Multinomial Naive Bayes

```
from sklearn.naive_bayes import MultinomialNB

# Train Naive Bayes

nb = MultinomialNB()

nb.fit(X_train_tfidf, y_train)

# Predict

y_pred_nb = nb.predict(X_test_tfidf)

# Evaluation

print("=== Naive Bayes ===")

print(classification_report(y_test, y_pred_nb))
```

🔹 **Step 1: Import**

from sklearn.naive_bayes import MultinomialNB

👉 You are using **Multinomial Naive Bayes**, which is good for **text data with counts/TF-IDF features**.

🔹 **Step 2: Train the Model**

nb = MultinomialNB()

nb.fit(X_train_tfidf, y_train)

👉 fit() trains the Naive Bayes classifier using your **training TF-IDF features** (X_train_tfidf) and their **labels** (y_train).

🔹 **Step 3: Predict**

y_pred_nb = nb.predict(X_test_tfidf)

👉 The trained model predicts labels for **test data**.

🔹 **Step 4: Evaluate**

print("=== Naive Bayes ===")

print(classification_report(y_test, y_pred_nb))

👉 classification_report() shows evaluation metrics:

- **Precision** → Out of predicted positives, how many are correct.
- **Recall** → Out of actual positives, how many were correctly predicted.
- **F1-score** → Balance of Precision & Recall.
- **Accuracy** → Overall correctness.

⚡ In short:
You trained a **Multinomial Naive Bayes** model on TF-IDF features → predicted test labels → checked performance with **classification report**.

🔹 1. What is Naive Bayes?

- **Naive Bayes** ek **probabilistic classifier** hai jo Bayes' theorem par based

  - Formula:

$$P(y|x) = \frac{P(x|y) \cdot P(y)}{P(x)}$$

yahan:

- $P(y|x)$ = probability ki class = y given data x
- $P(x|y)$ = likelihood (data x hone ki chance agar class y ho)
- $P(y)$ = prior probability (class ke hone ki chance)

👉 **Naive** isliye kehte hain kyunki ye मानता hai ki **features independent hote hain** (which is rarely true, lekin kaafi cases me kaam karta hai).

🔹 2. Types of Naive Bayes

Scikit-learn me 3 common versions milte hain:

1. **Gaussian Naive Bayes** → Jab features **continuous** hote hain aur normal distribution follow karte hain.
2. **Bernoulli Naive Bayes** → Jab features **binary (0/1)** hote hain (ex: word present or not).
3. **Multinomial Naive Bayes** → Jab features **counts** hote hain (ex: word frequencies, TF-IDF values).

🔹 3. Why Multinomial Naive Bayes for Text?

- Text classification me features (TF-IDF, Bag of Words) **counts ya weighted counts** hote hain.
-
- Example:
    - Doc1: "cat cat dog" → [cat=2, dog=1]
    - Doc2: "dog dog cat" → [cat=1, dog=2]

👉 Ye data **discrete counts** hai → Isliye **Multinomial Naive Bayes** perfect fit hai.

Naive Bayes general (Gaussian) use karne se galat hoga, kyunki TF-IDF values normal distribution follow nahi karte.

🔹 4. Core Concept of MNB

For each class y:

$$P(y|x) \propto P(y) \prod_{i=1}^{n} P(x_i|y)$$

- $P(y)$ = prior probability of class (ex: spam ya ham kitni baar aata hai training data me).
- $P(x_i|y)$ = probability of word i given class y.
- Multiplication hone ki wajah se chhote numbers aa jaate hain → isliye log-probabilities use karte hain.

🔹 5. How it Works in Text Classification

1. Har class ke liye **prior probability** nikalta hai.
   Example: Spam emails = 40%, Ham emails = 60%.

2. Har word ke liye **conditional probability** nikalta hai:
    - P("offer" | Spam) = 0.05
    - P("offer" | Ham) = 0.001
3. New document aaya: "limited time offer"
    - Spam probability = P(Spam) × P(limited|Spam) × P(time|Spam) × P(offer|Spam)
    - Ham probability = P(Ham) × P(limited|Ham) × P(time|Ham) × P(offer|Ham)

      👉 Jis class ki probability zyada hogi, us class me classify kar dega.

🔹 **6. Implementation Example (scikit-learn)**

```
from sklearn.naive_bayes import MultinomialNB

from sklearn.metrics import classification_report


# Model

nb = MultinomialNB()


# Train

nb.fit(X_train_tfidf, y_train)


# Predict

y_pred = nb.predict(X_test_tfidf)


# Evaluation

print(classification_report(y_test, y_pred))
```

🔹 **7. Summary**

- **Naive Bayes** = Probabilistic model using Bayes' theorem.
- **Multinomial Naive Bayes** = Best for **text data (word counts, TF-IDF)**.
- **Why not Gaussian?** → Kyunki hamara feature distribution continuous-normal nahi hai.
- **Why not Bernoulli?** → Wo sirf binary features ke liye hai (word present or absent).

