Can Python understand human feelings through words? – A brief intro to NLP and VADER Sentiment Analysis

Aryan Bajaj — Published On June 17, 2021 and Last Modified On August 23rd, 2022

Advanced Libraries NLP Project Python Structured Data Text Use Cases

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

Introduction

Imagine having the power to observe your customer's thoughts, like what they really think of a particular product/service. For instance, there is a new product launched by **NIKE** and **REEBOK**. Both the companies launched a pair of new sports shoes and posted them on their social media accounts like Instagram or Facebook for marketing purposes.

Is it possible for an individual to check all the thousands or lakhs of comments, obviously some will be good, and some will be bad?

As it is crucial to understand the customer behavior regarding the product/service offered:

> The customer's perception is your reality. – Kate Zabriskie

But is it really possible to understand every comment?

The answer is NO, an individual can't do that, but what if he/she could do it?

Wouldn't it be fascinating to do that?

I am sure till now, you have thought about how you actually could do it. So, let me help you with it.

It can be done using **Natural Language Processing Technique (NLP).**

As we all know, that computer understands Binary language, i.e., the language of 0 and 1. But with the help of NLP, we can help it to understand the **HUMAN LANGUAGE** as it creates dummy variables for all the alphabets, and it has a word cloud net commonly known as Dictionary from where it learns about the different meaning of the words that we use.

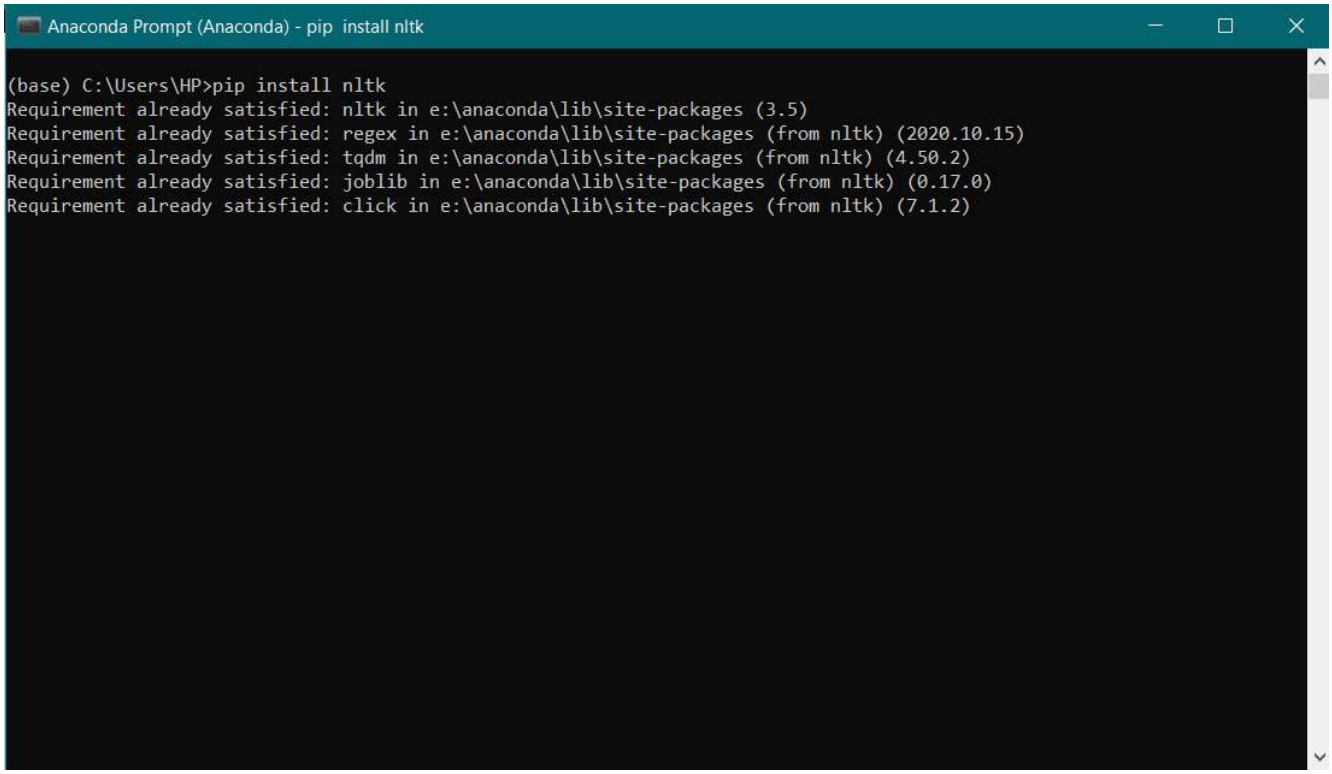
In simpler language, NLP is a process in which we make the computer understand our Human Language, then further we can perform any text analysis.

