Naive Bayes Types

1. Gaussian Naive Bayes (GNB)

`from sklearn.` `naive_bayes` `import` `GaussianNB`

- **Use Case**: For **continuous data** (e.g., height, weight, temperature).
- **Assumption**: Features follow a **normal (Gaussian) distribution**.

$$P(B_i|A) = \frac{1}{\sqrt{2\pi\sigma_A^2}} e^{-\frac{(B_i-\mu_A)^2}{2\sigma_A^2}}$$

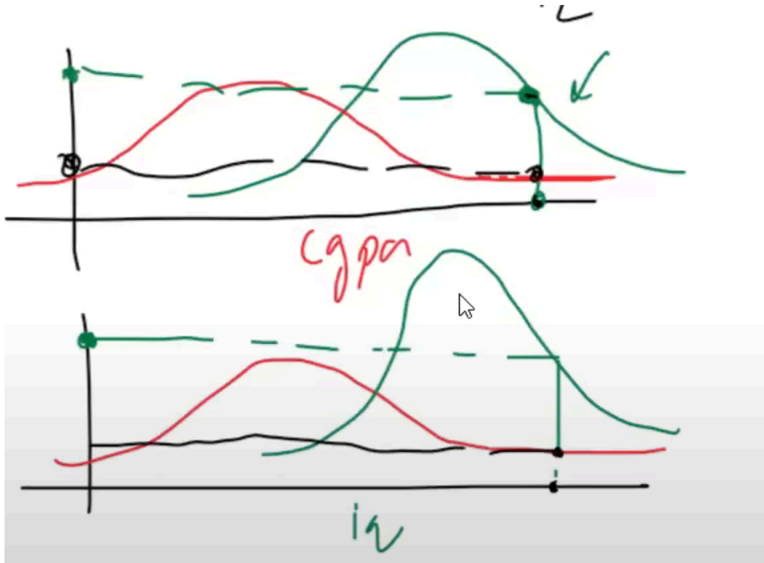
- μ_A: **Mean** of feature B_i for class A.
- σ_A: **Standard deviation** of feature B_i for class A.

Use Cases

✔️ Age, height, weight predictions

✔️ Weather classification

✔️ Medical diagnosis




```
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

# Load dataset
iris = load_iris()
X, y = iris.data, iris.target

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_stat
e=42)

# Create a Gaussian Naive Bayes model
model = GaussianNB()

# Train the model
model.fit(X_train, y_train)

# Predict on the test set
y_pred = model.predict(X_test)
```

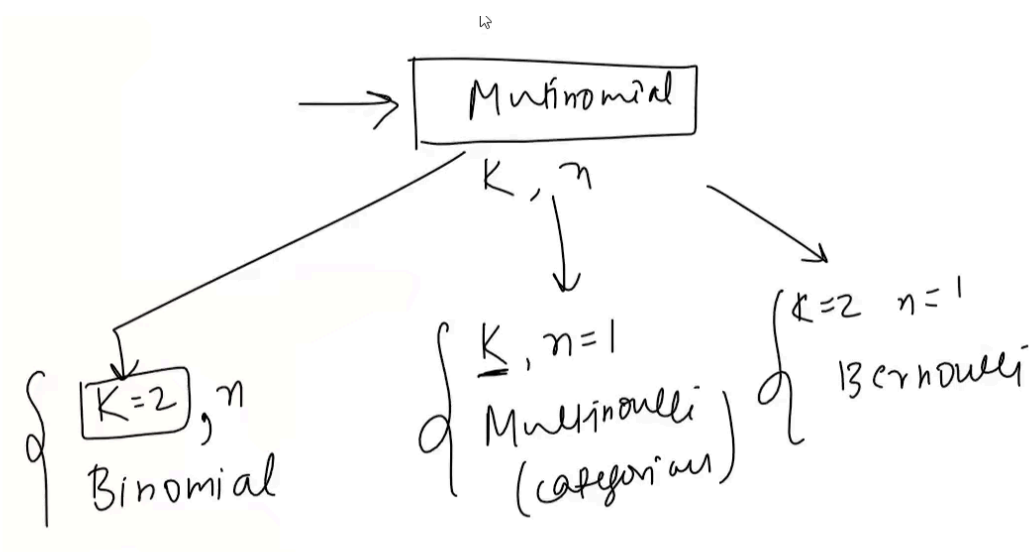
```
# Evaluate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

Output: Accuracy: 1.0
```

