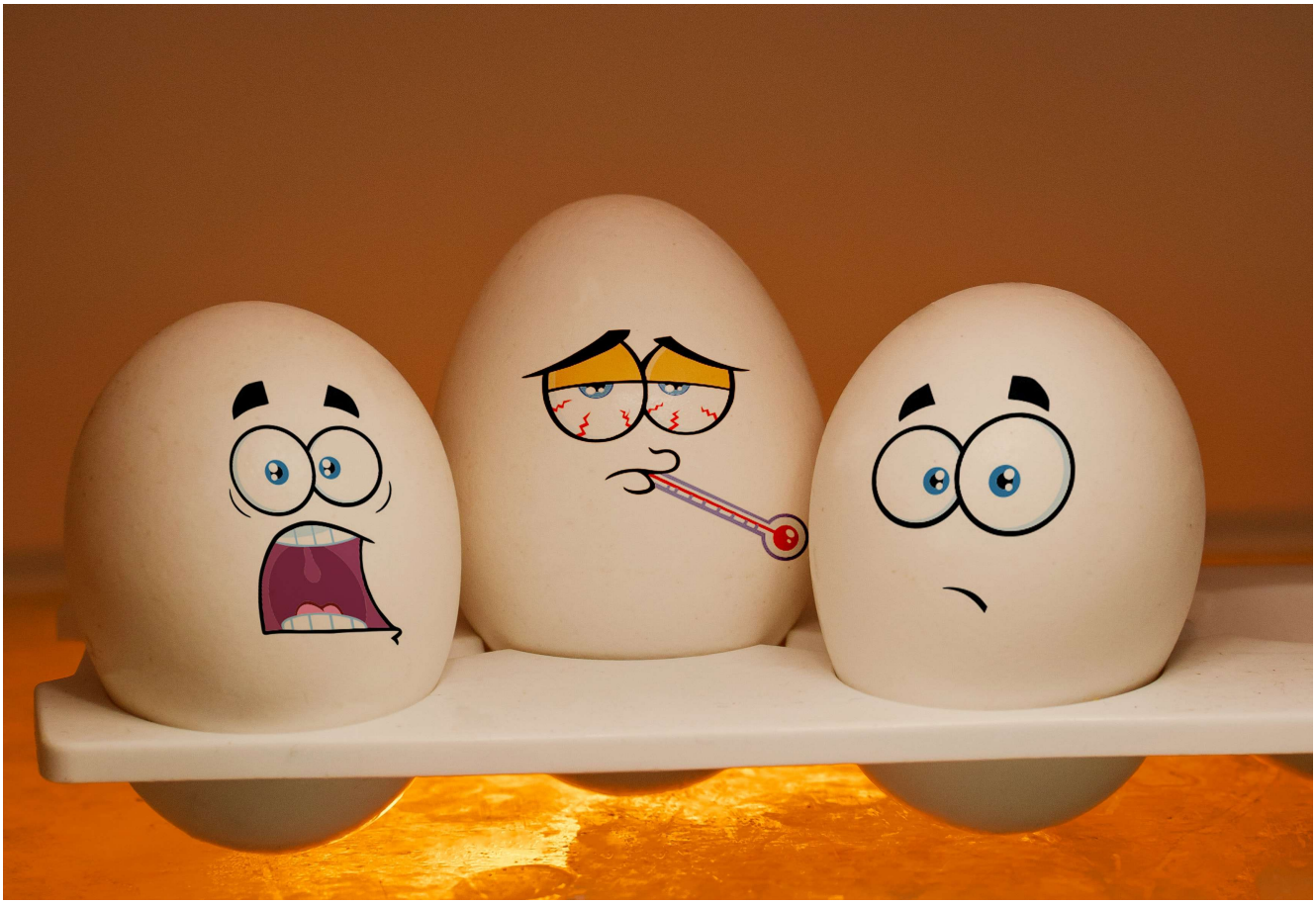
Suvrat Arora — Published On July 7, 2022 and Last Modified On July 12th, 2022

Beginner Machine Learning NLP Python

This article was published as a part of the Data Science Blogathon.