In this article, we will do Sentimental Analysis after doing Natural language Processing.

Can Python understand human feelings through words? – A brief intro to NLP and VADER Sentiment Analysis

We are doing all the analysis using python 3.

Firstly, we have to install Natural Language Processing Tool Kit Library. For that use, **pip install nltk,** then run the code.



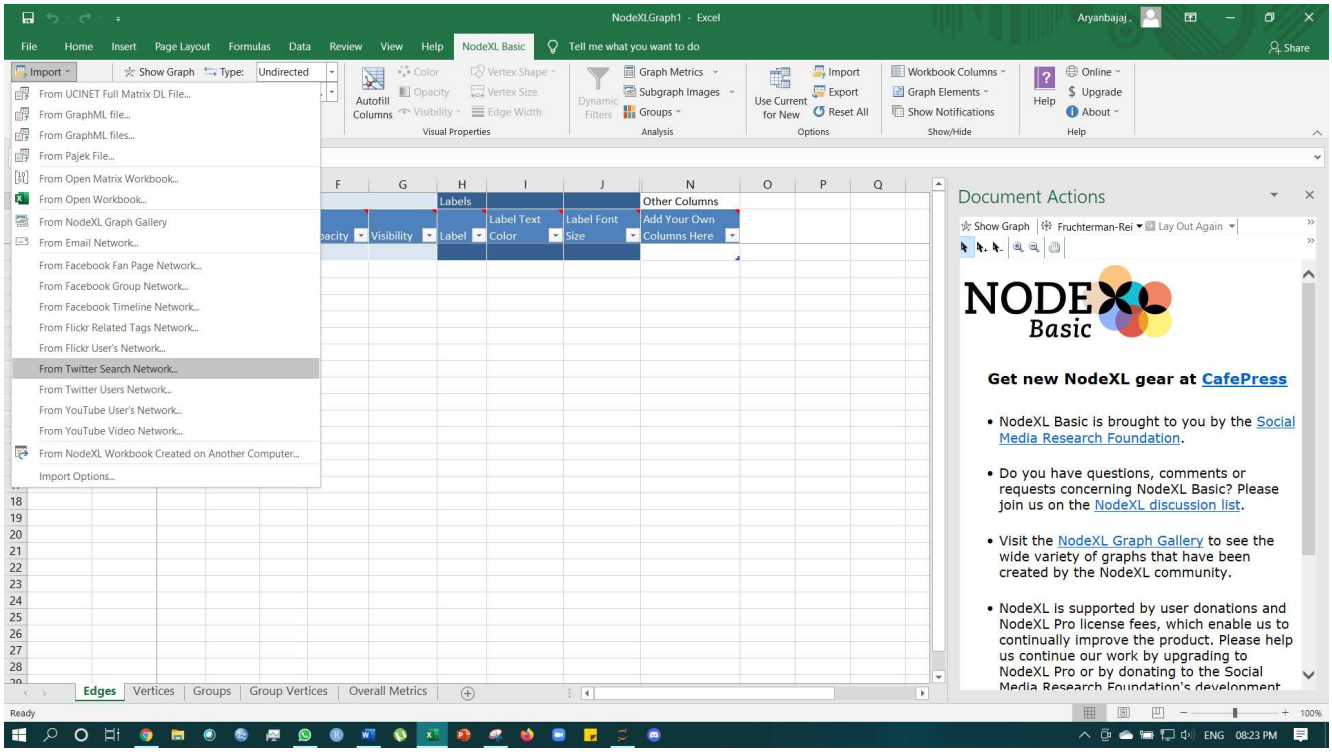
I have already installed it, that's why it is written: **"Requirement already satisfied".**

