1. **AIM : BUILD A SIMPLE SENTIMENT ANALYSIS CLASSIFIER USING BAG-OF-WORDS OR TF-IDF AND NAÏVE BAYES.**

**DESCRIPTION**

This project implements a **basic sentiment analysis classifier** that determines whether a given text expresses a **positive** or **negative** sentiment. It uses fundamental **Natural Language Processing (NLP)** techniques combined with a simple yet effective machine learning model.

The classifier is built using the following components:

### ****Bag-of-Words (BoW) Model****

The **Bag-of-Words** model is a simple and widely used technique in Natural Language Processing (NLP). It represents each document (text input) as a collection of words, **disregarding grammar and word order**, but **keeping multiplicity** (word frequency).

* Each unique word becomes a feature.
* A document is represented as a vector of word counts.

Example:

Input Sentences:

1. I love this phone

2. This phone is bad

BoW Features: ['bad', 'is', 'love', 'phone', 'this']

Vector for Sentence 1: [0, 0, 1, 1, 1]

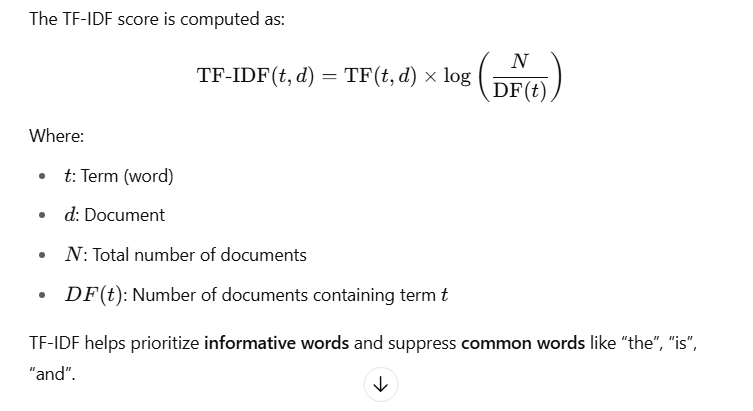
Vector for Sentence 2: [1, 1, 0, 1, 1]

### ****TF-IDF (Term Frequency–Inverse Document Frequency):****

**TF-IDF** is a statistical technique used to reflect how important a word is in a document relative to a collection (corpus) of documents.

It combines two measures:

* **Term Frequency (TF):** Measures how often a word appears in a document.
* **Inverse Document Frequency (IDF):** Reduces the weight of common words that appear in many documents.



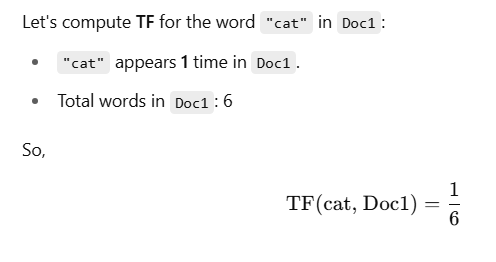
Suppose you have the following **3 documents**:

Doc1: "The cat sat on the mat"

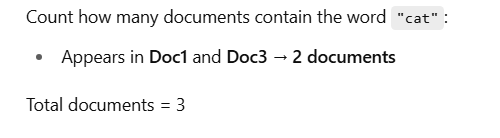
Doc2: "The dog sat on the log"

Doc3: "The cat chased the mouse"

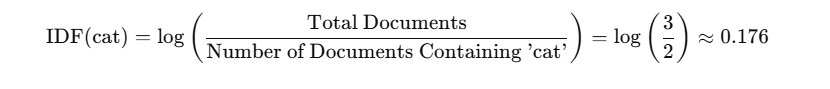
**Step 1: Term Frequency (TF)**



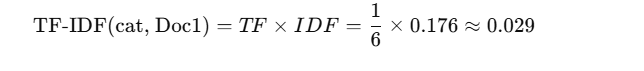
**Step 2: Document Frequency (DF)**



**Step 3: Inverse Document Frequency (IDF)**



Step 4: TF-IDF Score



### ****Interpretation****

* **Common words** like "the", "on" will have **low IDF**, so their TF-IDF score is small.
* **Unique or less frequent words** (like “mouse” or “log”) will have **higher IDF**, meaning they are more important for distinguishing documents.

| **Document** | **TF** | **IDF (log(3/2))** | **TF-IDF** |
| --- | --- | --- | --- |
| Doc1 | 1/6 | 0.176 | 0.029 |
| Doc2 | 0 | 0.176 | 0 |
| Doc3 | 1/6 | 0.176 | 0.029 |