2. Multinomial Naive Bayes:

`from sklearn.naive_bayes import MultinomialNB`

> **Multinomial Naive Bayes works well on text data.**



- **Use Case**: For **discrete data** where features represent **counts** or **frequencies** (e.g., word counts in text).
- **Assumption**: Features follow a **multinomial distribution**.

$$P(B_i|A) = \frac{\text{Count of } B_i \text{ in class } A + \alpha}{\text{Total count of all features in class } A + \alpha \cdot n}$$

- α: Smoothing parameter.
- n: Number of unique categories.

Use Cases

✅Spam filtering

✅Sentiment analysis

✅News categorization

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

# Example text data
texts = ["I love programming", "I hate bugs", "Programming is fun", "Bugs are annoying"]
labels = [1, 0, 1, 0]  # 1 = positive, 0 = negative

# Convert text to feature vectors
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(texts)

# Train Multinomial Naive Bayes
model = MultinomialNB()  # alpha=1 (default) → Laplace Smoothing
model.fit(X, labels)

# Predict
new_text = ["I love coding"]
new_X = vectorizer.transform(new_text)
print("Prediction (Multinomial):", model.predict(new_X))
```

```
Output:
Prediction (Multinomial): [1]
```

`alpha=1.0` ***is default***

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

emails = ["Free money now", "Click to win a prize", "Hello, how are you?", "Let's meet for coffee"]
labels = [1, 1, 0, 0]  # 1 = spam, 0 = not spam

vectorizer = CountVectorizer()
X = vectorizer.fit_transform(emails)

mnb = MultinomialNB()
mnb.fit(X, labels)

print(mnb.predict(vectorizer.transform(["Win free cash"])))  # Output: [1] (spam)

Output:
[1]
```

3. Categorical Naive Bayes

`from sklearn.naive_bayes import CategoricalNB`

- **Use Case**: For **categorical data** (e.g., color, type, category).
    - When all columns are categorical
- **Assumption**: Features follow a **categorical distribution**.

- **Example**: Classifying products based on their category (e.g., electronics, clothing).

$$P(B_i|A) = \frac{\text{Count of } B_i \text{ in class } A + \alpha}{\text{Total count of all features in class } A + \alpha \cdot n}$$

- α: Smoothing parameter.
- n: Number of unique categories.

```
from sklearn.naive_bayes import CategoricalNB
import numpy as np

X = np.array([[1, 2, 3], [4, 5, 6], [1, 2, 4], [4, 5, 3]])
y = np.array([0, 1, 0, 1])

cat_nb = CategoricalNB()
cat_nb.fit(X, y)

print(cat_nb.predict([[1, 2, 3]]))  # Output: class label

Output:
[0]
```

4. Bernoulli Naive Bayes (BNB)

- When the **features** are **binary** (i.e., they represent the presence or absence of a feature)
- **Assumption**: Features are **binary** (0 or 1).

Formula:

$$P(B_i|A) = P(B_i = 1|A) \cdot B_i + (1 - P(B_i = 1|A)) \cdot (1 - B_i)$$

- $P(B_i = 1 \mid A)$: Probability of feature B_i being 1 in class A.

- **Example**: Classifying documents based on the presence/absence of specific words.

```
from sklearn.naive_bayes import BernoulliNB

# Example binary data
X = [[1, 0, 1], [0, 1, 0], [1, 1, 0], [0, 0, 1]]
y = [1, 0, 1, 0]  # 1 = positive, 0 = negative

# Train Bernoulli Naive Bayes
model = BernoulliNB()
model.fit(X, y)

# Predict
new_X = [[1, 0, 0]]
print("Prediction (Bernoulli):", model.predict(new_X))

Output:
Prediction (Bernoulli): [1]
```

💡 **Note: 👇 The below example will not show good accuracy as the dataset is imbalanced.**

We have tried the same dataset with Complement Naive Bayes

Spam Classifier:

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import BernoulliNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report

