Home > Tutorials > Python

NLTK Sentiment Analysis Tutorial for Beginners

Python NLTK (natural language toolkit) sentiment analysis tutorial. Learn how to create and develop sentiment analysis using Python. Follow specific steps to mine and analyze text for natural language processing.

Updated Mar 23, 2023 · 13 min read

Moez Ali

Data Scientist, Founder & Creator of PyCaret

TOPICS

Python

Artificial Intelligence

In today's digital age, text analysis and text mining have become essential parts of various industries. Text analysis refers to the process of analyzing and extracting meaningful insights from unstructured text data. One of the most important subfields of text analysis is sentiment analysis, which involves determining the emotional tone of the text.

Sentiment analysis has numerous practical applications, from brand monitoring to customer feedback analysis. Python is a popular programming language used for text analysis and mining, and the Natural Language Toolkit (NLTK) library is one of the most widely used libraries for natural language processing in Python.

This tutorial will provide a step-by-step guide for performing sentiment analysis using the NLTK library in Python. By the end of this tutorial, you will have a solid understanding of how to perform sentiment analysis using NLTK in Python, along with a complete example that you can use as a starting point for your own projects. So, let's get started!

The Natural Language Toolkit (NLTK) Library

The Natural Language Toolkit (NLTK) is a popular open-source library for natural language processing (NLP) in Python. It provides an easy-to-use interface for a wide range of tasks, including tokenization, stemming, lemmatization, parsing, and sentiment analysis.

NLTK is widely used by researchers, developers, and data scientists worldwide to develop NLP applications and analyze text data.

One of the major advantages of using NLTK is its extensive collection of corpora, which includes text data from various sources such as books, news articles, and social media platforms. These corpora provide a rich data source for training and testing NLP models.

What is Sentiment Analysis

Sentiment analysis is a technique used to determine the emotional tone or sentiment expressed in a text. It involves analyzing the words and phrases used in the text to identify the underlying sentiment, whether it is positive, negative, or neutral.

Sentiment analysis has a wide range of applications, including social media monitoring, customer feedback analysis, and market research.

One of the main challenges in sentiment analysis is the inherent complexity of human language. Text data often contains sarcasm, irony, and other forms of figurative language that can be difficult to interpret using traditional methods.

However, recent advances in natural language processing (NLP) and machine learning have made it possible to perform sentiment analysis on large volumes of text data with a high degree of accuracy.

Three Methodologies for Sentiment Analysis

There are several ways to perform sentiment analysis on text data, with varying degrees of complexity and accuracy. The most common methods include a lexicon-based approach, a machine learning (ML) based approach, and a pre-trained transformer-based deep learning approach. Let's look at each in more detail:

Lexicon-based analysis

While lexicon-based analysis can be relatively simple to implement and interpret, it may not be as accurate as ML-based or transformed-based approaches, especially when dealing with complex or ambiguous text data.

Machine learning (ML)

This approach involves training a model to identify the sentiment of a piece of text based on a set of labeled training data. These models can be trained using a wide range of **ML algorithms**, including decision trees, support vector machines (SVMs), and neural networks.

ML-based approaches can be more accurate than rule-based analysis, especially when dealing with complex text data, but they require a larger amount of labeled training data and may be more computationally expensive.

Pre-trained transformer-based deep learning

A deep learning-based approach, as seen with BERT and **GPT-4**, involve using pre-trained models trained on massive amounts of text data. These models use complex neural networks to encode the context and meaning of the text, allowing them to achieve state-of-the-art accuracy on a wide range of NLP tasks, including sentiment analysis. However, these models require significant computational resources and may not be practical for all use cases.

- **Lexicon-based analysis** is a straightforward approach to sentiment analysis, but it may not be as accurate as more complex methods.
- **Machine learning-based** approaches can be more accurate, but they require labeled training data and may be more

computationally expensive.

- **Pre-trained transformer-based deep learning** approaches can achieve state-of-the-art accuracy but require significant computational resources and may not be practical for all use cases.

The choice of approach will depend on the specific needs and constraints of the project at hand.