Now import nltk and nltk.corpus using:

```
import nltk
```

```
import nltk.corpus
```

Can Python understand human feelings through words? – A brief intro to NLP and VADER Sentiment Analysis



Link for downloading NodeXL – *https://www.smrfoundation.org/nodexl/installation/*

One can directly import data from Twitter, Facebook or YouTube, etc. Then one can save the file in .csv format and import the file in python using the pandas library.

```
import pandas as pd
data = pd.read_csv('C:/Users/HP/Desktop/sem 4/Basic Python/Social_Media_Analysis.csv',encoding='latin1')
```

For NLP, I am using my own statement as if I would use the dataset, it will be a very lengthy analysis.

```
text = "In Brazil they drive on the right-hand side of the road. Brazil has a large coastli"
```

Now, from the library nltk.tokenize import word_tokenize as:

Input:

Python Code:

Can Python understand human feelings through words? – A brief intro to NLP and VADER Sentiment Analysis

Show files 0 Run 12

In this, we are tokenizing the words where tokenizing means splitting the sentences or phrases into small words

Input:

```
# finding the frequency distinct in the tokens
# Importing FreqDist library from nltk and passing token into FreqDist
from nltk.probability import FreqDist
fdist = FreqDist(token)
fdist
```

Output:

```
FreqDist({'Brazil': 2, 'the': 2, 'In': 1, 'they': 1, 'drive': 1, 'on': 1, 'right-hand': 1, 'side': 1, 'of': 1,
'road': 1, ...})
```

From the output, it can be observed that 'Brazil' has been used twice, 'the' has been used twice, 'In' has been used a single time, and so on.

After that, to find the frequency of the top 10 words, we'll use fdist.most_common(10).

Input:

```
# To find the frequency of top 10 words
fdist1 = fdist.most_common(10)
fdist1
```

Output:

```
[('Brazil', 2),
 ('the', 2),
 ('In', 1),
 ('they', 1),
 ('drive', 1),
 ('on', 1),
 ('right-hand', 1),
 ('side', 1),
 ('of', 1),
 ('road', 1)]
```

It can be observed that 'Brazil', 'the', 'In', 'they', 'drive', 'on', 'right-hand', 'side', 'of', 'road' are the top 10 words used in the sentence.

After this, we will use the stemming method to break down the word into a simpler form, for example, I am using waiting, given, giving and give:

For stemming, we will import **PorterStemmer** and **LancasterStemmer**, these are the two methods from which we can perform the stemming procedure.

Input:

```
# Importing Porterstemmer from nltk library
# Checking for the word 'waiting'
from nltk.stem import PorterStemmer
pst = PorterStemmer()
pst.stem("waiting")
```

Can Python understand human feelings through words? – A brief intro to NLP and VADER Sentiment Analysis

Input:

```
# Importing LancasterStemmer from nltk
from nltk.stem import LancasterStemmer
lst = LancasterStemmer()
stm = ["giving", "given", "given", "gave"]
for word in stm :
    print(word+ ":" +lst.stem(word))
```

Output:

```
giving:giv
given:giv
given:giv
gave:gav
```

After Stemming, comes **Lemmatization** which helps in breaking down words from plural to a singular form.

Lemmatization can be used in python3 by using Wordnet Lemmatizer, Spacy Lemmatizer, TextBlob, Stanford CoreNLP.