# Load dataset with specified encoding
df = pd.read_csv(r"https://raw.githubusercontent.com/shrudex/sms-spam-det
ection/refs/heads/main/sms-spam.csv")


#Convert labels to binary (spam = 1, ham = 0)
df['v1'] = df['v1'].map({'ham': 0, 'spam': 1})

df.dropna(axis=1, inplace=True)

df.head()
```

	v1	v2
0	0	Go until jurong point, crazy.. Available only ...
1	0	Ok lar... Joking wif u oni...
2	1	Free entry in 2 a wkly comp to win FA Cup fina...
3	0	U dun say so early hor... U c already then say...
4	0	Nah I don't think he goes to usf, he lives aro...

Convert Text into Binary Features

- `CountVectorizer(binary=True)`: Converts text into **binary vectors**, where 1 = word present, 0 = word absent.

```
# Convert text to binary features
vectorizer = CountVectorizer(binary=True, stop_words='english')
```

```
X = vectorizer.fit_transform(df['v2'])

# Labels (spam = 1, ham = 0)
y = df['v1']

# Split into training & testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_stat
e=42)

"Vocabulary size:", len(vectorizer.get_feature_names_out())

Output:
('Vocabulary size:', 8405)
```

Train & Evaluate Bernoulli Naïve Bayes Model

```
# Train Bernoulli Naïve Bayes
bnb = BernoulliNB()
bnb.fit(X_train, y_train)

# Predictions
y_pred = bnb.predict(X_test)

# Accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Classification Report
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 0.9748878923766816

Classification Report:
              precision    recall  f1-score   support

           0       0.97      1.00      0.99       965
           1       0.98      0.83      0.90       150

    accuracy                           0.97      1115
   macro avg       0.98      0.91      0.94      1115
weighted avg       0.98      0.97      0.97      1115
```

5. Complement Naive Bayes

- **Use Case**: For **imbalanced datasets** (e.g., one class has significantly more samples than others).
- Works well when one class dominates the dataset

- **Assumption**: A **variant of Multinomial Naive Bayes** designed to handle imbalanced data.
- **Formula**:
  - Uses the complement of each class to calculate probabilities.
  - Adjusts for class imbalance by weighting the likelihoods.
- **Example 1**: Classifying **rare events**, such as fraud detection.

  Example 2: This is typically used in situations where you have **imbalanced classes**, such as a large number of "non-spam" emails versus "spam" emails. Complement Naive Bayes helps the classifier perform better on the minority class.

```
from sklearn.naive_bayes import ComplementNB

# Example imbalanced data
```

```
X = [[1, 2], [2, 3], [3, 4], [4, 5], [5, 6], [6, 7]]
y = [0, 0, 0, 0, 1, 1]  # Imbalanced classes

# Train Complement Naive Bayes
model = ComplementNB()
model.fit(X, y)

# Predict
new_X = [[2, 3]]
print("Prediction (Complement):", model.predict(new_X))

Output:
Prediction (Complement): [0]
```

Spam Classifier using Complement Naive Bayes

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import BernoulliNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report

# Load dataset with specified encoding
df = pd.read_csv(r"https://raw.githubusercontent.com/shrudex/sms-spam-detection/refs/heads/main/sms-spam.csv")



#Convert labels to binary (spam = 1, ham = 0)
df['v1'] = df['v1'].map({'ham': 0, 'spam': 1})

df.dropna(axis=1, inplace=True)

df.head()
```

	v1	v2
0	0	Go until jurong point, crazy.. Available only ...
1	0	Ok lar... Joking wif u oni...
2	1	Free entry in 2 a wkly comp to win FA Cup fina...
3	0	U dun say so early hor... U c already then say...
4	0	Nah I don't think he goes to usf, he lives aro...

```
from sklearn.naive_bayes import ComplementNB
X = df['v2']
y=df['v1']

vectorizer = CountVectorizer(binary=True, stop_words='english')
X = vectorizer.fit_transform(df['v2'])

model = ComplementNB()
model.fit(X, y)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_stat
e=42)

"Vocabulary size:", len(vectorizer.get_feature_names_out())
```

```
('Vocabulary size:', 8404)
```