Installing NLTK and Setting up Python Environment

To use the NLTK library, you must have a Python environment on your computer. The easiest way to install Python is to download and install the Anaconda Distribution. This distribution comes with the Python 3 base environment and other bells and whistles, including Jupyter Notebook. You also do not need to install the NLTK library, as it comes pre-installed with NLTK and many other useful libraries.

If you choose to install Python without any distribution, you can directly download and install Python from python.org. In this case, you will have to install NLTK once your Python environment is ready.

To install NLTK library, open the command terminal and type:

```
pip install nltk
```

Explain code

POWERED BY datalab

It's worth noting that NLTK also requires some additional data to be downloaded before it can be used effectively. This data includes pre-trained models, corpora, and other resources that NLTK uses to perform various NLP tasks. To download this data, run the following command in terminal or your Python script:

```
import nltk

nltk.download('all')
```

Explain code

POWERED BY datalab

Preprocessing Text

Text preprocessing is a crucial step in performing sentiment analysis, as it helps to clean and normalize the text data, making it easier to analyze. The preprocessing step involves a series of techniques that help transform raw text data into a form you can use for analysis. Some common text preprocessing techniques include tokenization, stop word removal, stemming, and lemmatization.

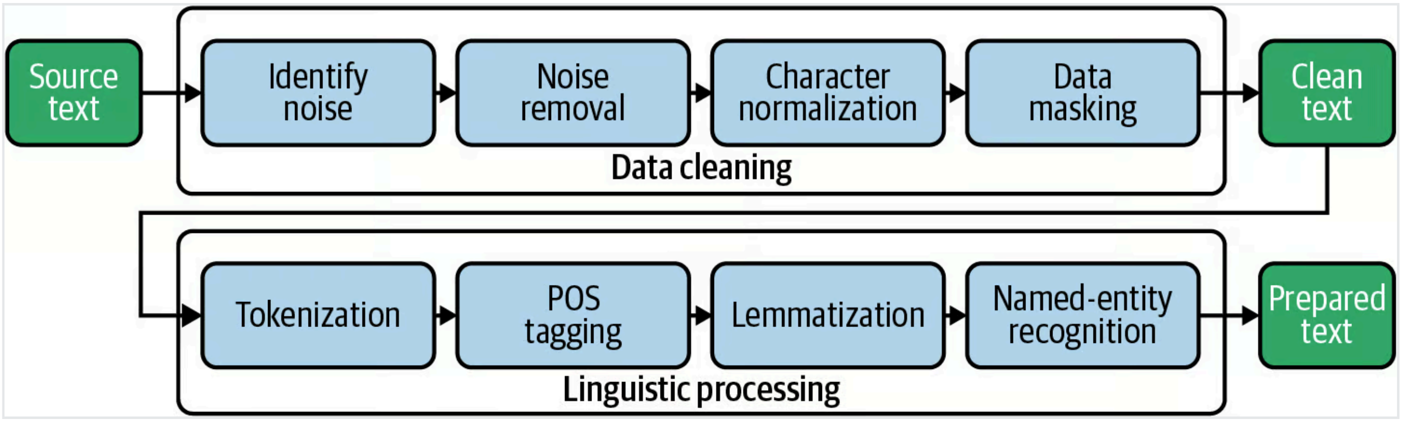


Image Source

Tokenization

Tokenization is a text preprocessing step in sentiment analysis that involves breaking down the text into individual words or tokens. This is an essential step in analyzing text data as it helps to separate individual words from the raw text, making it easier to analyze and understand. Tokenization is typically performed using NLTK's built-in `word_tokenize` function, which can split the text into individual words and punctuation marks.

Stop words

Stop word removal is a crucial text preprocessing step in sentiment analysis that involves removing common and irrelevant words that are unlikely to convey much sentiment. Stop words are words that are very common in a language and do not carry much meaning, such as "and," "the," "of," and "it." These words can cause noise and skew the analysis if they are not removed.

By removing stop words, the remaining words in the text are more likely to indicate the sentiment being expressed. This can help to improve the accuracy of the sentiment analysis. NLTK provides a built-in list of stop words for several languages, which can be used to filter out these words from the text data.

Stemming and Lemmatization

Stemming and lemmatization are techniques used to reduce words to their root forms. Stemming involves removing the suffixes from words, such as "ing" or "ed," to reduce them to their base form. For example, the word "jumping" would be stemmed to "jump."

