Naive Bayes Introduction

```
from sklearn.naive_bayes import GaussianNB , MultinomialNB , BernoulliNB
```

BEST VIDEO (Multinomial Naive Bayes**)** → https://youtu.be/O2L2Uv9pdDA?si=GD9j6LfBGyQ7F4eL

Gaussian Naive Bayes → **Gaussian Naive Bayes**

What is Naive Bayes?

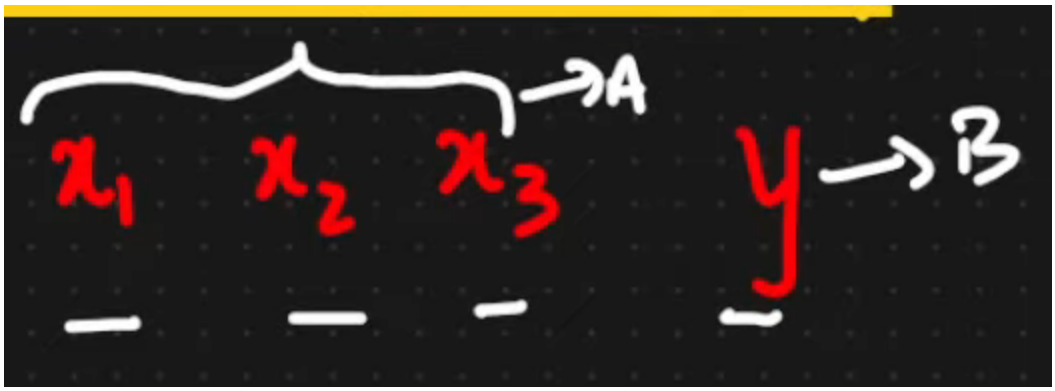
- **Definition**: A probabilistic machine learning algorithm based on **Bayes' Theorem**.
- **Purpose**: Used for **classification tasks**, such as spam detection, sentiment analysis, and medical diagnosis.
- Works well with **Textual Data**
  - Spam Filter
  - Sentiment analysis
- **Key Idea**: Assumes that all features are **independent** of each other (the "naive" assumption).
- Despite this unrealistic assumption, **it works surprisingly well** in many cases, especially in **text classification (spam filtering, sentiment analysis, etc.)**.

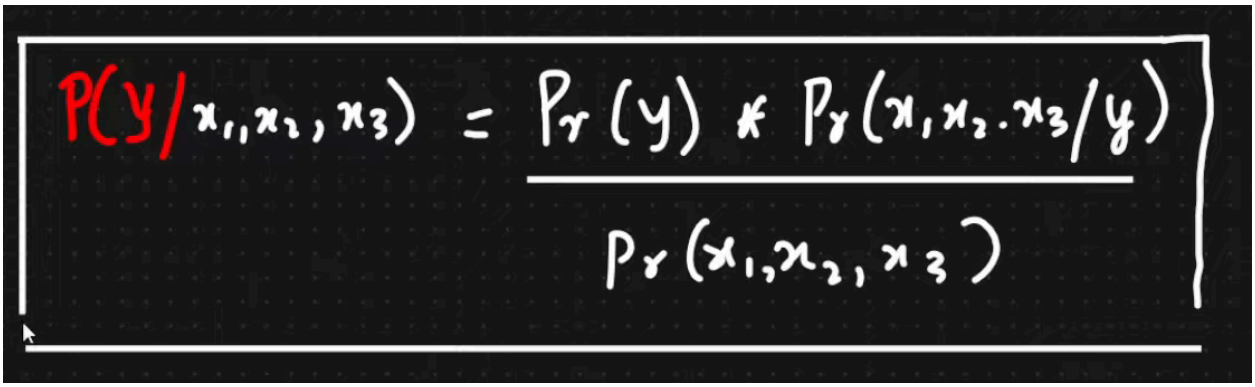
> 💡 It's called **Naive** because it treats all words equal.

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

- $P(A|B)$: Posterior probability (probability of class A given features B).
- $P(B|A)$: Likelihood (probability of features B given class A).
- $P(A)$: Prior probability (initial probability of class A).
- $P(B)$: Marginal probability (probability of features B).

$$P(B|A) = P(B_1|A) \cdot P(B_2|A) \cdot \cdots \cdot P(B_n|A)$$





Intuition

Outlook	Temperature	Humidity	Windy	PlayTennis
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No

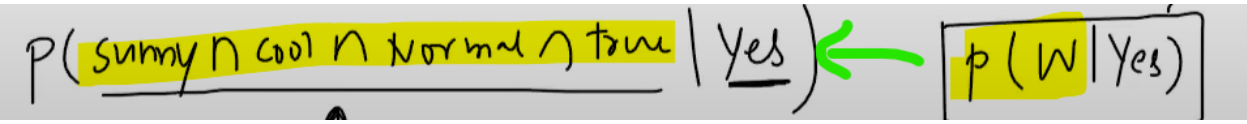
- We have X={Sunny, Cool, Normal, True}
- We want to find if the player will play tennis or not

- For this👆, first we will calculate **probabilities of Yes & No**.
- If probability of **Yes = 0.82** & probability of **No=0.12**, we'll choose **Yes**.

