Sentiment Analysis Day Of Demonstration

A Workshops For Ukraine Production

Code in this notebook available under the MIT License (see below).

Prepared by Dr. Adam Ross Nelson, Career Coach + Data Scientist.

Previewing content from a chapter in the upcoming

Confident Data Science: Discovering The Essential Skills of Data Science (Nelson, 2023). Available from Kogan Page, Inc, Amazon.com, and other booksellers beginning September 2023. This presentation will be a benefit to support **Workshops For Ukraine**.

You can...

Pre-Order *Confident Data Science* Today (At Amazon.com)

Time: 6pm to 8pm (European Central) | 12noon to 2pm (US Eastern) | 11am to 1pm (US Central / Central American) | 9am to 11am (US Pacific).

Date: Thursday February 23, 2023

The following is a preview of the workshop along with some additional advice on how to prepare yourself ahead of time.

Consider Learning How to Use Google Colab.

Some helpful Google Colab resources:

- How to Use Google Colab
- How To Get Data From Gdrive Into Google Colab

Below Is a Preview of NLTK

This is merely a quick demonstration: Here we show a preview of NLTK. During the workshop, and later in this notebook we will provide additional discussion and explanation of the code.

```
# Standard Imports
import pandas as pd
import numpy as np
import seaborn as sns

# Natural Language Tool Kit Imports
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from nltk.tokenize import sent_tokenize


# For The First Run Only
# nltk.download('vader_lexicon')


# Instantiate SentimentIntensity Analyzer
sid = SentimentIntensityAnalyzer()
# Analyze Example Text (And See Results)
sid.polarity_scores('I am happy to announce a promotion at work!')
```

```
    {'neg': 0.0, 'neu': 0.6, 'pos': 0.4, 'compound': 0.6114}
```

```
# With Different Punctuation Analyze Example Text
sid.polarity_scores('I am happy to announce a promotion at work.')
```

```
    {'neg': 0.0, 'neu': 0.619, 'pos': 0.381, 'compound': 0.5719}
```

We can interpret this output as follows. The neg score indicates 0.0 meaning there is no negative sentiment. The neutral score shows 0.619 and the positive score indicates 0.381. From this output, read as a whole, we understand the input 'I am happy to announce a promotion at work!' as expressing a positive sentiment.

When looking for a summary of the output, we can turn to the compound score. The compound score which will range from -1 to 1 interprets as a summary, of sorts, for the other output. The compound score will be negative when the overall sentiment is negative and positive when the overall sentiment is positive.

Thanks For Reading

Please also remember to...

Pre-Order *Confident Data Science* Today (At Amazon.com)

Time: 6pm to 8pm (European Central) | 12noon to 2pm (US Eastern) | 11am to 1pm (US Central / Central American) | 9am to 11am (US Pacific).

Date: Thursday February 23, 2023

Begin Today's Workshop

The first step for attendees is to get started in a Python environment. For the past several years Jupyter notebooks have been a prefered platform for teaching, learning, and testing. The two quickest ways to get started in a Jupyter notebook are:

- To use Google Colab, or...
- To use an installation of Anaconda.

I will demonstrate today's code in Google Colab.

Questions:

1. In the chat, are you NEW TO, SOMEWHAT FAMILIAR, or VERY FAMILIAR with Google Colab?
2. Also in the chat, are you NEW TO, SOMEWHAT FAMILIAR, or VERY FAMILIAR with Jupyter notebooks?
3. Before I go any further, can I help anyone get going in Google Colab? Any questions about Google Colab.

Whether you're in Google Colab, Jupyter notebooks or any other Python environment, one of the quickest (and most poetic) ways to make sure you are ready to go is to type `import this` and then to run that code. Here is how that will look for you:

```
import this
```

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Standard Imports

Most of the following examples assume you have also executed the following standard imports.

If you are not familiar with so-called standard imports here is some additional information on the topic:

In Python, a standard import is a commonly used import statement that is considered a best practice and is used by convention. Standard imports are widely used in the Python community and are generally understood by most Python programmers.

For example, in this code snippet the import pandas as `pd` statement is a standard import for the Pandas library. It imports the Pandas library and renames it to `pd`. The use of `pd` is somewhat arbitrary, we could have named it anything, except to rename Pandas as `pd` is a widely used convention.

Similarly, the import numpy as `np` statement is a standard import for the Numpy library. The long standing convention is to import and rename Numpy as `np`.