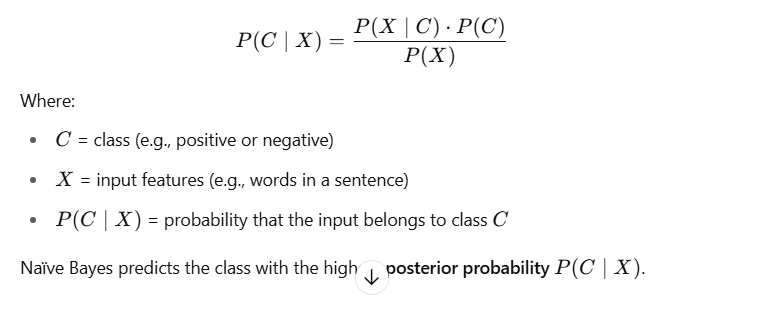
### ****NAÏVE BAYES****

**Naïve Bayes** is a **probabilistic classifier** based on **Bayes’ Theorem** with a strong (naïve) assumption:

All features (words, in our case) are conditionally independent given the class.

Despite this simplification, Naïve Bayes performs very well for **text classification** tasks, such as spam filtering or sentiment analysis.

**BAYES’ THEOREM**



### ****Types of Naïve Bayes:****

* **Multinomial Naïve Bayes** – for text data (word counts or frequencies)
* **Gaussian Naïve Bayes** – for continuous numeric data
* **Bernoulli Naïve Bayes** – for binary/boolean features

For **text classification**, we typically use **Multinomial Naïve Bayes**.

**Simple Example: Sentiment Analysis**

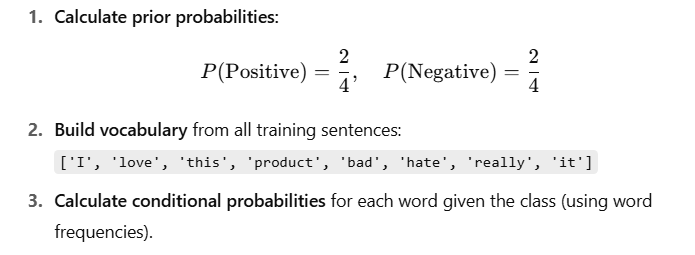
Suppose you train a model with the following labeled sentences:

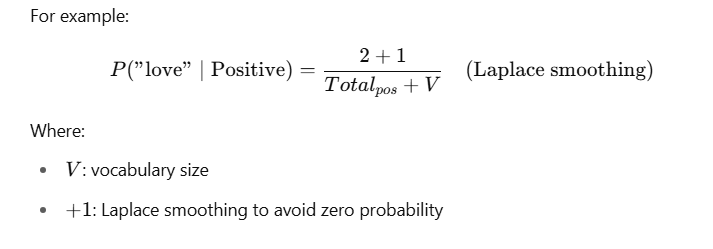
| **Sentence** | **Sentiment** |
| --- | --- |
| I love this product | Positive |
| This is a bad product | Negative |
| I hate it | Negative |
| I really love it | Positive |

You want to classify:

**"I love it"**

Steps Naïve Bayes Takes:





**4. Apply Bayes’ formula** to calculate the probability for each class given the new sentence **"I love it"** and choose the higher one.

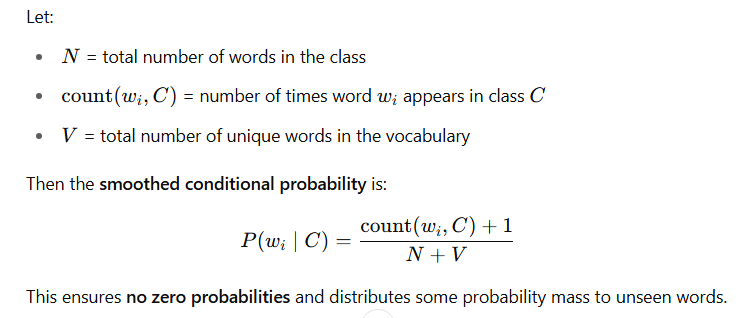
**LAPLACE SMOOTHING**

**Laplace Smoothing** is a technique used in **Naïve Bayes classifiers** to handle the problem of **zero probabilities** for words that are not present in the training data for a specific class.

Suppose you are classifying text and encounter a word **not seen** in training for a particular class.  
Since probabilities are multiplied, a **zero probability** for one word makes the entire product zero — which can mislead the classification.

To fix this, we "pretend" that every word appeared **at least once** by adding 1 to every word count.

