

## Classification- Naive Bayes

**Classification** is a **supervised learning** technique where the goal is to **predict a categorical label** for a given input.

### Examples:

- **Email classification:** spam or not spam
- **Disease detection:** positive or negative
- **Sentiment analysis:** positive, neutral, or negative

### What is Naive Bayes (NB)?

Naive Bayes is a **probabilistic classifier** based on **Bayes' Theorem** with a **naive assumption**: all features are **independent** given the class.

### Bayes' Theorem:

### ◆ 3. Bayes' Theorem:

$$P(C|X) = \frac{P(X|C) \cdot P(C)}{P(X)}$$

Where:

- $P(C|X)$ : Posterior probability of class  $C$  given input  $X$
  - $P(X|C)$ : Likelihood of input  $X$  given class  $C$
  - $P(C)$ : Prior probability of class  $C$
  - $P(X)$ : Evidence (probability of input  $X$ ) — we can ignore it during classification as it's constant
- 

### ◆ 4. Naive Bayes Assumption:

If input  $X$  has features  $x_1, x_2, \dots, x_n$ , NB assumes:

$$P(X|C) = P(x_1|C) \cdot P(x_2|C) \cdots P(x_n|C)$$

So,

$$P(C|X) \propto P(C) \cdot \prod_{i=1}^n P(x_i|C)$$

Naive Bayes is a probabilistic classifier that applies **Bayes' Theorem** with a **strong (naive) assumption**:

Given the class label, the presence or absence of each word (feature) is independent of all other words.

Formally:

For a class  $C$  and features (words)  $x_1, x_2, \dots, x_n$ :

$$P(x_1, x_2, \dots, x_n|C) = P(x_1|C) \cdot P(x_2|C) \cdots P(x_n|C)$$

This assumption simplifies computation, making the model fast and easy to train, especially for text classification.

### Why It's a Problem in Real Language:

In real-world text — especially **informal, sarcastic, or idiomatic expressions** — words often **depend on each other** for meaning.

### Examples:

- *"Great, just what I needed!"* — sounds positive but may be sarcastic.
  - Words like "Great" depend on context (e.g., "just what I needed" can flip the sentiment).
- *"Not bad at all"* — the word "Not" negates "bad", making it positive.
  - The model may treat "bad" as negative, without considering "Not".

So, when the model assumes independence, it **fails to capture**:

- Context
- Word order
- Interactions between words
- Naive Bayes is based on a simplifying assumption: **features are independent given the class**.
- This helps with speed and simplicity.
- But it can lead to poor performance in situations where **word meaning depends on context**, such as **sarcasm, irony, or informal speech**.

## Types of Naive Bayes:

- **Gaussian NB**: Assumes continuous features follow Gaussian (normal) distribution
- **Multinomial NB**: For document classification (word counts)
- **Bernoulli NB**: For binary/boolean features (e.g., word present or not)

### Example: Email Spam Detection (Multinomial NB)

Review	Sentiment
I love this movie	Positive
This movie is amazing	Positive
I hate this movie	Negative
This movie is terrible	Negative
I really love this film	Positive
I really hate this film	Negative

### Step 1: Preprocess the Text

Tokenize and remove stop words (for simplicity, assume we keep only important words):

- **Positive reviews tokens:**

- love, movie
- movie, amazing
- really, love, film

- **Negative reviews tokens:**

- hate, movie
- movie, terrible
- really, hate, film

**Vocabulary:**

[love, movie, amazing, hate, terrible, really, film]

Calculate Class Priors

There are 6 reviews: 3 positive, 3 negative.

$$P(\text{Positive}) = \frac{3}{6} = 0.5, \quad P(\text{Negative}) = \frac{3}{6} = 0.5$$

Word Frequencies

**For Positive:**

- love: 2
- movie: 2
- amazing: 1
- really: 1
- film: 1

Apply Laplace Smoothing:

Apply Laplace Smoothing:

$$P(\text{word}|\text{Positive}) = \frac{\text{count} + 1}{7 + 7} = \frac{\text{count} + 1}{14}$$

Example:

- $P(\text{love}|\text{Positive}) = \frac{2+1}{14} = \frac{3}{14}$
- $P(\text{hate}|\text{Positive}) = \frac{0+1}{14} = \frac{1}{14}$

Do it for all words

## Predict on a New Review

**New review:** "I really love this movie"

After removing stop words: ["really", "love", "movie"]

**For Positive:**

**For Positive:**

$$\begin{aligned} P(\text{Positive}|\text{review}) &\propto P(\text{Positive}) \cdot P(\text{really}|\text{Positive}) \cdot P(\text{love}|\text{Positive}) \cdot P(\text{movie}|\text{Positive}) \\ &= 0.5 \cdot \frac{2}{14} \cdot \frac{3}{14} \cdot \frac{3}{14} \end{aligned}$$