Lemmatization, however, involves reducing words to their base form based on their part of speech. For example, the word "jumped" would be lemmatized to "jump," but the word "jumping" would be lemmatized to "jumping" since it is a present participle.

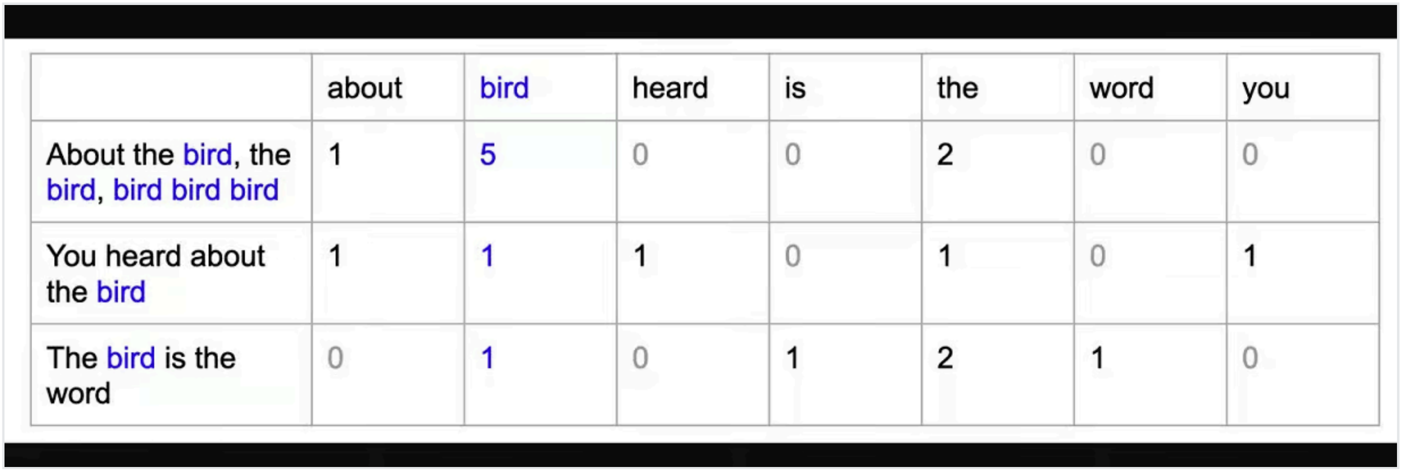
To learn more about stemming and lemmatization, check out our Stemming and Lemmatization in Python tutorial.

Bag of Words (BoW) Model

The bag of words model is a technique used in natural language processing (NLP) to represent text data as a set of numerical features. In this model, each document or piece of text is represented as a "bag" of words, with each word in the text represented by a separate feature or dimension in the resulting vector. The value of each feature is determined by the number of times the corresponding word appears in the text.

The bag of words model is useful in NLP because it allows us to analyze text data using machine learning algorithms, which typically require numerical input. By representing text data as numerical features, we can train machine learning models to classify text or analyze sentiments.

The example in the next section will use the NLTK Vader model for sentiment analysis on the Amazon customer dataset. In this particular example, we do not need to perform this step because the NLTK Vader API accepts text as an input instead of numeric vectors, but if you were building a supervised machine learning model to predict sentiment (assuming you have labeled data), you would have to transform the processed text into a bag of words model before training the machine learning model.



	about	bird	heard	is	the	word	you
About the bird, the bird, bird bird bird	1	5	0	0	2	0	0
You heard about the bird	1	1	1	0	1	0	1
The bird is the word	0	1	0	1	2	1	0

Image Source

End-to-end Sentiment Analysis Example in Python

To perform sentiment analysis using NLTK in Python, the text data must first be preprocessed using techniques such as tokenization, stop word removal, and stemming or lemmatization. Once the text has been preprocessed, we will then pass it to the Vader sentiment analyzer for analyzing the sentiment of the text (positive or negative).

Step 1 - Import libraries and load dataset

First, we'll import the necessary libraries for text analysis and sentiment analysis, such as `pandas` for data handling, `nltk` for natural language processing, and `SentimentIntensityAnalyzer` for sentiment analysis.