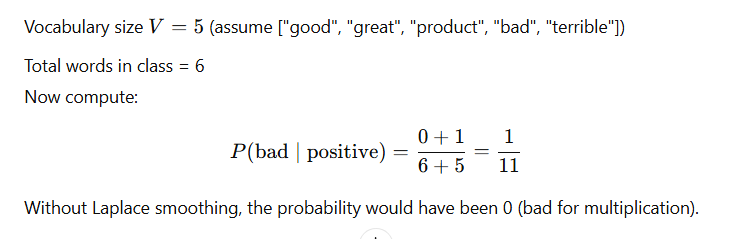
**Formula:**



### Example:

Assume class "positive" has these word counts:

| **Word** | **Count** |
| --- | --- |
| good | 3 |
| great | 2 |
| product | 1 |



| **Without Smoothing** | **With Laplace Smoothing** |
| --- | --- |
| Zero probabilities | No zero probabilities |
| Risk of total 0 | More stable probability model |
| Poor for unseen words | Handles unseen words gracefully |

### ****1. Text Representation – Bag-of-Words or TF-IDF****

To convert raw text into a machine-readable format (numerical vectors), the model uses either:

* **Bag-of-Words (BoW)**: Represents text based on word occurrence.
* **TF-IDF (Term Frequency–Inverse Document Frequency)**: Weighs words based on how frequently they appear in a document compared to how rare they are across all documents.

These methods convert text data into a structured format suitable for machine learning algorithms.

### ****2. Classifier – Naïve Bayes Algorithm****

The classifier uses the **Multinomial Naïve Bayes** algorithm, which is particularly effective for text classification tasks due to its simplicity and performance with high-dimensional data (like word vectors).

* It applies **Bayes' Theorem** with the “naïve” assumption that all features (words) are independent given the class label.

### ****3. Dataset****

A small set of labeled text samples (such as Facebook or Twitter-style comments) is used to simulate sentiment classification. Each comment is labeled as:

* **1** for positive sentiment
* **0** for negative sentiment

This labeled data is split into training and test sets for model training and evaluation.

1. **Text preprocessing** (optional: lowercasing, stopword removal)
2. **Vectorization** using TF-IDF or BoW
3. **Train-test split** of data
4. **Model training** using Naïve Bayes
5. **Prediction** on unseen comments
6. **Evaluation** using accuracy and classification metrics

### ****Output****

* Accuracy of the classifier
* Classification report (precision, recall, F1-score)
* Sentiment predictions for new/unseen text comments

### ****Applications****

* Social media sentiment monitoring
* Customer review analysis
* Feedback classification

**PROGRAM**

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

from sklearn.feature\_extraction.text import TfidfVectorizer # Use CountVectorizer for Bag-of-Words

from sklearn.naive\_bayes import MultinomialNB

from sklearn.metrics import accuracy\_score, classification\_report

# Sample dataset: You can expand this or load from a file

texts = [

"I love this product. It's amazing!",

"This is the worst thing I’ve ever bought.",

"Absolutely wonderful experience, highly recommend.",

"I am very disappointed with this service.",

"Not bad, could be better.",

"Totally worth the money. I'm happy!",

"Terrible experience, will never use again.",

"Fantastic support and great quality.",

"Okay product, not too great.",

"Awful! Complete waste of money."

]

# Labels: 1 for positive sentiment, 0 for negative sentiment

labels = [1, 0, 1, 0, 0, 1, 0, 1, 0, 0]

# Step 1: Split the data

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

# Step 2: Vectorize the text using TF-IDF (you can use CountVectorizer for Bag-of-Words)

vectorizer = TfidfVectorizer()

X\_train\_vect = vectorizer.fit\_transform(X\_train)

X\_test\_vect = vectorizer.transform(X\_test)

# Step 3: Train the Naïve Bayes classifier

model = MultinomialNB()

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

# Step 4: Make predictions

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

# Step 5: Evaluate the model

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

print("\nClassification Report:\n", classification\_report(y\_test, y\_pred))

# Step 6: Predict new/unseen comments

new\_comments = [

"I really enjoyed the experience!",

"Such a horrible product.",

"I’m not happy with the service.",

"Best decision I ever made!"

]

new\_comments\_vect = vectorizer.transform(new\_comments)

predictions = model.predict(new\_comments\_vect)

print("\nNew Comment Predictions:")

for comment, label in zip(new\_comments, predictions):

print(f"'{comment}' --> {'Positive' if label == 1 else 'Negative'}")

**OUTPUT**

C:\Users\student\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\student\PycharmProjects\PythonProject\topdown.py

Accuracy: 0.3333333333333333

Classification Report:

precision recall f1-score support

0 0.50 0.50 0.50 2

1 0.00 0.00 0.00 1

accuracy 0.33 3

macro avg 0.25 0.25 0.25 3

weighted avg 0.33 0.33 0.33 3

New Comment Predictions:

'I really enjoyed the experience!' --> Negative

'Such a horrible product.' --> Positive

'I’m not happy with the service.' --> Negative

'Best decision I ever made!' --> Negative

Process finished with exit code 0

**RESULT**

Thus, we build a simple sentiment analysis classifier using bag-of-words or TF-IDF and Naïve Bayes successfully.