**For Negative:**

$$= 0.5 \cdot \frac{2}{14} \cdot \frac{1}{14} \cdot \frac{3}{14}$$

Compare both values, choose the higher one → label as **Positive**.

```
from sklearn.feature_extraction.text import CountVectorizer
```

```
from sklearn.naive_bayes import MultinomialNB
```

```
from sklearn.pipeline import make_pipeline
```

```
# Sample dataset
```

```
texts = [
```

```
    "I love this movie",
```

```
    "This movie is amazing",
```

```
    "I hate this movie",
```

```
    "This movie is terrible",
```

```
    "I really love this film",
```

```
    "I really hate this film"
```

```
]
```

```
labels = ["Positive", "Positive", "Negative", "Negative", "Positive", "Negative"]
```

```
# Model pipeline
```

```
model = make_pipeline(CountVectorizer(), MultinomialNB())
```

```
# Train the model
```

```
model.fit(texts, labels)
```

```
# Predict on a new review
```

```
print(model.predict(["I really love this movie"])) # Output: ['Positive']
```

```
print(model.predict(["I hate this terrible film"])) # Output: ['Negative']
```

## Advantages of Naive Bayes –

### 1. Fast and Efficient:

- Trains quickly, even on large, messy text data.
- Works well with high-dimensional data (e.g., thousands of words).

### 2. Robust to Irrelevant Features:

- Because it treats each feature independently, the impact of irrelevant or noisy words (e.g., typos, emojis, repeated characters) is often **diluted**.

### 3. Performs Surprisingly Well:

- Despite its naive assumptions, it often gives **strong baselines**, especially for tasks like **spam detection, sentiment analysis, or topic classification**.

## ✗ Limitations on Noisy Social Media Data:

### 1. Can't Handle Word Dependencies:

- Informal language, sarcasm, slang, abbreviations, and emojis often require **context** to understand. Naive Bayes ignores this.
- Example:
  - "I'm dying 😂😂" → positive (humor)
  - "I'm dying inside." → negativeNaive Bayes treats words like "dying" the same in both cases.

### 2. Sensitive to Feature Quality:


- Misspellings, creative punctuation, code-mixed languages, or hashtags can reduce accuracy because they introduce many rare/unseen tokens.
- E.g., "luv u so mch", "#blessedaf"

### 3. Assumption of Feature Independence is Often Violated:

- Especially harmful in **short texts**, where dependencies matter more (since there are fewer words per instance).



## Comparison: Lexicon-Based vs. Statistical Methods (e.g., Naive Bayes)

Feature	Lexicon-Based	Statistical (e.g., Naive Bayes) 
Approach	Uses predefined sentiment dictionaries (e.g., positive and negative word lists)	Learns from labeled data using probability distributions
Training Data	Not required	Required (supervised learning)
Context Handling	Weak – each word is analyzed in isolation	Moderate – can learn associations and patterns between words and classes
Handling Negation/Irony	Poor unless explicitly coded	Limited – better than lexicon, but still weak in nuanced language
Customizability	Easy to modify lexicons	Needs retraining with new data
Speed	Fast, rule-based	Fast, with more flexibility
Performance on Short, Noisy Texts	Often poor	Can be better with good preprocessing

### Why Lexicon-Based Methods Struggle with Domain-Specific Texts?

#### 1. Vocabulary Mismatch

- Lexicons are often built from general-purpose English.
- Domain-specific terms (e.g., medical, finance, tech, gaming) may not be included or may have different sentiment.
- Example:
  - "The infection was aggressive."  
→ In medicine, neutral/clinical, but may be misclassified as negative.

#### 2. Polysemy and Context Dependence

- Words change meaning in different contexts:
  - "Killer performance" → positive in entertainment

- "Killer disease" → negative in healthcare

- Lexicons treat "killer" as uniformly negative.

### 3. Inability to Learn

- Lexicon-based approaches don't adapt to new data or slang.
- They can't learn that "fire" might mean awesome in slang without being explicitly told.

### 4. No Handling of Word Interactions or Syntax

- Phrases like:
  - "Not bad" (positive)
  - "Hardly helpful" (negative)  
are challenging for simple word-level scoring.

### 5. Emoji, Hashtags, and Abbreviations

- Lexicons often don't include:
  - Emojis 🍕 🤔
  - Hashtags (#blessed, #fail)
  - Acronyms (LOL, SMH)
- These are crucial in social media and other domain-specific corpora.