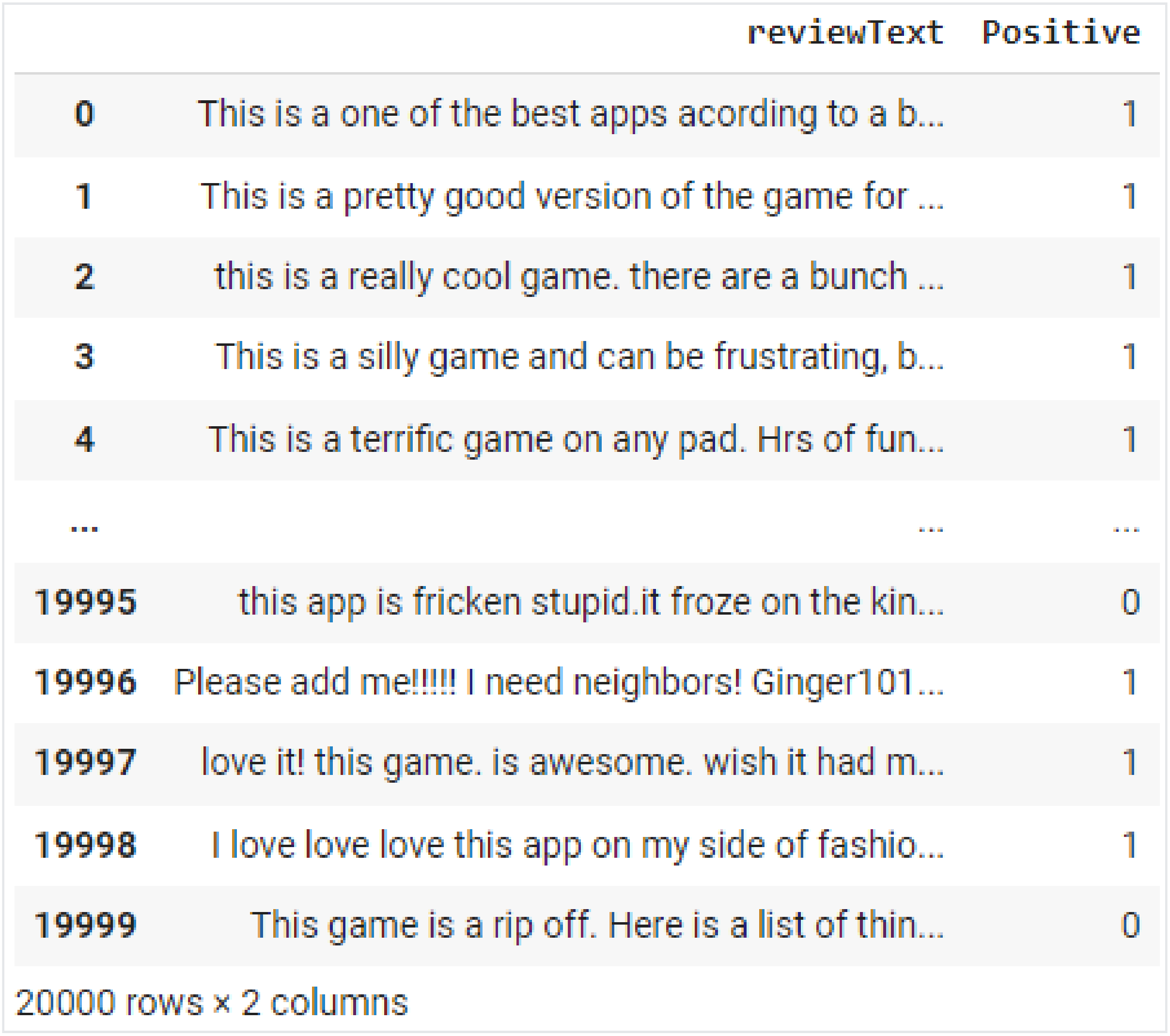
We'll then download all of the NLTK corpus (a collection of linguistic data) using `nltk.download()`.