In addition to importing Pandas as pd and Numpy as np, this code also follows the standard convention for importing Seaborn which is to rename it as sns. Subsequent code can now reference these tools by their renamed shorthand.

```
import pandas as pd
import numpy as np
import seaborn as sns
```

NLTK + Lexicon Base Approaches

Lexicon-based approaches take advantage of established dictionaries called "lexicons" that include basic words and their associated sentiment. For example, a lexicon might contain the word "love" and indicate that it is strongly positive.

Note: For the code below we will need a list of words to "look for" in the lexicon. We will collect words from attendees. Otherwise a suggested list is:

```
adjectives = [
    'Adoringly', 'Brave', 'Creative', 'Daring', 'Doomed', 'Energetic',
    'Friendly', 'Generous', 'Grin', 'Honest', 'Intelligent', 'Joyful',
    'Kind', 'Lost', 'Loyal', 'Magnificent', 'Noble', 'Optimistic',
    'Playful', 'Questionable', 'Rebellious', 'Strong', 'Trick',
    'Trustworthy', 'Unkind', 'Verdict', 'Wreck', 'Worry', 'Youthful',
    'Zealot']
```

The following code will show you specific examples.

```
import nltk

# Necessary for the first run only (downloading the dictionary)
# nltk.download('vader_lexicon')

# Import SentimentIntensityAnalyzer
from nltk.sentiment.vader import SentimentIntensityAnalyzer

# Load VADER lexicon
vader_lexicon = SentimentIntensityAnalyzer().lexicon

# Specify a list of adjectives (be sure to replace underscores)
adjectives = [
    '____', '____', '____', '____', '____', '____',
    'Friendly', 'Generous', 'Grin', 'Honest', 'Intelligent', 'Joyful',
    'Kind', 'Lost', 'Love', 'Magnificent', 'Noble', 'Optimistic',
    'Playful', 'Questionable', 'Rebellious', 'Strong', 'Trick',
    'Trustworthy', 'Unkind', 'Verdict', 'Wreck', 'Worry', 'Youthful',
    'Zealot']

# Declare empty lists to be populated in the for loop
word_col = []
sent_col = []

# Use for loop to populate list
for word in adjectives:
  # Use try and except statments to manage words not in lexicon
  try:
    sent_col.append(vader_lexicon[word.lower()])
    word_col.append(word)
  except:
    sent_col.append(np.nan)
    word_col.append(f'{word} Not Found')

# Convert lists to Pandas DataFrame for display
pd.DataFrame({'Word 1':word_col[:10], 'Sentiment 1':sent_col[:10],
              'Word 2':word_col[10:20], 'Sentiment 2':sent_col[10:20],
              'Word 3':word_col[20:], 'Sentiment 3':sent_col[20:]})
```

	Word 1	Sentiment 1	Word 2	Sentiment 2	Word 3	Sentiment 3
0	___ Not Found	NaN	Intelligent	2.0	Rebellious	-1.2
1	___ Not Found	NaN	Joyful	2.9	Strong	2.3
2	___ Not Found	NaN	Kind	2.4	Trick	-0.2
3	___ Not Found	NaN	Lost	-1.3	Trustworthy	2.6

Further explanation of the code above.

The code starts by importing the necessary libraries, including pandas and NLTK. It then downloads the VADER lexicon using the `nltk.download()` function, and loads the VADER sentiment analyzer using the `SentimentIntensityAnalyzer()` function.

The code then specifies a list of adjectives (`adjectives`) to be analyzed, and declares two empty lists to store the words (`word_col`) and their associated sentiment scores (`sent_col`). It uses a for loop to loop through each adjective in the list, and for each adjective, it gets its associated sentiment score from the VADER lexicon using the `vader_lexicon[word.lower()]` syntax, where word is the current adjective in the loop. The sentiment score is then appended to the `sent_col` list, and the word is appended to the `word_col` list.

Finally, the code creates a Pandas DataFrame to display the words and their associated sentiment scores in a table format. The DataFrame has three major columns which shows each word and the associated sentiment. Each column contains 10 adjectives and their associated sentiment scores.

Mini-Workshop On Your Own

Task 1: Write code that will extract the lexicon's sentiment score for the following "words," just one word at a time.

```
abandon
idk
xoxo
mwah
;)
```

Task 2: Write code that will display the entire lexicon.

```
# Put the code for Task 1 here:
```

```
# Put the code for Task 2 here:
```

Working With A Large Collection Of Data

For most use cases, finding the sentiment of a single word, or a single sentence is not sufficient. We need a way to return sentiments for a large collection of entries. This section will demonstrate further.