For example, I am using the word 'corpora' and 'rocks':

> **Lemmatization**
>
> For example, lemmatization would correctly identify the base form of 'caring' to 'care', whereas, stemming would cutoff the 'ing' part and convert it to a car. Lemmatization can be implemented in python by using Wordnet Lemmatizer, Spacy Lemmatizer, TextBlob, Stanford CoreNLP

Input:

```
# Importing Lemmatizer library from nltk
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
print("rocks :", lemmatizer.lemmatize("rocks"))
print("corpora :", lemmatizer.lemmatize("corpora"))
```

Output:

```
rocks : rock
corpora : corpus
```

After Lemmatization, we have to identify **Stop Words**. Stop words are nothing but the articles used in the English language like "the", "a", "at", "for", "above", "on", "is", "all".

Articles don't have any meaning while performing sentimental analysis as we cannot conclude anything from "the", "a", "at", "for", "above", "on", "is", "all". The system will look for the words that have some meaning in identifying a product/service like 'Good', 'Bad' or 'Great'.

For example, I will use a sentence:

```
text = "Cristiano Ronaldo was born on February 5, 1985, in Funchal, Madeira, Portugal."
```

Let's check the output:

> "Stop words" are the most common words in a language like "the", "a", "at", "for", "above", "on", "is", "all". These words do not provide any meaning and are usually removed from texts. We can remove these stop words using nltk library

Input:

```
# importing stopwords from nltk library
from nltk import word_tokenize
from nltk.corpus import stopwords
a = set(stopwords.words('english'))
text = "Cristiano Ronaldo was born on February 5, 1985, in Funchal, Madeira, Portugal."
text1 = word_tokenize(text.lower())
print(text1)
stopwords = [x for x in text1 if x not in a]
print(stopwords)
```

Output:

```
['cristiano', 'ronaldo', 'was', 'born', 'on', 'february', '5', ',', '1985', ',', 'in', 'funchal', ',',
'madeira', ',', 'portugal', '.']
['cristiano', 'ronaldo', 'born', 'february', '5', ',', '1985', ',', 'funchal', ',', 'madeira', ',',
'portugal', '.']
```

After this,

We have to make the system understand that what all words are 'nouns', 'pronouns', etc., for better analysis.

It is known as **Part of Speech Tagging (POS)**

For example, I will use a sentence:

```
text = "vote to choose a particular man or a group (party) to represent them in parliament."
```

> **Part of speech tagging (POS)**
>
> Part-of-speech tagging is used to assign parts of speech to each word of a given text (such as nouns, verbs, pronouns, adverbs, conjunction, adjectives, interjection) based on its definition and its context. There are many tools available for POS taggers and some of the widely used taggers are NLTK, Spacy, TextBlob, Standford CoreNLP, etc.

Input:

```
text = "vote to choose a particular man or a group (party) to represent them in parliament"
#Tokenize the text
tex = word_tokenize(text)
for token in tex:
    print(nltk.pos_tag([token]))
```

Output:

```
[('choose', 'NN')]
[('a', 'DT')]
[('particular', 'JJ')]
[('man', 'NN')]
[('or', 'CC')]
[('a', 'DT')]
[('group', 'NN')]
[('(', '(')]
[('party', 'NN')]
[(')', ')')]
[('to', 'TO')]
[('represent', 'NN')]
[('them', 'PRP')]
[('in', 'IN')]
[('parliament', 'NN')]
```

After POS, we have to do the **Named entity Recognition** step

It is the step in which the system identifies which word is a person's name, location, etc.

For this, your system should have Ghostscript installed in the file where you have installed python3.

For this step, from **nltk** import **ne_chunk** as it helps in giving a tree-like structure to the output so that it will be easily understandable for us.

> **Named entity recognition**
>
> It is the process of detecting the named entities such as the person name, the location name, the company name, the quantities and the monetary value

Input:

```
from nltk import ne_chunk# tokenize and POS Tagging before doing chunk
text = "vote to choose a particular man or a group (party) to represent them in parliament"
#importing chunk library from nltk
token = word_tokenize(text)
tags = nltk.pos_tag(token)
chunk = ne_chunk(tags)
chunk
```

Output:

Out[31]:

Link to download Ghostscript – https://www.ghostscript.com/download/gsdnld.html

After this, **Chunking** took place, i.e., grouping the individual piece of information into bigger pieces.