Once the environment is set up, we will load a dataset of Amazon reviews using `pd.read_csv()`. This will create a DataFrame object in

Python that we can use to analyze the data. We'll display the contents of the DataFrame using df .

```
# import libraries
import pandas as pd

import nltk

from nltk.sentiment.vader import SentimentIntensityAnalyzer

from nltk.corpus import stopwords

from nltk.tokenize import word_tokenize

from nltk.stem import WordNetLemmatizer


# download nltk corpus (first time only)
import nltk

nltk.download('all')




# Load the amazon review dataset

df = pd.read_csv('https://raw.githubusercontent.com/pycaret/p

df
```

Explain code POWERED BY datalab



	reviewText	Positive
0	This is a one of the best apps acording to a b...	1
1	This is a pretty good version of the game for ...	1
2	this is a really cool game. there are a bunch ...	1
3	This is a silly game and can be frustrating, b...	1
4	This is a terrific game on any pad. Hrs of fun...	1
...	...	...
19995	this app is fricken stupid.it froze on the kin...	0
19996	Please add me!!!!! I need neighbors! Ginger101...	1
19997	love it! this game. is awesome. wish it had m...	1
19998	I love love love this app on my side of fashio...	1
19999	This game is a rip off. Here is a list of thin...	0

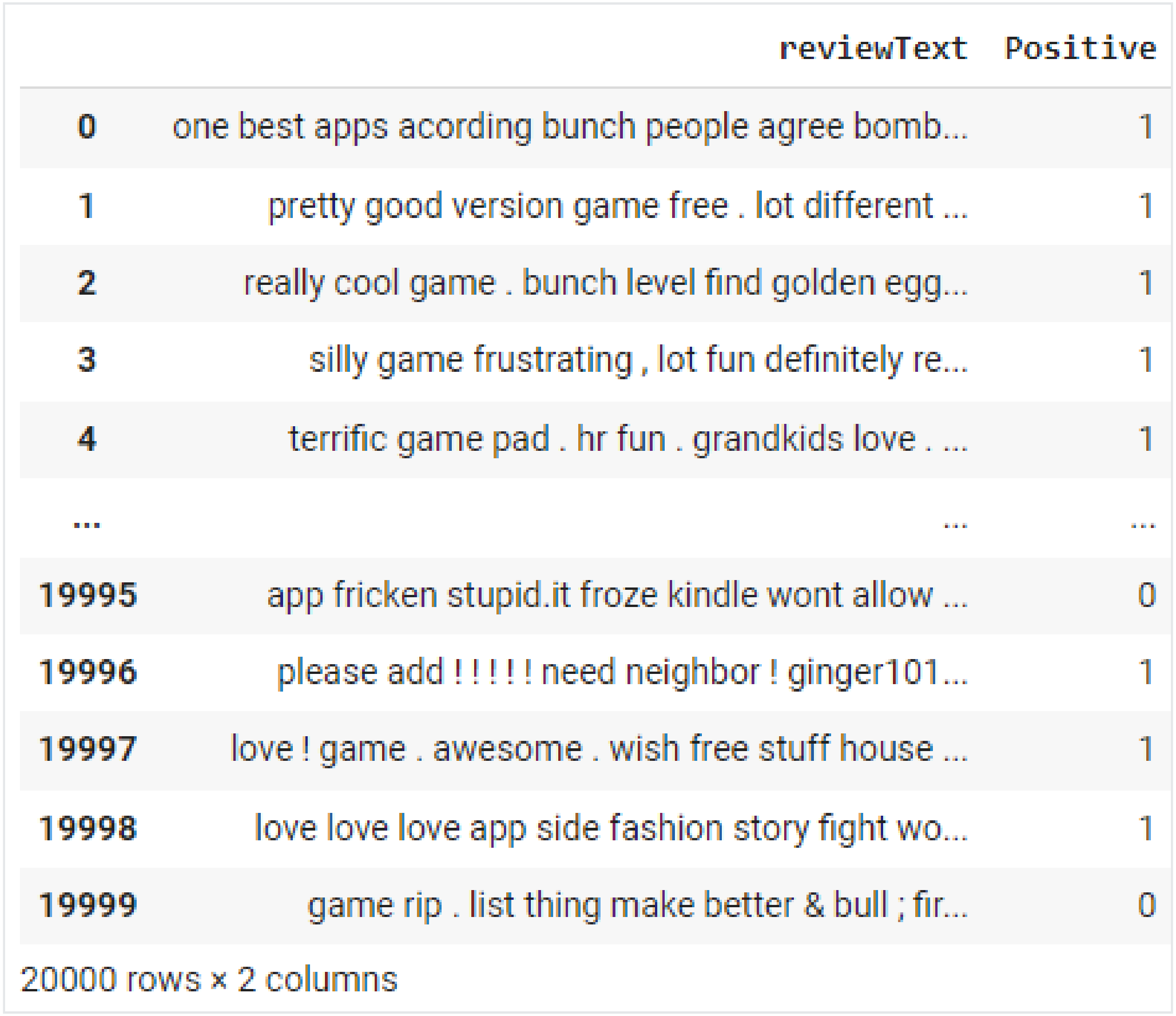
20000 rows × 2 columns

Step 2 - Preprocess text

Let's create a function preprocess_text in which we first tokenize the documents using word_tokenize function from NLTK, then we remove step words using stepwords module from NLTK and finally, we lemmatize the filtered_tokens using WordNetLemmatizer from NLTK.

```
# create preprocess_text function
def preprocess_text(text):

    # Tokenize the text
    tokens = word_tokenize(text.lower())



    # Remove stop words

    filtered_tokens = [token for token in tokens if token not



    # Lemmatize the tokens

    lemmatizer = WordNetLemmatizer()

    lemmatized_tokens = [lemmatizer.lemmatize(token) for toke



    # Join the tokens back into a string

    processed_text = ' '.join(lemmatized_tokens)

    return processed_text

# apply the function df

df['reviewText'] = df['reviewText'].apply(preprocess_text)
df
```

Explain code POWERED BY datalab



	reviewText	Positive
0	one best apps acording bunch people agree bomb...	1
1	pretty good version game free . lot different ...	1
2	really cool game . bunch level find golden egg...	1
3	silly game frustrating , lot fun definitely re...	1
4	terrific game pad . hr fun . grandkids love	1