First, we will collect data from the "sentence-level." After collecting data from the sentence level we will also collapse that data to summarize it at the "observation-level."

Second, we will forgo first collecting sentiment data from the "sentence-level" and grab sentiment first-off from the "observation level."

Before we do this, lets gather some data. Who would care to share the a post of theirs from LinkedIn that we can use as examples? If you have LinkedIn (or Facebook could work too), hop on over to your profile copy a recent post and past it in the chat. I'll collect those responses here.

```
social_posts = pd.Series(
    ['''I was not a fan of The Hobbit. This moving is not a fan of mine.
    do not waste your time. See a different movie this weeken!''',
     '''I am not even sure what I was thinking. That was a bad idea.
     Next time I will have to be more careful.''',
     '''___''',
     '''___''',
     '''___''',
     '''___'''])
```

To revisit and refresh ourselves on the single sentence potential... we can pass a single entry from this Pandas Series into the sentiment analyzer.

```
# Instantiate SentimentIntensity Analyzer
sid = SentimentIntensityAnalyzer()

# Analyze Example Text (And See Results)
sid.polarity_scores(social_posts[0])

    {'neg': 0.159, 'neu': 0.753, 'pos': 0.088, 'compound': -0.2225}
```

Sentence-Level Analysis

To work more efficiently, lets define a function that will iteratively move through the entire Pandas series and return a DataFrame that summarizes the sentiments for each of the social posts.

Notice that we also add two additional imports here.

Related, notice we also have a necessary download (`nltk.download('punkt')`).

```
# For The First Run Only
# nltk.download('punkt')
```

```
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from nltk.tokenize import sent_tokenize

# Name the function sentence_scores. Pass a Pandas Series to the function.
def sentence_scores(series):
    '''
    Takes a Pandas Series that contains text data.
    Returns a Pandas DataFrame that reports sentiment scores
    for each sentence in the original Series.'''

    # Define empty lists that will contain each sentence’s results.
    index = []
    sentences = []
    neg_scores = []
    neu_scores = []
    pos_scores = []
    compounds = []

    # Instantiate the analyzer.
    sid = SentimentIntensityAnalyzer()

    # Loop through each item in the Series.
    for i in series.index:
        # Look through each sentence in the Series item.
        for s in sent_tokenize(series.loc[i]):

            # Extract scores for each sentence; append to lists.
            try:
                scores = sid.polarity_scores(s)
                index.append(i)
                sentences.append(s)
                neg_scores.append(scores['neg'])
                neu_scores.append(scores['neu'])
                pos_scores.append(scores['pos'])
                compounds.append(scores['compound'])
            except:
                pass

    # Compile & return Pandas DataFrame
    return(pd.DataFrame({'index':index,
                         'Sentence':sentences,
                         'Negative':neg_scores,
                         'Neutral':neu_scores,
                         'Positive':pos_scores,
                         'Compound':compounds}).set_index('index'))
```

```
df = sentence_scores(social_posts)
df
```

	Sentence	Negative	Neutral	Positive	Compound
index					
0	I was not a fan of The Hobbit.	0.282	0.718	0.000	-0.2411
0	This moving is not a fan of mine.	0.246	0.754	0.000	-0.2411
0	do not waste your time.	0.000	0.632	0.368	0.3252
0	See a different movie this weeken!	0.000	1.000	0.000	0.0000
1	I am not even sure what I was thinking.	0.246	0.754	0.000	-0.2411
1	That was a bad idea.	0.538	0.462	0.000	-0.5423
1	Next time I will have to be more careful.	0.000	0.787	0.213	0.2247
2	___	0.000	1.000	0.000	0.0000
3	___	0.000	1.000	0.000	0.0000
4	___	0.000	1.000	0.000	0.0000
5	___	0.000	1.000	0.000	0.0000

In the above code the `index` column corresponds to each document. In the default example that came with this notebook the first document is at index 0 and there are four rows (one row for each sentence). Thus, the above output shows sentiment scores at the "sentence-level."