For example, I have used:

```
text = "We saw the yellow dog."
```

> Chunking means picking up individual pieces of information and grouping them into bigger pieces. In the context of NLP and text mining, chunking means a grouping of words or tokens into chunks.

Input:

```
text = "We saw the yellow dog"
token = word_tokenize(text)
tags = nltk.pos_tag(token)
reg = "NP: {<DT>?<JJ>*<NN>}"
a = nltk.RegexpParser(reg)
result = a.parse(tags)
print(result)
```

Output:

```
(S We/PRP saw/VBD (NP the/DT yellow/JJ dog/NN))
```

This is the last step of **Natural Language Processing.** Now we can further use this text (comments dataset) in text Analysis (as I earlier mentioned about **Sentiment Analysis**).

Now, its time for the most awaited moment – SENTIMENTAL ANALYSIS

For Sentiment Analysis, we'll use **VADER Sentiment Analysis,** where VADER means Valence Aware Dictionary and sEntiment Reasoner.

VADER is a lexicon and rule-based feeling analysis instrument that is explicitly sensitive to suppositions communicated in web-based media. VADER utilizes a mix of lexical highlights (e.g., words) that are, for the most part, marked by their semantic direction as one or the other positive or negative. Thus, VADER not only tells about the Polarity score yet, in addition, it tells us concerning how positive or negative a conclusion is.

For this, first, install the VADER package using:

> **VADER Sentiment Analysis**

VADER belongs to a kind of sentiment analysis that depends on lexicons of sentiment-related words. In this methodology, every one of the words in the vocabulary is appraised with respect to whether it is positive or negative, and, how +ve or -ve. Beneath you can see an extract from VADER's vocabulary, where more positive words have higher positive evaluations and more adverse words have lower negative grades.

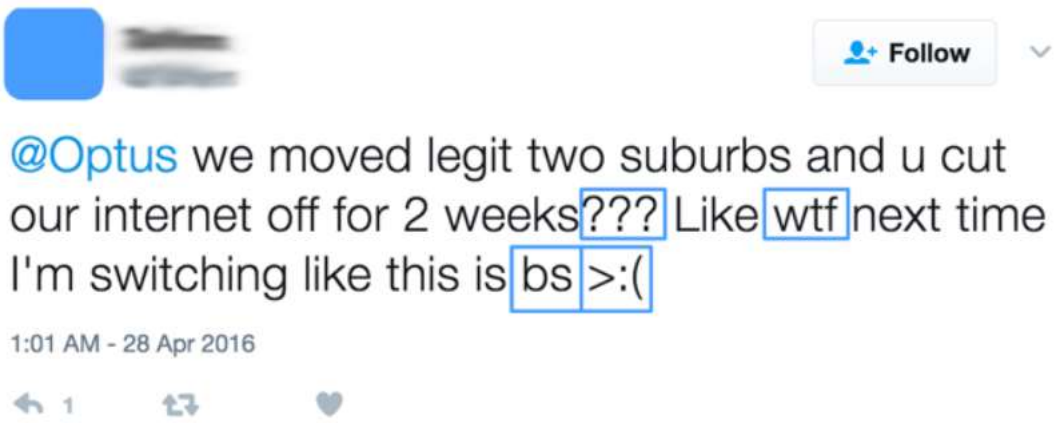
Word	Sentiment rating
tragedy	-3.4
rejoiced	2.0
insane	-1.7
disaster	-3.1
great	3.1

words in your content, else, it will not be extremely accurate. On the other side, when there is a solid match between the lexicon and the content, this methodology is exact and rapidly returns results even on a lot of text. By chance, the engineers of VADER utilized Amazon's Mechanical Turk to get the vast majority of their evaluations, which is an extremely speedy and modest approach to get their grades!