...	...	...
19995	app fricken stupid.it froze kindle wont allow ...	0
19996	please add ! ! ! ! ! need neighbor ! ginger101...	1
19997	love ! game . awesome . wish free stuff house ...	1
19998	love love love app side fashion story fight wo...	1
19999	game rip . list thing make better & bull ; fir...	0

20000 rows × 2 columns

Notice the changes in the "review text" column as a result of the preprocess_text function that we applied in the above step.

Step 3 - NLTK Sentiment Analyzer

First, we'll initialize a Sentiment Intensity Analyzer object from the nltk.sentiment.vader library.

Next, we'll define a function called get_sentiment that takes a text string as its input. The function calls the polarity_scores method of the analyzer object to obtain a dictionary of sentiment scores for the text, which includes a score for positive, negative, and neutral sentiment.

The function will then check whether the positive score is greater than 0 and returns a sentiment score of 1 if it is, and a 0 otherwise. This means that any text with a positive score will be classified as having a positive sentiment, and any text with a non-positive score will be classified as having a negative sentiment.

Finally, we'll apply the get_sentiment function to the reviewText column of the df DataFrame using the apply method. This creates a new column called sentiment in the DataFrame, which stores the sentiment score for each review. We'll then display the updated DataFrame using df .

```
# initialize NLTK sentiment analyzer

analyzer = SentimentIntensityAnalyzer()


# create get_sentiment function

def get_sentiment(text):

    scores = analyzer.polarity_scores(text)

    sentiment = 1 if scores['pos'] > 0 else 0

    return sentiment




# apply get_sentiment function

df['sentiment'] = df['reviewText'].apply(get_sentiment)
```

```
df
```

Explain code POWERED BY datalab

	reviewText	Positive	sentiment
0	one best apps acording bunch people agree bomb...	1	1
1	pretty good version game free . lot different ...	1	1
2	really cool game . bunch level find golden egg...	1	1
3	silly game frustrating , lot fun definitely re...	1	1
4	terrific game pad . hr fun . grandkids love	1	1
...	...	...	...
19995	app fricken stupid.it froze kindle wont allow ...	0	0
19996	please add ! ! ! ! ! need neighbor ! ginger101...	1	1
19997	love ! game . awesome . wish free stuff house ...	1	1
19998	love love love app side fashion story fight wo...	1	1
19999	game rip . list thing make better & bull ; fir...	0	1