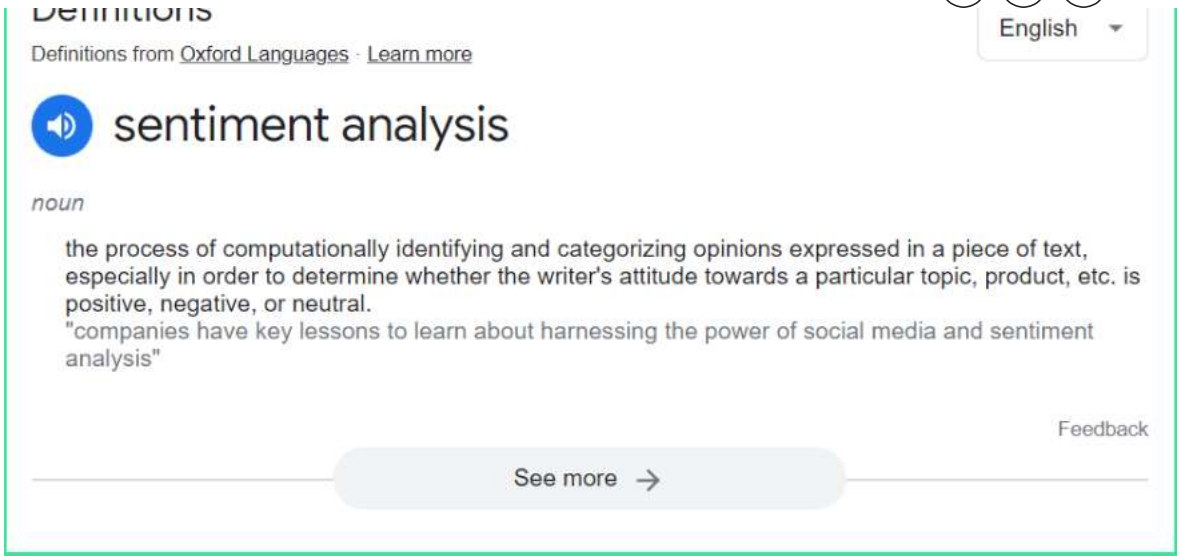
Introduction

Whether you speak of Twitter, Goodreads or Amazon—hardly is there a digital space not saturated with peoples' opinions. In today's world, it is crucial for organizations to dig into these opinions and get insights about their products or services. However, this data exists in such astounding amounts that gauging it manually is a next to impossible pursuit. This is where Data Science's yet another boon comes to play—**Sentiment Analysis**. In this article, we'll explore what sentiment analysis encompasses and the various ways to implement it in Python.



What is Sentiment Analysis?

Sentiment Analysis is a use case of **Natural Language Processing (NLP)** and comes under the category of **text classification**. To put it simply, Sentiment Analysis involves classifying a text into various sentiments, such as positive or negative, Happy, Sad or Neutral, etc. Thus, the ultimate goal of sentiment analysis is to decipher the underlying mood, emotion, or sentiment of a text. This is also known as **Opinion Mining**.



Gaining Insights and Making Decisions with Sentiment Analysis

Well, by now I guess we are somewhat accustomed to what sentiment analysis is. But what is its significance and how do organizations benefit from it? Let us try and explore the same with an example. Assume you start a company that sells perfumes on an online platform. You put up a wide range of fragrances out there and soon customers start flooding in. After some time you decide to change the pricing strategy of perfumes—you plan to increase the prices of the popular fragrances and at the same time offer discounts on unpopular ones. Now, in order to determine which fragrances are popular, you start going through customer reviews of all the fragrances. But you're stuck! They are just so many that you cannot go through them all in one lifetime. This is where sentiment analysis can rope you out of the pit.

You simply gather all the reviews in one place and apply sentiment analysis to it. The following is a schematic representation of sentiment analysis on the reviews of three fragrances of perfumes—Lavender, Rose, and Lemon. (Please note that these reviews might have incorrect spellings, grammar, and punctuations as it is in the real-world scenarios)

From these results, we can clearly see that:

pricing.

Fragrance-3 (Lemon) has an overall **negative** sentiment associated with it—thus, your company should **consider offering a discount** on it to balance the scales.

This was just a simple example of how sentiment analysis can help you gain insights into your products/services and help your organization make decisions.

