Code Explanation: Sentiment Analysis Using Naive Bayes

This document provides a detailed explanation of a Python code implementation that performs sentiment analysis using Natural Language Processing (NLP) and a Naive Bayes classifier. The steps include loading and preprocessing the text data, handling class imbalance, building a model pipeline, and predicting the sentiment of new comments.

**What is NLP (Natural Language Processing)?**

Natural Language Processing (NLP) is a field of artificial intelligence (AI) that focuses on the interaction between computers and human language. The goal of NLP is to enable machines to understand, interpret, and respond to human language in a way that is both meaningful and useful. It involves several tasks, including:

1. **Text Processing:** Breaking down text into smaller components (words, sentences) for analysis.
2. **Part-of-Speech Tagging:** Identifying whether a word is a noun, verb, adjective, etc.
3. **Named Entity Recognition (NER):** Identifying names of people, places, organizations, dates, etc., in a text.
4. **Sentiment Analysis:** Determining the emotional tone behind a body of text (positive, negative, neutral).
5. **Machine Translation:** Translating text from one language to another.
6. **Speech Recognition:** Converting spoken language into text.
7. **Text Summarization:** Creating a concise summary of a larger body of text.

NLP uses a combination of computational linguistics (understanding language rules), machine learning (learning patterns from data), and deep learning (using neural networks) to process and analyze large amounts of natural language data.

**Why Do We Need NLP for Sentiment Analysis?**

Sentiment analysis is the process of determining the sentiment or emotional tone expressed in a piece of text. Here’s why NLP is essential for sentiment analysis:

1. **Understanding Human Language:**
   * Human language is complex and full of nuances, such as slang, idioms, and sarcasm. NLP allows computers to process this language by converting it into a structured form that algorithms can understand. This helps identify whether the sentiment is positive, negative, or neutral.
2. **Handling Different Writing Styles:**
   * People express opinions in many different ways, with varied word choices and sentence structures. NLP techniques, like tokenization, stemming, and lemmatization, help standardize the text to make it easier to analyze consistently across different writing styles.
3. **Extracting Sentiments from Text:**
   * NLP techniques enable the identification of sentiment by analyzing the words and phrases used. For example, words like "amazing" or "excellent" are often associated with positive sentiment, while words like "boring" or "terrible" suggest negative sentiment. NLP helps detect these patterns and assign an overall sentiment score.
4. **Automating Analysis at Scale:**
   * Manually analyzing thousands of reviews is impractical. NLP automates this process, allowing sentiment analysis to be done on a large scale, such as analyzing thousands of book reviews in a matter of seconds.
5. **Dealing with Ambiguity and Context:**
   * Sentiments expressed in text can be ambiguous. For instance, "The book was surprisingly good" may appear neutral but actually conveys a positive sentiment. NLP models can understand context better by considering word relationships and sentence structure.
6. **Improving Accuracy with Advanced Models:**
   * Modern NLP models, such as BERT or GPT, use deep learning to understand the context of words in a sentence. This makes sentiment analysis more accurate, even for complex text where meaning depends on subtle context.
7. **Handling Multilingual Content:**
   * NLP can be used for sentiment analysis across multiple languages. This is useful if book reviews are written in different languages, as NLP techniques can help analyze sentiments without manual translation.

In summary, NLP is crucial for sentiment analysis because it transforms unstructured text data into structured information, making it possible to automatically detect sentiments, handle language variations, and analyze large datasets efficiently.

**Why Do We Need NLP for Sentiment Analysis?**

**1.To convert the unstructured data into structured data**

**2. Understanding Human Language  
3. Handling Different Writing Styles  
4. Extracting Sentiments from Text  
5. Improving Accuracy with Advanced Models**

|  |  |
| --- | --- |
|  |  |

# 1. Imports and Setup

First, we import the necessary libraries for data manipulation, text preprocessing, machine learning, and handling class imbalances. Here's a breakdown of the libraries used:  
- **pandas:** To handle data in tabular form (DataFrames).  
- **nltk:** Natural Language Toolkit for text processing tasks such as tokenization and lemmatization.  
- **sklearn:** Machine learning library providing tools for model building, training, and evaluation.  
- **imblearn:** A library that helps manage imbalanced datasets by oversampling the minority class.

# 2. Downloading NLTK Resources

Before we can use NLTK's text processing tools, we need to download specific resources:  
- punkt: A tokenizer to split text into words.  
- stopwords: A list of common English words that are often filtered out in NLP tasks.  
- wordnet: A lexical database used for lemmatizing words (i.e., reducing words to their base form).

# 3. Loading the Dataset

The dataset is loaded using pandas from a CSV file, and stored in a DataFrame. The dataset contains two key columns: comments (text) and their corresponding sentiment labels.

