

```
import nltk from nltk.tokenize import word_tokenize from nltk.corpus import stopwords from nltk.probability import FreqDist from nltk.classify import NaiveBayesClassifier
```

ChatGPT

**ChatGPT**

Sure! Let's break down the code step by step:

1. `import nltk`: This imports the Natural Language Toolkit (NLTK) library, which is a popular library for working with human language data in Python.
2. `from nltk.tokenize import word_tokenize`: This line imports the `word_tokenize` function from the `tokenize` module of NLTK. `word_tokenize` is used to split a sentence into individual words.
3. `from nltk.corpus import stopwords`: This line imports the `stopwords` module from NLTK's `corpus` package. Stopwords are common words (like "the", "is", "and", etc.) that are often filtered out from text data because they typically don't carry much meaning.
4. `from nltk.probability import FreqDist`: This line imports the `FreqDist` class from the `probability` module of NLTK. `FreqDist` is used to find the frequency distribution of words in a text.
5. `from nltk.classify import NaiveBayesClassifier`: This line imports the `NaiveBayesClassifier` class from the `classify` module of NLTK. Naive Bayes is a popular machine learning algorithm used for classification tasks, including text classification.

The code you've provided imports necessary components from NLTK for text processing and classification tasks. It sets up the environment for tasks like tokenization (breaking text into words), filtering out stopwords, analyzing word frequencies, and training a Naive Bayes classifier.

DA

**You**

```
positive_documents = ["The movie was great and highly enjoyable.", "I loved the book; it was fantastic."]
negative_documents = ["The concert was terrible and disappointing.", "The service at the restaurant was awful."]
# Combine positive and negative documents into one list
documents = [(doc, "Positive") for doc in positive_documents] + [(doc, "Negative") for doc in negative_documents]
```

ChatGPT

**ChatGPT**

This part of the code is preparing the data for sentiment analysis using a Naive Bayes classifier. Here's what each line does:

1. `positive_documents` and `negative_documents`: These are lists containing example sentences expressing positive and negative sentiments, respectively. Each sentence represents a document in our dataset.
2. `documents`: This line combines the positive and negative documents into one list of tuples. Each tuple consists of two elements: the document itself (a string) and its corresponding sentiment label ("Positive" or "Negative"). This format is commonly used in supervised machine learning tasks, where each data point is paired with its label.

So, after executing this code, the `documents` list will contain all the example sentences along with their sentiment labels, ready to be used for training a Naive Bayes classifier to classify new documents as either positive or negative based on their content.

DA

**You**

```
nltk.download('punkt') nltk.download('stopwords') # Tokenize and preprocess the documents
all_words = [] for document, sentiment in documents: words = word_tokenize(document) words
= [word.lower() for word in words if word.isalpha() and word.lower() not in
stopwords.words('english')] all_words.extend(words) # Extract the most common words as
features word_features = FreqDist(all_words).most_common(100) word_features = [word for
word, _ in word_features]
```

ChatGPT

**ChatGPT**

This part of the code tokenizes and preprocesses the documents to extract features for sentiment analysis. Let's go through each step:

1. `nltk.download('punkt')` and `nltk.download('stopwords')`: These lines download the necessary NLTK resources for tokenization and stopwords removal. The `punkt` resource provides a tokenizer, while the `stopwords` resource contains common stopwords that should be filtered out from the text.
2. Tokenization and preprocessing loop:
  - The loop iterates over each document in the `documents` list, which consists of tuples `(document, sentiment)`.
  - For each document, it tokenizes the text into individual words using the `word_tokenize` function.
  - It converts each word to lowercase (`word.lower()`) and filters out non-alphabetic words (`word.isalpha()`) and stopwords using list comprehension.
  - The filtered words are then added to the `all_words` list.
3. Extracting most common words as features:
  - After processing all documents, the code creates a frequency distribution (`FreqDist`) of all the words in `all_words`.
  - It then selects the 100 most common words as features using the `most_common(100)` method.
  - Finally, it extracts only the words (ignoring their frequencies) and stores them in the `word_features` list.