Sentiment Analysis Use Cases

We just saw how sentiment analysis can empower organizations with insights that can help them make data-driven decisions. Now, let's peep into some more use cases of sentiment analysis.

1. **Social Media Monitoring for Brand Management:** Brands can use sentiment analysis to gauge their Brand's public outlook. For example, a company can gather all Tweets with the company's mention or tag and perform sentiment analysis to learn the company's public outlook.
2. **Product/Service Analysis:** Brands/Organizations can perform sentiment analysis on customer reviews to see how well a product or service is doing in the market and make future decisions accordingly.
3. **Stock Price Prediction:** Predicting whether the stocks of a company will go up or down is crucial for investors. One can determine the same by performing sentiment analysis on News Headlines of articles containing the company's name. If the news headlines pertaining to a particular organization happen to have a positive sentiment—its stock prices should go up and vice-versa.

Ways to Perform Sentiment Analysis in Python

Python is one of the most powerful tools when it comes to performing data science tasks—it offers a multitude of ways to perform sentiment analysis. The most popular ones are enlisted here:

1. Using Text Blob
2. Using Vader
3. Using Bag of Words Vectorization-based Models
4. Using LSTM-based Models
5. Using Transformer-based Models

Let's dive deep into them one by one.

> **Note:** For the purpose of demonstrations of methods 3 & 4 (Using Bag of Words Vectorization-based Models and Using LSTM-based Models) sentiment analysis has been used. It comprises more than 5000 text excrepts labelled as positive, negative or neutal. The dataset lies under the Creative Commons licence.

Using Text Blob

Text Blob is a Python library for Natural Language Processing. Using Text Blob for sentiment analysis is quite simple. It takes text as an input and can return **polarity** and **subjectivity** as outputs.

Polarity determines the sentiment of the text. Its values lie in [-1,1] where -1 denotes a highly negative sentiment and 1 denotes a highly positive sentiment.

Subjectivity determines whether a text input is factual information or a personal opinion. Its value lies between [0,1] where a value closer to 0 denotes a piece of factual information and a value closer to 1 denotes a personal opinion.

Installation:

```
from textblob import TextBlob
```

Code Implementation for Sentiment Analysis Using Text Blob:

Writing code for sentiment analysis using TextBlob is fairly simple. Just import the TextBlob object and pass the text to be analyzed with appropriate attributes as follows:

Python Code:

Sentiment Analysis Using Python

Show files 0 Run 1

Subjectivity of Text 2 is 1.0

social media text. Just like Text Blob, its usage in Python is pretty simple. We'll see its usage in code implementation with an example in a while.

Installation:

```
pip install vaderSentiment
```

Importing SentimentIntensityAnalyzer class from Vader:

```
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
```

Code for Sentiment Analysis Using Vader:

Firstly, we need to create an object of the SentimentIntensityAnalyzer class; then we need to pass the text to the polarity_scores() function of the object as follows:

```
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
sentiment = SentimentIntensityAnalyzer()
text_1 = "The book was a perfect balance between wrtiting style and plot."
text_2 =  "The pizza tastes terrible."
sent_1 = sentiment.polarity_scores(text_1)
sent_2 = sentiment.polarity_scores(text_2)
print("Sentiment of text 1:", sent_1)
print("Sentiment of text 2:", sent_2)
```

Output:

```
Sentiment of text 1: {'neg': 0.0, 'neu': 0.73, 'pos': 0.27, 'compound': 0.5719}
Sentiment of text 2: {'neg': 0.508, 'neu': 0.492, 'pos': 0.0, 'compound': -0.4767}
```

As we can see, a VaderSentiment object returns a dictionary of sentiment scores for the text to be analyzed.

Using Bag of Words Vectorization-Based Models

In the two approaches discussed as yet i.e. Text Blob and Vader, we have simply used Python libraries to perform sentiment analysis. Now we'll discuss an approach wherein we'll train our own model for the task. The steps involved in performing sentiment analysis using the Bag of Words Vectorization method are as follows:

1. Pre-Process the text of training data (Text pre-processing involves Normalization, Tokenization, Stopwords Removal, and Stemming/Lemmatization.)
2. Create a Bag of Words for the pre-processed text data using the Count Vectorization or TF-IDF Vectorization approach.
3. Train a suitable classification model on the processed data for sentiment classification.

