# **A BASIC NLP MODEL:TEXT CLASSIFICATION AND SENTIMENT ANALYSIS**

**TEXT CLASSIFICATION**

**Definition**

Text classification involves categorizing text into predefined categories. It is a type of supervised  machine learning where the input is a text and the output is a category.

**Applications**

* **Spam Detection**: Identifying whether an email is spam or not.
* **Sentiment Analysis**: Determining whether a text expresses a positive, negative, neutral sentiment.
* **Topic Labeling**: Assigning topics to articles (e.g., sports, politics, technology).
* **Language Detection**: Detecting the language in which a piece of text is written.
* **Intent Detection**: Understanding the user's intention in a query (e.g., booking a flight,

checking weather).

* **Chatbot, language translation**.

**Sentiment analysis**

Sentiment analysis is a natural language processing (NLP) technique used to determine the emotional tone behind a body of text. It helps to classify the sentiment expressed in the text as positive, negative, or neutral. This analysis can be applied to various types of content, such as social media posts, product reviews, customer feedback, and news articles.

**Applications**:

* **Business**: Analyzing customer feedback and reviews to improve products and services.
* **Marketing**: Monitoring brand reputation and public perception.
* **Social Media**: Evaluating the sentiment of posts and comments related to trending topics.
* **Politics**: Analyzing public sentiment on political issues or candidates.

**Step-by-Step Process for** **text classification** **using** **Naive Bayes**

**Step 1: Collecting and Preparing Data**

The first step in text classification is collecting labeled data that can be used for training the model. The data needs to have both the **text** and its corresponding **label** (such as positive/negative, spam/ham, etc.).

**Example Dataset**:



**Step 2: Preprocessing Text Data**

Before we can apply any machine learning models, we need to preprocess the text data to convert it into a numerical format.

**Common Preprocessing Steps:**

* **Tokenization**: Split the text into individual words (tokens).
* **Lowercasing**: Convert all words to lowercase to avoid case sensitivity.
* **Remove Stop Words**: Optional – Remove common words like "the", "is", "in" that may not add much meaning.
* **Stemming/Lemmatization**: Reduce words to their base form (optional for this explanation).

**Example**:

* "I love this movie" becomes [i, love, this, movie]
* "This movie is amazing" becomes [this, movie, is, amazing]
* After tokenization, the dataset looks like this:



**Step 3: Building Vocabulary**

The next step is to build a **vocabulary** (a set of unique words) from the tokenized data. This vocabulary will be used for further calculations.

**Vocabulary**

The total vocabulary size is the number of unique words across both positive and negative reviews. This is important for Laplace smoothing.

* Vocabulary:,I, love, this, movie, is, amazing, hate, terrible
* Vocabulary size = 8 (unique words)

