X\_y=True)

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

model = GaussianNB()

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

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

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

# K-Nearest Neighbors (KNN)

**Definition**

**KNN** is a **non-parametric, instance-based** learning algorithm that classifies a new data point based on the **majority class of its K nearest neighbors** in the training data.

In simple terms:

“Tell me who your neighbors are, and I’ll tell you who you are.”

**Working Principle**

1. Choose a value for **K** (e.g., 3 or 5)
2. Compute the **distance** between the new point and all training points (using Euclidean distance)
3. Select the **K nearest** data points
4. Perform **majority voting** among those neighbors
5. Assign the class with the **highest votes** to the new point

**Distance Formula**

For two points and :

**Example**

|  |  |  |  |
| --- | --- | --- | --- |
| **Point** | **X** | **Y** | **Class** |
| P1 | 1 | 2 | Red |
| P2 | 2 | 3 | Red |
| P3 | 3 | 3 | Blue |
| P4 | 6 | 5 | Blue |

We need to classify **Q(3,2)** with K=3.

* Distance(Q,P1)=√((3−1)²+(2−2)²)=2
* Distance(Q,P2)=√((3−2)²+(2−3)²)=1.41
* Distance(Q,P3)=√((3−3)²+(2−3)²)=1
* Distance(Q,P4)=√((3−6)²+(2−5)²)=4.24

Closest 3 neighbors → P2, P3, P1  
Classes → [Red, Blue, Red]  
Majority = **Red**  
**Q is classified as Red**

from sklearn.datasets import load\_iris

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

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score

X, y = load\_iris(return\_X\_y=True)

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

model = KNeighborsClassifier(n\_neighbors=5)

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

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

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