So, `word_features` will contain the 100 most common words across all documents after preprocessing, which will be used as features for training the Naive Bayes classifier. These features represent the most informative words in the dataset for distinguishing between positive and negative sentiments.

DA

**You**

```
def document_features(document): document_words = set(word_tokenize(document.lower()))
features = {} for word in word_features: features['contains({})'.format(word)] = (word in
document_words) return features # Generate feature sets for the documents feature_sets =
[(document_features(doc), sentiment) for doc, sentiment in documents]
```

ChatGPT

**ChatGPT**

This code defines a function `document_features` that generates feature sets for each document. Here's what each part of the code does:

1. **def document\_features(document):**: This line defines a function named **document\_features** that takes a single argument **document**, representing a single document (text).
2. **document\_words = set(word\_tokenize(document.lower()))**: This line tokenizes the input document into individual words, converts them to lowercase, and stores them in a set. Using a set ensures that each word appears only once in the set, removing duplicates.
3. **features = {}**: This line initializes an empty dictionary named **features** to store the features of the document.
4. Feature extraction loop:
  - This loop iterates over each word in the **word\_features** list (which contains the 100 most common words extracted earlier).
  - For each word, it checks if that word is present in the set of **document\_words**. If the word is present, it assigns a value of **True** to the feature named **'contains(word)'**; otherwise, it assigns **False**.
5. **return features**: This line returns the dictionary **features**, which contains the feature set for the input document.
6. Generating feature sets:
  - This line creates **feature\_sets**, a list of tuples where each tuple consists of two elements:
    - The first element is the feature set generated by the **document\_features** function for a specific document.
    - The second element is the sentiment label associated with that document (either "Positive" or "Negative").

After executing this code, **feature\_sets** will contain a list of feature sets, where each feature set represents a document along with its associated sentiment label. These feature sets are ready to be used for training the Naive Bayes classifier. Each feature set consists of features indicating whether each of the top 100 most common words is present in the document or not.

DA

**You**

```
classifier = NaiveBayesClassifier.train(feature_sets)
```

ChatGPT

**ChatGPT**

This line of code trains a Naive Bayes classifier using the feature sets generated from the documents. Here's what it does:

1. **NaiveBayesClassifier.train(feature\_sets)**: This line uses the **train** method of the **NaiveBayesClassifier** class to train a Naive Bayes classifier. The **train** method expects a list of labeled feature sets as input, which is exactly what **feature\_sets** contains.
  - Each feature set in **feature\_sets** represents a document along with its associated sentiment label.
  - During training, the classifier learns from these labeled feature sets to make predictions about the sentiment of unseen documents based on their features.

After executing this line of code, the variable **classifier** will hold a trained Naive Bayes classifier that can be used to classify new documents into positive or negative sentiments based on their features.

DA

## You

```
# Example document to classify new_document = "The movie was terrible and boring." #
Preprocess and extract features from the new document new_features =
document_features(new_document) # Classify the new document using the trained classifier
classification = classifier.classify(new_features) print("Document sentiment:", classification)
```

ChatGPT

## ChatGPT

This code segment demonstrates how to classify a new document using the trained Naive Bayes classifier. Here's a breakdown:

1. `new_document = "The movie was terrible and boring."`: This line defines a new document (a string) that we want to classify.
2. `new_features = document_features(new_document)`: This line preprocesses and extracts features from the new document using the `document_features` function defined earlier. It generates a feature set for the new document based on the top 100 most common words extracted from the training data.
3. `classification = classifier.classify(new_features)`: This line classifies the new document using the trained Naive Bayes classifier (`classifier`). It uses the `classify` method of the classifier, passing the feature set (`new_features`) of the new document as input. The classifier predicts the sentiment of the new document based on its features.
4. `print("Document sentiment:", classification)`: This line prints the predicted sentiment of the new document to the console. The predicted sentiment will be either "Positive" or "Negative" based on the trained classifier's decision.

After executing this code, you'll get the predicted sentiment of the new document printed to the console.