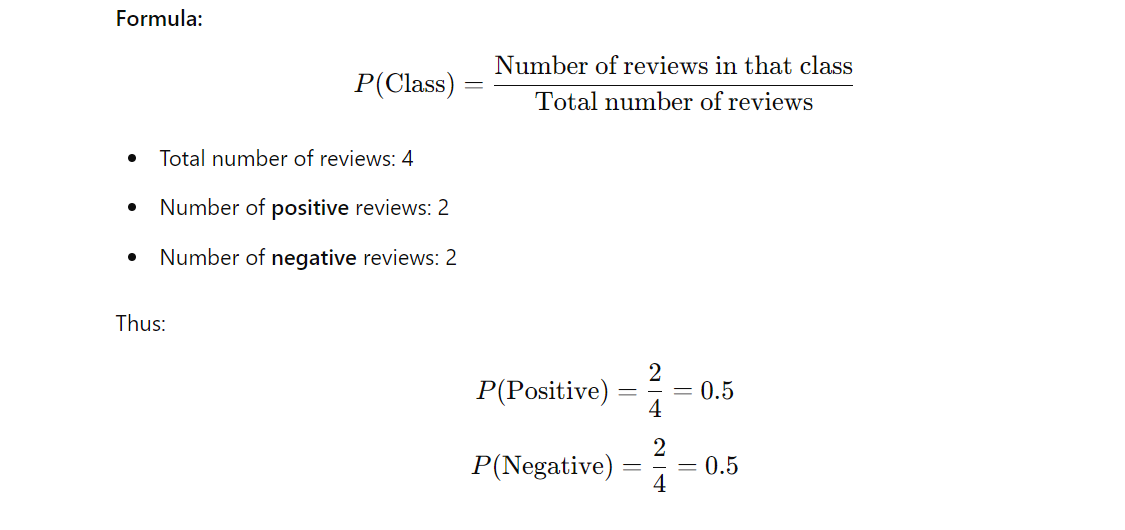
**Step 4: Word Count**

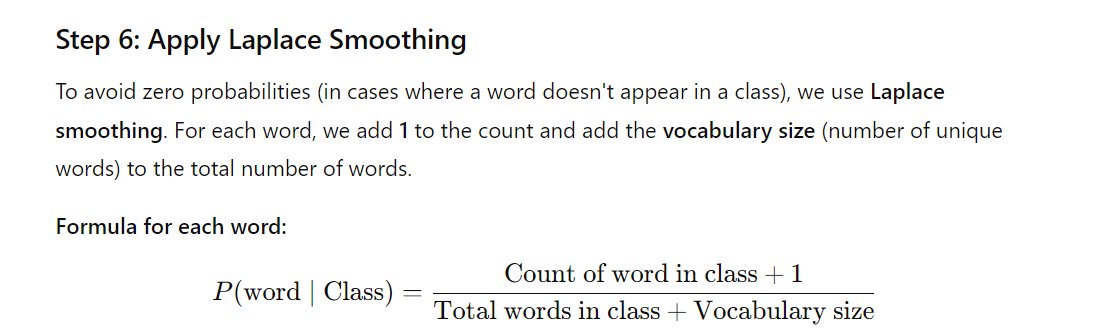
Let’s first count how many times each word appears in the **positive** and **negative** reviews:

* **Positive Reviews**:
  + "I love this movie"
  + "This movie is amazing"
  + Total words: I, love, this, movie, this, movie, is, amazing,
  + Word counts:
    - I: 1
    - love: 1
    - this: 2
    - movie: 2
    - is: 1
    - amazing: 1
  + **Total words in positive reviews** = 8
* **Negative Reviews**:
  + "I hate this movie"
  + "This movie is terrible"
  + Total words: I, hate, this, movie, this, movie, is, terrible
  + Word counts:
    - I: 1
    - hate: 1
    - this: 2
    - movie: 2
    - is: 1
    - terrible: 1
  + **Total words in negative reviews** = 8

**Step 5: Calculate Prior Probabilities**

Now, we calculate the **prior probability** for each class (positive/negative). This tells us the probability of a review being positive or negative without considering any words.

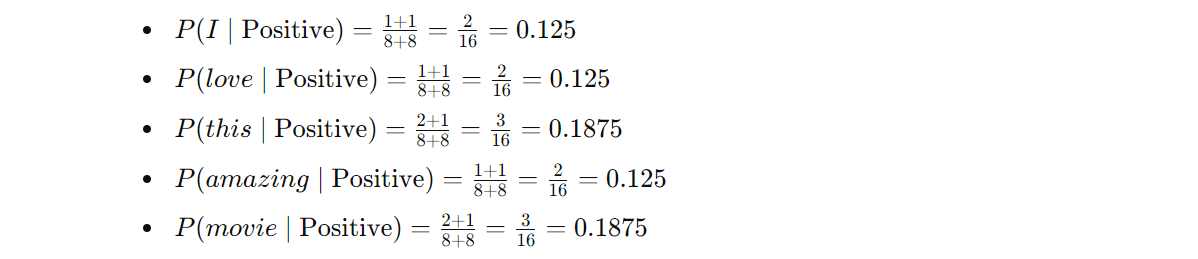




**Likelihood Calculation for the Positive Class**

We now calculate the likelihood for each word in the new review "I love this amazing movie" given the **positive class**:

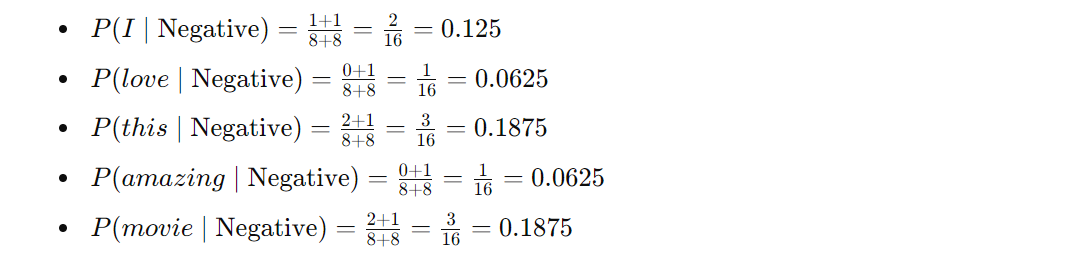
**Word by word breakdown for Positive Class:**



**Likelihood Calculation for the Negative Class**

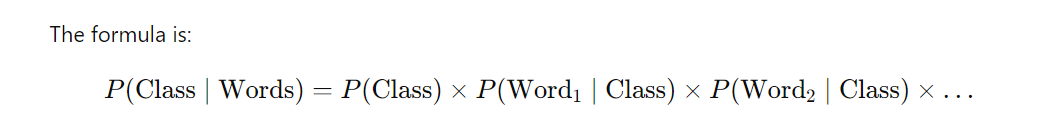
Next, we calculate the likelihood for each word in the new review "I love this amazing movie" given the **negative class**:

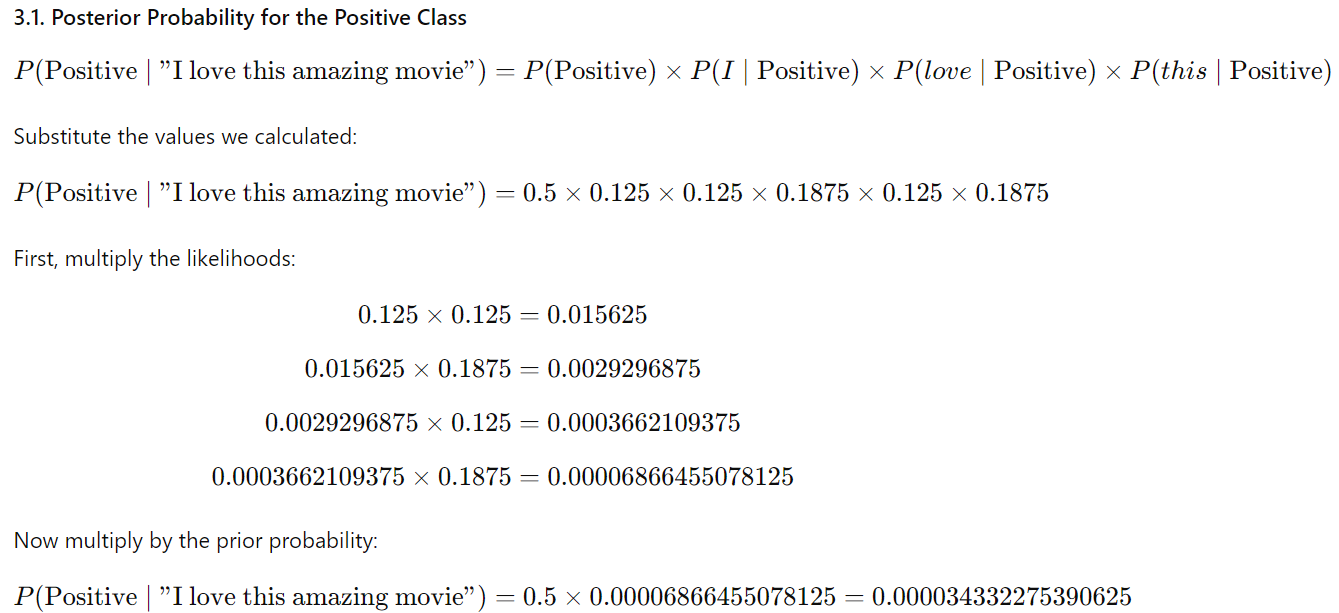
**Word by word breakdown for Negative Class**:

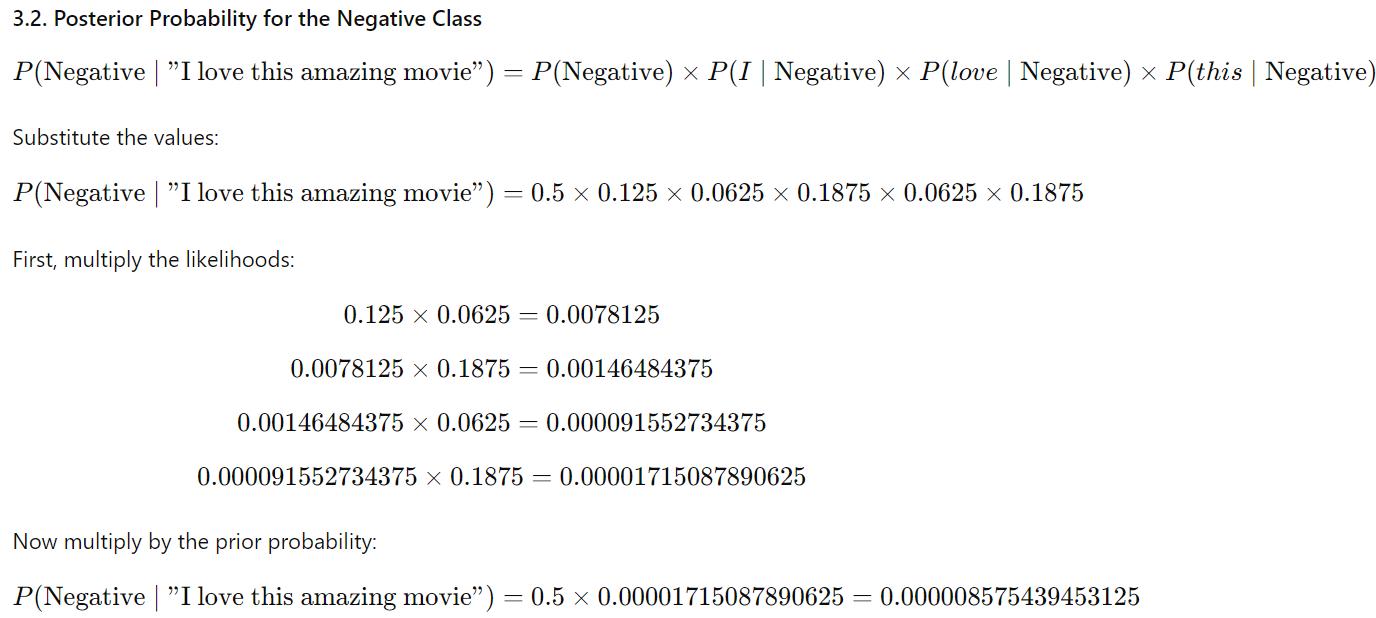


**Step 7: Posterior Probability Calculation**

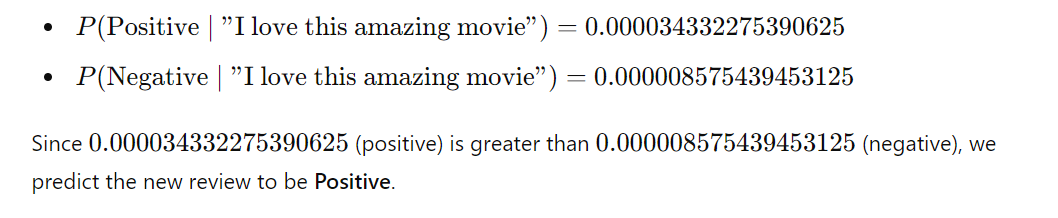
Now that we have the prior and likelihood values, we can calculate the **posterior probability** for both the positive and negative classes. The **posterior probability** is the likelihood of a class given the words in the new review.







**Step 8: Comparison of Posterior Probabilities**



# Code for Text Classification and Sentiment Analysis

# Importing necessary libraries

import pandas as pd

import re

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

from sklearn.feature\_extraction.text import TfidfVectorizer

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

from sklearn.naive\_bayes import MultinomialNB

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

# Defining the dataset

data = {

'review': [

"This movie was fantastic!",

"I hated the movie.",

"It was an amazing performance.",

"The film was boring."

],

'sentiment': [

"Positive",

"Negative",

"Positive",

"Negative"

]

}

df = pd.DataFrame(data)

# Defining stopwords

stop\_words = set(stopwords.words('english'))

# Defining text preprocessing function

def preprocess\_text(text):

text = text.lower() # Convert text to lowercase

text = re.sub(r'\W', ' ', text) # Remove punctuation

tokens = word\_tokenize(text) # Tokenize the text

tokens = [word for word in tokens if word not in stop\_words] # Remove stopwords

return ' '.join(tokens)

# Applying preprocessing to the dataset

df['cleaned\_review'] = df['review'].apply(preprocess\_text)

# Printing original and preprocessed text

for original, cleaned in zip(df['review'], df['cleaned\_review']):

print(f'Original: {original}')

print(f'Preprocessed: {cleaned}')

print()

# Vectorizing the text using TF-IDF

tfidf = TfidfVectorizer(max\_features=5000)

X = tfidf.fit\_transform(df['cleaned\_review']).toarray()

y = df['sentiment']

# Splitting the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Training the Naive Bayes model

model = MultinomialNB()

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

# Making predictions on the test set

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

# Evaluating the model

print(f'Accuracy: {accuracy\_score(y\_test, y\_pred)}')

print(classification\_report(y\_test, y\_pred))

# Predicting sentiment for a new review

new\_review = "The movie was great!"

new\_review\_cleaned = preprocess\_text(new\_review)

new\_review\_tfidf = tfidf.transform([new\_review\_cleaned]).toarray()

prediction = model.predict(new\_review\_tfidf)

print(f'Prediction: {"Positive" if prediction[0] == "Positive" else "Negative"}')

**OUTPUT**

Original: This movie was fantastic!

Preprocessed: movie fantastic

Original: I hated the movie.

Preprocessed: hated movie

Original: It was an amazing performance.

Preprocessed: amazing performance

Original: The film was boring.

Preprocessed: film boring

[[0. 0. 0. 0. 0.78528828 0.6191303

0. ]]

Accuracy: 0.0

Prediction: Positive