As you would have speculated, when VADER examines a piece of text, it verifies whether any of the words in the content are available in the lexicon. For instance, the sentence "The food is amazing, and the environment is awesome" has two words in the lexicon (amazing and awesome) with the grading of 1.9 and 1.8, respectively (according to the algorithm calculation).
VADER produces four sentiment measurements from these word grading, which you can see underneath. The initial three, +ve, neutral, and -ve, address the extent of the content that falls into those classifications. It can be observed that the model sentence was graded as 45% +ve, 55% neutral, and 0% -ve. The last measurement, the compound score, is the total amount of the lexicon grades (1.9 and 1.8 for this situation), which have been normalized to run between – 1 and 1. For this situation, our model sentence has a rating of 0.69, which is strongly on the +ve side.

What makes VADER for social Media Text?

As you would have speculated, the way that lexicons are costly and tedious to create. It implies that they are not frequently refreshed. This means that they do not have a ton of current slang that might be utilized to communicate – how an individual is feeling. Take the beneath tweet to Optus' client service account, for instance. All the components of this content that show that the individual is miserable (in the blue boxes) are casual compositions – numerous punctuation marks, abbreviations, and emojis. On the off chance that, if you didn't consider this data, this tweet would look neutral to a sentiment analysis calculation!



VADER can handle these types of terms for its lexicon.

Now let's have a look at how VADER MODEL works,

To install VADER Library use:

Input:

```
pip install vaderSentiment
```

Output:

```
satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in e:anacondalibsite-packages (from requests-
>vaderSentiment) (1.25.11) Requirement already satisfied: chardet<4,>=3.0.2 in e:anacondalibsite-packages
(from requests->vaderSentiment) (3.0.4) Requirement already satisfied: idna<3,>=2.5 in e:anacondalibsite-
packages (from requests->vaderSentiment) (2.10) Requirement already satisfied: certifi>=2017.4.17 in
e:anacondalibsite-packages (from requests->vaderSentiment) (2020.6.20) Note: you may need to restart the
kernel to use updated packages.
```

I have already installed it, that's why it is written: **"Requirement already satisfied".**

Now it's time to import SentimentIntensityAnalyzer from vaderSentiment.vaderSentiment

Input:

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

After importing VADER libraries, it's time to create **LOOP Functions** that help identify our text like – what is the percentage of the entered text (Text Dataset) is Negative/Positive/Neutral?

Input:

```
# function to print sentiments
# of the sentence.
def sentiment_scores(sentence):
# Create a SentimentIntensityAnalyzer object.
    sid_obj = SentimentIntensityAnalyzer()
# polarity_scores method of SentimentIntensityAnalyzer
# oject gives a sentiment dictionary.
# which contains pos, neg, neu, and compound scores.
    sentiment_dict = sid_obj.polarity_scores(sentence)
    print("Overall sentiment dictionary is : ", sentiment_dict)
    print("sentence was rated as ", sentiment_dict['neg']*100, "% Negative")
    print("sentence was rated as ", sentiment_dict['neu']*100, "% Neutral")
    print("sentence was rated as ", sentiment_dict['pos']*100, "% Positive")
    print("Sentence Overall Rated As", end = " ")
    # decide sentiment as positive, negative and neutral
    if sentiment_dict['compound'] >= 0.05 :
        print("Positive")
    elif sentiment_dict['compound'] <= - 0.05 :
        print("Negative")
    else :
        print("Neutral")
```

This function will make a sentence **positive** if the value of the text will be **more than or equal to 0.05**. It will make the text **negative** if the value of the text will be **less than or equal to -0.05, else** it will make the sentence **neutral**.

As I mentioned at the beginning of the article, I am using my own sentences/paragraphs as doing the whole social media analysis would be very lengthy.

One can use the recorded sentiments after cleaning them with the NLP and use the variable that has been selected for the entire comments/sentiments.