Aggregating Sentence-Level Results

To see a "document-level" summary we can use the `pd.groupby()` method along with the `pd.concat()` function as follows.

```
aggregated = pd.concat([pd.DataFrame(social_posts),
                         df.groupby(by='index').mean()],
                        axis = 1)
aggregated
```

	0	Negative	Neutral	Positive	Compound
0	I was not a fan of The Hobbit. This moving is ...	0.132000	0.776000	0.092	-0.039250
1	I am not even sure what I was thinking. That w...	0.261333	0.667667	0.071	-0.186233
2	___	0.000000	1.000000	0.000	0.000000
3	___	0.000000	1.000000	0.000	0.000000
4	___	0.000000	1.000000	0.000	0.000000
5	___	0.000000	1.000000	0.000	0.000000

```
# A good way to further understand this output is to check correlations.
aggregated.corr()
```

	Negative	Neutral	Positive	Compound
Negative	1.000000	-0.988961	0.858047	-0.957987
Neutral	-0.988961	1.000000	-0.924674	0.904912
Positive	0.858047	-0.924674	1.000000	-0.674699
Compound	-0.957987	0.904912	-0.674699	1.000000

In the above output we see high correlations. This is to be expected. As a passage's positive sentiment increases we would expect its negative sentiment to correspondingly decrease.

Observation-Level Analysis

So, in the above examples we looked at first generating results at the sentence-level. Then we aggregated the results (rolled up / collapsed) the results to the observation level. **Important question:** What are the strengths and weaknesses of that approach? Is there potentially another (better way)?

As implied above: Another approach is to compile scores for each post at the observation level, instead of for each sentence within each post. The following function uses one less nested loop which returns a set of scores for each item in the Pandas Series instead of a set of scores for each sentence within each item in the Pandas Series.

```
# Name the function post_scores. Pass a Pandas Series to the function.
def post_scores(series):
    '''
    Takes a Pandas Series that contains text data. Returns
    a Pandas DataFrame that reports NLTK's sentiment scores
    for each item in the original Series.'''


    # Define empty lists that will contain each entry’s result.
    index = []
    posts = []
    neg_scores = []
    neu_scores = []
    pos_scores = []
    compounds = []

    # Instantiate the analyzer.
    sid = SentimentIntensityAnalyzer()

    # Loop through each item in the Series.
    for i in series.index:

        # Extract scores for each sentence; append to lists.
        try:
            scores = sid.polarity_scores(series.loc[i])
            index.append(i)
            posts.append(series.loc[i])
            neg_scores.append(scores['neg'])
            neu_scores.append(scores['neu'])
            pos_scores.append(scores['pos'])
            compounds.append(scores['compound'])
        except:
            pass

    # Compile & return Pandas DataFrame
    return(pd.DataFrame({'index':index,
                         'Sentence':posts,
                         'Negative':neg_scores,
                         'Neutral':neu_scores,
                         'Positive':pos_scores,
                         'Compound':compounds}).set_index('index'))
```

```
observation_lev = post_scores(social_posts)
observation_lev
```

index	Sentence	Negative	Neutral	Positive	Compound
0	I was not a fan of The Hobbit. This moving is ...	0.159	0.753	0.088	-0.2225
1	I am not even sure what I was thinking. That w...	0.234	0.685	0.081	-0.5528
2	___	0.000	1.000	0.000	0.0000
3	___	0.000	1.000	0.000	0.0000
4	___	0.000	1.000	0.000	0.0000
5	___	0.000	1.000	0.000	0.0000

Comparing The Results

Above we looked at two approaches.

First, we will collected data from the "sentence-level." After collecting data from the sentence level we also collapsed (rolled up / aggregared) that data to summarize it at the "observation-level."

Second, we skipped first collecting sentiment data from the "sentence-level" and grabed sentiment first-off from the "observation level."

Here we can review the results from the first approach and compare them against the results from the second approach. Correlation analysis

```
aggregated
```

	0	Negative	Neutral	Positive	Compound
0	I was not a fan of The Hobbit. This moving is ...	0.132000	0.776000	0.092	-0.039250
1	I am not even sure what I was thinking. That w...	0.261333	0.667667	0.071	-0.186233
2	___	0.000000	1.000000	0.000	0.000000
3	___	0.000000	1.000000	0.000	0.000000
4	___	0.000000	1.000000	0.000	0.000000
5	___	0.000000	1.000000	0.000	0.000000

```
observation_lev['Compound'].corr(aggregated['Compound'])
```

```
    0.9814669543970205
```

Thanks For Attending

Please remember to...

Pre-Order *Confident Data Science* Today (At Amazon.com)