Code for Sentiment Analysis using Bag of Words Vectorization Approach:

To build a sentiment analysis model using the BOW Vectorization Approach we need a labeled dataset. As stated earlier, the dataset used for this demonstration has been obtained from Kaggle. We have simply used sklearn's count vectorizer to create the BOW. After, we trained a Multinomial Naive Bayes classifier, for which an accuracy score of 0.84 was obtained.

Dataset can be obtained from here.

```
import pandas as pd
data = pd.read_csv('Finance_data.csv')
#Pre-Prcoessing and Bag of Word Vectorization using Count Vectorizer
from sklearn.feature_extraction.text import CountVectorizer
from nltk.tokenize import RegexpTokenizer
token = RegexpTokenizer(r'[a-zA-Z0-9]+')
cv = CountVectorizer(stop_words='english',ngram_range = (1,1),tokenizer = token.tokenize)
text_counts = cv.fit_transform(data['sentences'])
#Splitting the data into trainig and testing
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(text_counts, data['feedback'], test_size=0.25,
random_state=5)
#Training the model
from sklearn.naive_bayes import MultinomialNB
MNB = MultinomialNB()
MNB.fit(X_train, Y_train)
#Caluclating the accuracy score of the model
from sklearn import metrics
predicted = MNB.predict(X_test)
accuracy_score = metrics.accuracy_score(predicted, Y_test)
print("Accuracuy Score: ",accuracy_score)
```

Output:

```
Accuracuy Score:  0.9111675126903553
```

The trained classifier can be used to predict the sentiment of any given text input.

Using LSTM-Based Models

Though we were able to obtain a decent accuracy score with the Bag of Words Vectorization method, it might fail to yield the same results when dealing with larger datasets. This gives rise to the need to employ deep learning-based models for the training of the sentiment analysis model.

For NLP tasks we generally use RNN-based models since they are designed to deal with sequential data. Here, we'll train an LSTM (Long Short Term Memory) model using **TensorFlow** with **Keras**. The steps to perform sentiment analysis using LSTM-based models are as follows:

1. Pre-Process the text of training data (Text pre-processing involves Normalization, Tokenization, Stopwords Removal, and Stemming/Lemmatization.)
2. Import **Tokenizer** from Keras.preprocessing.text and create its object. Fit the tokenizer on the entire training text (so that the Tokenizer gets trained on the training data vocabulary). Generated text embeddings using the texts_to_sequence() method of the Tokenizer and store them after padding them to an equal length. (Embeddings are numerical/vectorized representations of text. Since we cannot feed our model with the text data directly, we first need to convert them to embeddings)
3. After having generated the embeddings we are ready to build the model. We build the model using TensorFlow—add Input, LSTM, and dense layers to it. Add dropouts and tune the hyperparameters to get a decent accuracy score. Generally, we tend to use **ReLU** or **LeakyReLU** activation functions in the inner layers of LSTM models as it avoids the vanishing gradient problem. At the output layer, we use Softmax or Sigmoid activation function.

Code for Sentiment Analysis using LSTM-based model approach:

Here, we have used the same dataset as we used in the case of the BOW approach. A training accuracy of 0.90 was obtained.

```
import nltk
import pandas as pd
from textblob import Word
from nltk.corpus import stopwords
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
from keras.models import Sequential
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.layers import Dense, Embedding, LSTM, SpatialDropout1D
from sklearn.model_selection import train_test_split
#Loading the dataset
data = pd.read_csv('Finance_data.csv')
#Pre-Processing the text
def cleaning(df, stop_words):
    df['sentences'] = df['sentences'].apply(lambda x: ' '.join(x.lower() for x in x.split()))
    # Replacing the digits/numbers
    df['sentences'] = df['sentences'].str.replace('d', '')
    # Removing stop words
    df['sentences'] = df['sentences'].apply(lambda x: ' '.join(x for x in x.split() if x not in stop_words))
    # Lemmatization
    df['sentences'] = df['sentences'].apply(lambda x: ' '.join([Word(x).lemmatize() for x in x.split()]))
    return df
stop_words = stopwords.words('english')
data_cleaned = cleaning(data, stop_words)
#Generating Embeddings using tokenizer
tokenizer = Tokenizer(num_words=500, split=' ')
tokenizer.fit_on_texts(data_cleaned['verified_reviews'].values)
X = tokenizer.texts_to_sequences(data_cleaned['verified_reviews'].values)
X = pad_sequences(X)
#Model Building
model = Sequential()
model.add(Embedding(500, 120, input_length = X.shape[1]))
model.add(SpatialDropout1D(0.4))
model.add(LSTM(704, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(352, activation='LeakyReLU'))
model.add(Dense(3, activation='softmax'))
model.compile(loss = 'categorical_crossentropy', optimizer='adam', metrics = ['accuracy'])
print(model.summary())
#Model Training
model.fit(X_train, y_train, epochs = 20, batch_size=32, verbose =1)
#Model Testing
model.evaluate(X_test,y_test)
```