For VADER analysis, I am going to use:

Text = "ANN is like our brain; millions and billions of cells — called neurons, which processes information in the form of electric

Can Python understand human feelings through words? – A brief intro to NLP and VADER Sentiment Analysis

This paragraph is taken from one of my previous written Blogs: https://aryanbajaj13.medium.com/.

Now the Final step adding the text into the Driver Code, i.e., it is the code a programmer has set to analyse something.

Input for the Driver Code:

```
# Driver code
if __name__ == "__main__" :
    print("Text Selected for VADER Sentimental Analysis :")
    sentence1 = ('''ANN is like our brain; millions and billions of cells – called neurons, which processes
information in the form of electric signals. Similarly, in ANN, the network structure has an input layer, a
hidden layer, and the output layer. It is also called Multi-Layer Perceptron as it has multiple layers. The
hidden layer is known as a "distillation layer" that distils some critical patterns from the data/information
and passes it onto the next layer. It then makes the network quicker and more productive by distinguishing the
data from the data sources, leaving out the excess data.''')
    print(sentence1)
```

output for the Driver Code:

```
Text Selected for VADER Sentimental Analysis :
ANN is like our brain; millions and billions of cells – called neurons, which processes information in the
form of electric signals. Similarly, in ANN, the network structure has an input layer, a hidden layer, and the
output layer. It is also called Multi-Layer Perceptron as it has multiple layers. The hidden layer is known as
a "distillation layer" that distils some critical patterns from the data/information and passes it onto the
next layer. It then makes the network quicker and more productive by distinguishing the data from the data
sources, leaving out the excess data.
```

The final command for the result of the Analysis:

In this command, we'll get to know about the paragraph, i.e., whether it is a positive statement, Negative Statement or Neutral Statement.

In this command, we call the function that we have created for the Analysis to understand in what percentage the sentence is Positive, Negative and Neutral.

Input:

```
# function calling
sentiment_scores(sentence1)
```

Output:

```
Overall sentiment dictionary is :  {'neg': 0.023, 'neu': 0.951, 'pos': 0.025, 'compound': 0.0516}
sentence was rated as  2.3 % Negative
sentence was rated as  95.1 % Neutral
sentence was rated as  2.5 % Positive
Sentence Overall Rated As Positive
```

Conclusion:

It can be observed that the Sentence is on the Positive side as it has more positive words than negative. (Negative Words = 2.3% whereas, Positive Words = 2.5%)

If you want me to do **Valence Aware Dictionary for Sentiment Reasoning** of a **particular product/service,** MAIL ME AT aryanbajaj104@gmail.com

Contacts:

If you have any questions or suggestions on what my next article should be about, please leave a comment below or write to me at aryanbajaj104@gmail.com.

If you want to keep updated with my latest articles and projects, follow me on Medium.

Connect with me via:

LinkedIn

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

blogathon vader vader library

About the Author

Aryan Bajaj

Our Top Authors

view more

Can Python understand human feelings through words? – A brief intro to NLP and VADER Sentiment Analysis

Previous Post
Build User Interface With GRADIO For Your Deep Learning Project!

Next Post
Understanding Random Forest

Leave a Reply

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

Comment

Name*

Email*

Website

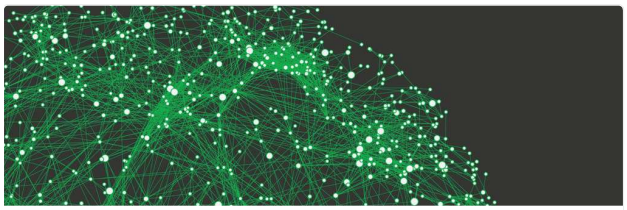
Notify me of follow-up comments by email.

Notify me of new posts by email.

Submit

Top Resources

Python Tutorial: Working with CSV file for Data Science



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

Can Python understand human feelings through words? – A brief intro to NLP and VADER Sentiment Analysis

Understanding Random Forest

Sruthi E R - JUN 17, 2021

Understanding Support Vector Machine(SVM) algorithm from examples (along with code)

Sunil Ray - SEP 13, 2017