20000 rows × 3 columns

The NLTK sentiment analyzer returns a score between -1 and +1. We have used a cut-off threshold of 0 in the `get_sentiment` function above. Anything above 0 is classified as 1 (meaning positive). Since we have actual labels, we can evaluate the performance of this method by building a confusion matrix.

```
from sklearn.metrics import confusion_matrix
```

```
print(confusion_matrix(df['Positive'], df['sentiment']))
```

Explain code POWERED BY datalab

Output:

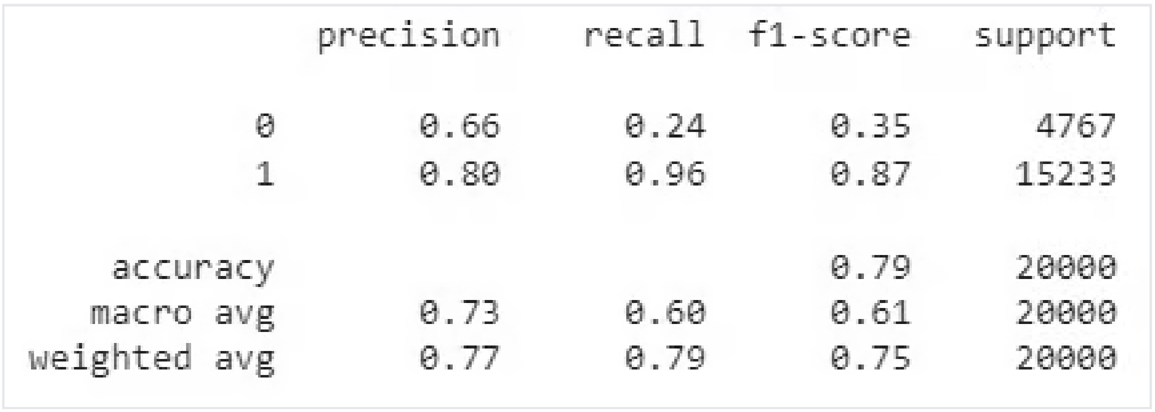
```
[[ 1131  3636]
 [  576 14657]]
```

Explain code POWERED BY datalab

We can also check the classification report:

```
from sklearn.metrics import classification_report
print(classification_report(df['Positive'], df['sentiment'])
```

Explain code POWERED BY datalab



```
              precision    recall  f1-score   support

           0       0.66      0.24      0.35      4767
           1       0.80      0.96      0.87     15233

    accuracy                           0.79     20000
   macro avg       0.73      0.60      0.61     20000
weighted avg       0.77      0.79      0.75     20000
```

As you can see, the overall accuracy of this rule-based sentiment analysis model is 79%. Since this is labeled data, you can also try to build a ML model to evaluate if an ML-based approach will result in better accuracy.

Check out the full notebook on the Datacamp workspace.

Conclusion

NLTK is a powerful and flexible library for performing sentiment analysis and other natural language processing tasks in Python. By using NLTK, we can preprocess text data, convert it into a bag of words model, and perform sentiment analysis using Vader's sentiment analyzer.

Through this tutorial, we have explored the basics of NLTK sentiment analysis, including preprocessing text data, creating a bag of words model, and performing sentiment analysis using NLTK Vader. We have also discussed the advantages and limitations of NLTK sentiment analysis, and provided suggestions for further reading and exploration.

Overall, NLTK is a powerful and widely used tool for performing sentiment analysis and other natural language processing tasks in Python. By mastering the techniques and tools presented in this tutorial, you can gain valuable insights into the sentiment of text data and use these insights to make data-driven decisions in a wide range of applications.

If you want to learn how to apply NLP to real-world data, including TED talks, articles, and movie reviews, using Python libraries and frameworks, including NLTK, scikit-learn, spaCy, and SpeechRecognition, check out the resources below:

- Introduction to Natural Language Processing in Python
- Natural Language Processing in Python

It provides a strong foundation for processing and analyzing text data using Python. Whether you're new to NLP or looking to expand your skills, this course will equip you with the tools and knowledge to convert unstructured data into valuable insights.

TOPICS

Python Artificial Intelligence

Python Courses