Using Transformer-Based Models

Transformer-based models are one of the most advanced Natural Language Processing Techniques. They follow an Encoder-Decoder-based architecture and employ the concepts of self-attention to yield impressive results. Though one can always build a transformer model from scratch, it is quite tedious a task. Thus, we can use pre-trained transformer models available on Hugging Face. Hugging Face is an open-source AI community that offers a multitude of pre-trained models for NLP applications. These models can be used as such or can be fine-tuned for specific tasks.

Installation:

```
pip install transformers
```

Code for Sentiment Analysis Using Transformer based models:

To perform any task using transformers, we first need to import the pipeline function from transformers. Then, an object of the pipeline function is created and the task to be performed is passed as an argument (i.e sentiment analysis in our case). We can also specify the model that we need to use to perform the task. Here, since we have not mentioned the model to be used, the distillery-base-uncased-finetuned-sst-2-English mode is used by default for sentiment analysis. You can check out the list of available tasks and models here.

```
from transformers import pipeline
sentiment_pipeline = pipeline("sentiment-analysis")
data = ["It was the best of times.", "t was the worst of times."]
sentiment_pipeline(data)
```

Output:

```
[{'label': 'POSITIVE', 'score': 0.999457061290741},  {'label': 'NEGATIVE', 'score': 0.9987301230430603}]
```

Conclusion

In this age when users can express their viewpoints effortlessly and data is generated in superfluity in just fractions of seconds—drawing insights from such data is vital for organizations to make efficient decisions—and Sentiment Analysis proves to be the missing piece of the puzzle!

By now we have covered in great detail what exactly sentiment analysis entails and the various methods one can use to perform it in Python. But these were just some rudimentary demonstrations—you must surely go ahead and fiddle with the models and try them out on your own data.

Before you go…for doubts, queries, or potential opportunities, please feel free to connect with me on LinkedIn or Instagram.

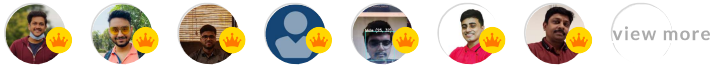
The media shown in this article is not owned by Analytics Vidhya and is used at the Author's discretion.

blogathon sentiment analysis Sentiment Analysis importance

About the Author

Suvrat Arora

Our Top Authors



Previous Post
Why is Face Alignment Important for Face Recognition?

Next Post
Building Smarter Solutions with Machine Learning

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name*

Email*

Website

Notify me of follow-up comments by email.

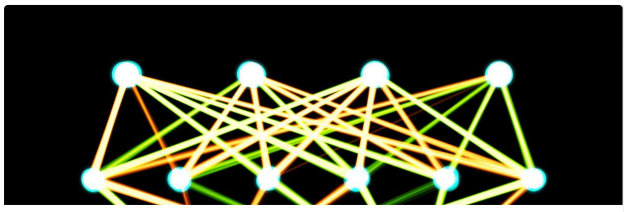
Notify me of new posts by email.

Submit

Top Resources

Python Tutorial: Working with CSV file for Data Science

Harika Bonthu - AUG 21, 2021



Boost Model Accuracy of Imbalanced COVID-19 Mortality Prediction Using GAN-based..

Bala Gangadhar Thilak Adiboina - OCT 07, 2020

The Most Comprehensive Guide to K-Means Clustering You'll Ever Need

Pulkit Sharma - AUG 19, 2019

Creating a Music Streaming Backend Like Spotify Using MongoDB

WoolDoughnut310 - NOV 03, 2022