Train the data:

```
# Train Bernoulli Naïve Bayes
cnb = ComplementNB()
cnb.fit(X_train, y_train)

# Predictions
y_pred = cnb.predict(X_test)
```

```
# Accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Classification Report
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 0.9408071748878923

Classification Report:
               precision    recall  f1-score   support

           0       0.99      0.94      0.96       965
           1       0.71      0.96      0.81       150

    accuracy                           0.94      1115
   macro avg       0.85      0.95      0.89      1115
weighted avg       0.95      0.94      0.94      1115
```

Test with new messages

```
# Test with new messages
new_messages = ["Win money now!!!", "Hello, how are you?", "Click this link t
o claim your prize!"]
X_new = vectorizer.transform(new_messages)

# Predict spam or ham
predictions = cnb.predict(X_new)

for msg, pred in zip(new_messages, predictions):
    print(f"Message: '{msg}' → {'Spam' if pred == 1 else 'Ham'}")
```

```
Message: 'Win money now!!!' → Spam
Message: 'Hello, how are you?' → Ham
Message: 'Click this link to claim your prize!' → Spam
```

Out-of-Core Naive Bayes

- **Out-of-core learning** means training a model **without loading all the data into memory at once**.
- This is useful when dealing with **huge datasets** that cannot fit into RAM.

- **How it works**: Processes data in small chunks (batches) instead of loading the entire dataset at once.
- **Use Case**: Ideal for big data applications where the dataset exceeds available RAM.

How Out-of-Core Naive Bayes Works

1. **Load Data in Batches**: Read the dataset in small chunks (e.g., 1000 rows at a time).
2. **Update Model Incrementally**: For each batch, update the model's parameters (e.g., counts for Multinomial Naive Bayes).
3. **Repeat**: Continue processing batches until the entire dataset is processed.

```
import numpy as np
import pandas as pd
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split

# Load a large dataset (Example: Spam detection)
df = pd.read_csv(r"https://raw.githubusercontent.com/shrudex/sms-spam-detection/refs/heads/main/sms-spam.csv")
df.dropna(inplace=True, axis=1)
df.columns = ["label", "message"]
df['label'] = df['label'].map({'ham': 0, 'spam': 1})  # Convert labels to binary
```

```
# Convert text into numerical features
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(df['message'])  # Convert text to bag-of-words
y = df['label'].values

# Split data into smaller chunks (simulate streaming)
chunk_size = 1000  # Process 1000 rows at a time
num_chunks = X.shape[0] // chunk_size

# Initialize Naïve Bayes model for out-of-core learning
nb = MultinomialNB()

# Train model in chunks using partial_fit
for i in range(num_chunks):
    start = i * chunk_size
    end = start + chunk_size
    X_chunk, y_chunk = X[start:end], y[start:end]

    if i == 0:
        nb.partial_fit(X_chunk, y_chunk, classes=np.array([0, 1]))  # Initialize with all classes
    else:
        nb.partial_fit(X_chunk, y_chunk)  # Incrementally update model

# Predict on new data
new_messages = ["You won a lottery! Claim your prize now!", "Hello, how are you?"]
X_new = vectorizer.transform(new_messages)
predictions = nb.predict(X_new)

# Output results
for msg, pred in zip(new_messages, predictions):
    print(f"Message: '{msg}' → {'Spam' if pred == 1 else 'Ham'}")
```

```
Message: 'You won a lottery! Claim your prize now!' → Spam
Message: 'Hello, how are you?' → Ham
```

Key Features of `partial_fit`

- **Processes data in chunks** instead of all at once
- **Efficient for large datasets** (reduces memory usage)
- **Allows online learning** (updates model dynamically)
- **Supports only incremental learning models** (`MultinomialNB`, `BernoulliNB`, `GaussianNB`)

Fit vs Partial Fit

Feature	Standard Naïve Bayes (`fit()`)	Out-of-Core Naïve Bayes (`partial_fit()`)
Memory Usage	High (loads all data at once)	Low (processes in chunks)
Suitable for Big Data?	❌ No	✅ Yes
Can update model over time?	❌ No	✅ Yes
Training Speed	Slower for large data	Faster (incremental updates)