# 4. Text Preprocessing

We define a function called `preprocess\_text` to clean and prepare the text for analysis. This function:  
- Converts the text to lowercase.  
- Tokenizes the text (splits it into individual words).  
- Removes stopwords (common words that do not add much meaning, such as 'the', 'is').  
- Lemmatizes the remaining words, reducing them to their base form (e.g., 'running' becomes 'run').

# 5. Preprocessing the Comments

The `preprocess\_text` function is applied to each comment in the dataset. The cleaned and processed text is stored in a new column called 'Processed\_Comment'.

# 6. Handling Class Imbalance

In many datasets, there is often an imbalance in the classes (e.g., more positive comments than negative ones). We use **RandomOverSampler** to oversample the minority class to ensure that both classes have equal representation. This helps improve model performance and avoids bias toward the majority class.

# 7. Splitting Data into Training and Testing Sets

The dataset is split into two parts: a training set (80%) used to train the model, and a testing set (20%) used to evaluate model performance. The splitting is done using `train\_test\_split` from sklearn.

# 8. Building a Pipeline

A pipeline is created to streamline the machine learning process. The pipeline consists of two key components:  
- TF-IDF Vectorizer: Converts the cleaned text into a numerical representation. It uses n-grams (both unigrams and bigrams) to capture the importance of words or phrases.  
- Multinomial Naive Bayes: A classifier that is well-suited for text classification tasks.

# 9. Training the Model

The pipeline is trained on the training data using the `fit` method. During training, the model learns to associate certain word patterns with positive or negative sentiments.

# 10. Evaluating the Model

Once the model is trained, we use the testing set to evaluate its performance. The model predicts the sentiments of the test data, and we print a classification report showing metrics like precision, recall, and F1-score.

# 11. Predicting Sentiment for New Comments

The code allows for predicting the sentiment of new comments by taking user input, preprocessing it, and passing it through the trained model. The predicted sentiment is then displayed.

# Code

import pandas as pd  
import nltk  
from sklearn.model\_selection import train\_test\_split  
from sklearn.feature\_extraction.text import TfidfVectorizer  
from sklearn.naive\_bayes import MultinomialNB  
from sklearn.pipeline import Pipeline  
from sklearn.metrics import classification\_report  
from nltk.corpus import stopwords  
from nltk.tokenize import word\_tokenize  
from nltk.stem import WordNetLemmatizer  
from imblearn.over\_sampling import RandomOverSampler  
  
# Ensure necessary NLTK packages are downloaded  
nltk.download('punkt')  
nltk.download('stopwords')  
nltk.download('wordnet')  
  
# Load the dataset  
data = pd.read\_csv('/content/sorna.csv') # Update the path to your dataset  
  
# Initialize the lemmatizer  
lemmatizer = WordNetLemmatizer()  
  
# Text preprocessing function  
def preprocess\_text(text):  
 words = word\_tokenize(text.lower())  
 stop\_words = set(stopwords.words('english'))  
 words = [lemmatizer.lemmatize(word) for word in words if word.isalpha() and word not in stop\_words]  
 return ' '.join(words)  
  
# Preprocess the comments  
data['Processed\_Comment'] = data['Comment'].apply(preprocess\_text)  
  
# Display class distribution before oversampling  
print("Class distribution before oversampling:")  
print(data['Sentiment'].value\_counts())  
  
# Split features and labels  
X = data['Processed\_Comment']  
y = data['Sentiment']  
  
# Oversample to handle class imbalance  
ros = RandomOverSampler(random\_state=42)  
X\_resampled, y\_resampled = ros.fit\_resample(X.values.reshape(-1, 1), y)  
  
# Convert back to series for processing  
X\_resampled = pd.Series([x[0] for x in X\_resampled])  
  
# Display class distribution after oversampling  
print("  
Class distribution after oversampling:")  
print(y\_resampled.value\_counts())  
  
# Split the dataset into training and testing sets  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_resampled, y\_resampled, test\_size=0.2, random\_state=42)  
  
# Build the TF-IDF + Naive Bayes pipeline  
model\_pipeline = Pipeline([  
 ('tfidf', TfidfVectorizer(ngram\_range=(1, 2))),  
 ('classifier', MultinomialNB())  
])  
  
# Train the model  
model\_pipeline.fit(X\_train, y\_train)  
  
# Evaluate the model  
y\_pred = model\_pipeline.predict(X\_test)  
print("  
Classification Report:")  
print(classification\_report(y\_test, y\_pred))  
  
# Predict the sentiment of a sample review  
sample\_comment = input("Enter a book review comment: ")  
processed\_sample = preprocess\_text(sample\_comment)  
predicted\_sentiment = model\_pipeline.predict([processed\_sample])  
print(f"  
The predicted sentiment for the comment is: {predicted\_sentiment[0]}")