👉 Matlab: Text ke liye **Multinomial Naive Bayes** is the **go-to choice**.
Ye fast, simple aur surprisingly accurate hota hai text classification me (spam detection, sentiment analysis, etc.).

🔹 **1. Why not Bernoulli NB for text?**

- **Bernoulli NB** me features **binary (0 ya 1)** hote hain.
    - Example: Word "sad" present hai → 1
    - Word "sad" absent hai → 0

👉 Matlab bas check karta hai **word aaya ya nahi**.

❌ Problem: Yeh **frequency ignore karta hai**. Agar ek document me "sad" 10 baar likha hai aur dusre me 1 baar — dono ko same treat karega.

🔹 **2. Why Multinomial NB is better for text?**

- **Multinomial NB** word ke **counts** ya TF-IDF weight use karta hai.

- Agar "sad" word depression wale text me baar-baar aata hai, to uski importance zyada hogi.

  ✅ Isliye **Multinomial NB text classification ke liye best** hota hai.

🔹 **3. How it works in Depression Text Data**

Step A: Collect data

Examples:

- "I feel hopeless and tired every day" → **Depressed**
- "I am happy and motivated" → **Not Depressed**

Step B: Convert text → numbers (TF-IDF / CountVectorizer)

Suppose vocab = {hopeless, tired, happy, motivated}

Sentence 1 → [1, 1, 0, 0]
Sentence 2 → [0, 0, 1, 1]

Step C: Learn word probabilities per class

- In **Depressed class**:
  - P(hopeless|Depressed) high
  - P(tired|Depressed) high
- In **Not Depressed class**:
  - P(happy|NotDepressed) high
  - P(motivated|NotDepressed) high

Step D: New text classification

New text: "I feel tired but not hopeless"

- Model will calculate:
  - P(Depressed) × P(tired|Depressed) × P(hopeless|Depressed)

- P(NotDepressed) × P(tired|NotDepressed) × P(hopeless|NotDepressed)

👉 Jis class ka product probability zyada hoga, wahi prediction hoga.

🔹 **4. Summary**

- **Bernoulli NB** → Only word presence/absence (good for short texts like tweets, but loses frequency info).
- **Multinomial NB** → Uses word counts/TF-IDF, captures intensity of words → better for **Depression classification** where repeated negative words matter.

⚡ So in depression detection:

- Agar patient likhta hai *"I am tired tired tired hopeless"*,
  - **Bernoulli**: sirf "tired=1, hopeless=1" lega → weak signal.
  - **Multinomial**: "tired=3, hopeless=1" lega → strong signal of depression. ✅

.

🎯 **Imagine 2 Buckets**

- Ek bucket = **Depressed texts**
- Dusra bucket = **Not Depressed texts**

Har bucket me words ka **count** pada hai.

- Depressed bucket → “tired” 100 baar, “hopeless” 80 baar
- Not Depressed bucket → “tired” 10 baar, “hopeless” 5 baar

🎯 New Text: "I feel tired but not hopeless"

Model kya karta hai?

1. Dekhta hai **Depressed bucket me "tired" kitni baar hai** vs **Not Depressed bucket me "tired" kitni baar hai**.
    - Depressed bucket → tired bahut baar → high chance
    - Not Depressed bucket → tired kam baar → low chance
2. Fir dekhta hai **"hopeless" word dono bucket me kitna aata hai**.
    - Depressed bucket → hopeless zyada
    - Not Depressed bucket → hopeless bahut kam
3. Ab dono words ke chance ko multiply karta hai har bucket ke liye:
    - Depressed bucket → (chance of tired × chance of hopeless)
    - Not Depressed bucket → (chance of tired × chance of hopeless)
4. Jis bucket ka product bada nikalta hai → wahi **prediction** ban jaata hai.

🎯 Ek Chhoti Number Example

Suppose model ne calculate kiya:

- **Depressed bucket** → tired=0.6, hopeless=0.7 → 0.6 × 0.7 = **0.42**
- **Not Depressed bucket** → tired=0.2, hopeless=0.1 → 0.2 × 0.1 = **0.02**

👉 Compare: 0.42 vs 0.02 → bada hai 0.42 → Answer = **Depressed** ✅

🌟 Simple Baat

- Har word ek **vote** deta hai bucket ko.
- Agar word mostly depressed texts me aata hai → Depressed bucket ko zyada vote milega.
- Sab votes (probabilities) multiply karke dekha jaata hai → jis bucket ke votes strong hote hain, wahi final prediction hota hai.