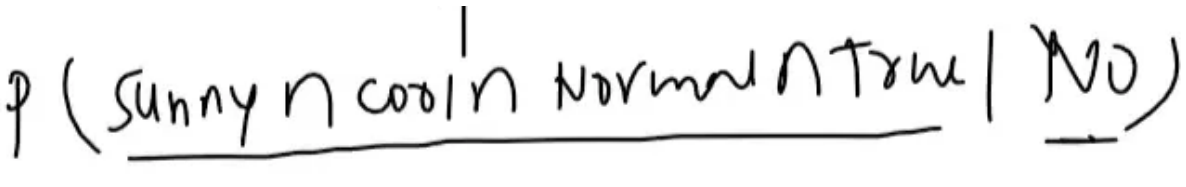
$$P(Yes \mid W) = \frac{P(W \mid Yes)\, P(Yes)}{P(W)}$$

$$P(No \mid W) = \frac{P(W \mid No)\, P(No)}{P(W)}$$

- 👆 You can remove the denominator as it's same.



- 👆 We want to find the combination where probability is **Yes,** with {Sunny, Cool, Normal, True} combination.



- Similarly, 👆 this is, combination where probability is **No,** with {Sunny, Cool, Normal, True} combination.

- **But we cannot find this combination in the above data. Both the probabilities are Zero.**
- 👆This is the problem with Bayes' Theorem: We cannot find the combination

- **Naive Bayes** solves this problem by multiplying the probabilities like 👇

- You do the same with **No.**

For Numeric Values

- We assume it follows the **Gaussian Distribution**
- & calculate **μ & σ**
  - We calculate separate **μ & σ** for **Yes** and **No**
- We'll calculate the Probability Density for the given point
- By this way, even if the number is not in the column, we can have its probability.

- If the data does not follow Gaussian Distribution, you can try out other distributions like:
  - **Multinomial Naive Bayes, Complement Naive Bayes, Bernoulli Naive Bayes, Categorical Naive Bayes.**

Naïve Bayes on Text Data

- Ex. Sentiment Analysis with the help of a text.

- Review vs Sentiment

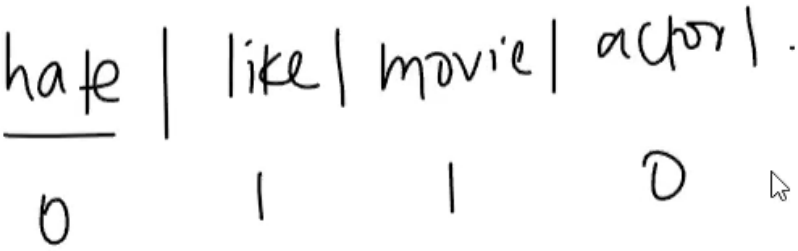
	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive

Steps:

1. Test preprocessing
2. Vectorize the text (Convert text → **numbers**)
    - Bag of Words
    - One Hot Encoding
    - TFIDF
    - Embeddings
3. Apply Naive Bayes

Bag of Words

- Find out total no. of words
- Find out **unique** words
- Find out most frequently used words
- Go through each review and figure out how many times the word occured in the review



Python code:

```
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
```

Underflow

- **Underflow** occurs when a **number is too small** to be represented accurately in a given system or data type.
- Computer **rounds off** this number to **zero.**
- In short, underflow is when a computed value falls outside the smallest possible value that can be represented, leading to inaccurate or zero results.

- In case of multiple columns, the probability could be extremely small.
- To prevent this, we take Log of the probabilities & compare.

`predict_log_proba( X )`

- Log of numbers from 0 to 1 is a negative number.
  - If the number is closer to 0, smaller will be its log.

Laplace Additive Smoothing

`MultinomialNB( alpha=1 )` **←by default**

> **Except Gaussian NB, every other type has alpha parameter.**

- **Definition**: A technique used to handle **zero probabilities** in probabilistic models, such as Naive Bayes.
- **Purpose**: Prevents the model from assigning a probability of zero to any event, which can cause issues in calculations.

- **Also Known As**: Additive Smoothing or Lidstone Smoothing.

Why is Laplace Smoothing Needed?

- In probabilistic models, if a feature value never occurs in a class, the likelihood $P(B_i \mid A)$ becomes zero.
- This can lead to the entire probability P(A ∣ B)$P(A \mid B)$ becoming zero, even if other features strongly suggest the class.
- **Laplace Smoothing adds a small constant(α) to the counts to avoid zero probabilities.**
  - usually **α=1**
- You add α to numerator & $n\alpha$ to denominator.
  - value of n is fixed.

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

  - α: Smoothing parameter (usually $\alpha = 1$ for Laplace Smoothing).
  - n: Number of unique feature values.

- **Effect**:
  - Ensures that no probability is zero.
  - Adds a small "pseudo-count" to each feature value.

Changing the alpha value affects bias & variance.