Data Analytics - Module III: Textual data analysis with Python

Objectives

- Understand textual data structure.
- Master the TextBlob library to import, explore, and manipulate textual datasets
- Perform sentiment analysis on textual data
- Perform tokenization, parsing, frequency detection and spelling correction
- Build word cloud in Python to visualize textual data

Introduction

Large language models (LLMs) are based on textual data. A high-quality textual dataset is the heart of large language model training as it directly impacts the model's performance. It can improve accuracy as clean, accurate, and relevant data enables the LLM to learn accurate patterns and generate more accurate outputs. It can also reduce bias as diverse and representative data helps mitigate biases that may be present in the training data, leading to more fair and unbiased models. Moreover, it can enhance generalization as a wide range of data allows the LLM to generalize better to new and unseen data, improving its adaptability. It can also mean faster training as high-quality data can accelerate the training process by reducing the need for data cleaning and preprocessing. The data should also be cleaned to remove irrelevant or noisy information. Finally, the data should be normalized to ensure consistency in formatting and style.

There are different Python libraries for analyzing textual data including NLTK, TextBlob, and spaCy. In this part of the lesson, we will learn about TextBlob which is a Python library for processing textual data. It provides a simple API for diving into common natural language processing (NLP) tasks such as

part-of-speech tagging, noun phrase extraction, sentiment analysis, and classification. With TextBlob, you can perform several other tasks like part-of-speech tagging, tokenization (splitting text into words and sentences), word and phrase frequencies, parsing, n-grams, word inflection (pluralization and singularization) and lemmatization, and spelling correction. We cover some of the basic features of this powerful Python library in this lesson.

Note

Typically, we would use "pip install" to install to a virtual environment and "conda install" to install to a conda environment. To install TextBlob using pip, you can open your command prompt or terminal and use the following commands

- **pip install -U textblob**
- **python -m textblob.download_corpora**

TextBlob is also available as a conda package. To install with conda, run:

- **conda install -c conda-forge textblob**
- **python -m textblob.download_corpora**

NOTE: Sometimes you also need to include the following code in your JupyterLab to install required components of the TextBlob and NLTK library:

nltk.download('punkt')
nltk.download('wordnet')
nltk.download('omw-1.4')
nltk.download('punkt_tab')
nltk.download('averaged_perceptron_tagger_eng')

1. Textual data analysis with TextBlob

1.1. Create a TextBlob

TextBlob aims to provide access to common text-processing operations through a familiar interface. You can treat TextBlob objects as if they were Python strings that learned how to do Natural Language Processing. First, let's import TextBlob library.

Code:

```
from textblob import TextBlob
```

Let's create our first TextBlob. We need to feed TextBlob something like plain text.

Code:

```
wiki = TextBlob("Python is a high-level, general-purpose
programming language.")
```

1.2. Part-of-speech Tagging

Part-of-speech tags can be accessed through the tags property.

Code:

```
wiki.tags
```

Output:

```
[('Python', 'NNP'), ('is', 'VBZ'), ('a', 'DT'),
('high-level', 'JJ'), ('general-purpose', 'JJ'),
('programming', 'NN'), ('language', 'NN')]
```

The output is a list of tuples, where each tuple represents a word and its corresponding part-of-speech (POS) tag. These tags are assigned using a specific tagging scheme. These are tags in this example:

- NNP stands for "Proper Noun, Singular". This indicates that "Python" is a specific, named entity.
- VBZ stands for "Verb, Present Tense, 3rd Person Singular". This means "is" is a verb in the present tense, used with a third-person singular subject.
- DT stands for "Determiner". This indicates that "a" is an article, used to specify a noun.
- JJ stands for "Adjective". This means "high-level" is a descriptive word modifying a noun.
- NN stands for "Noun, Singular". This means "programming" is a singular noun.

1.3. Sentiment Analysis

One type of sentiment analysis is to calculate the overall positivity or negativity of a text corpus. There are a variety of algorithms and scales. TextBlob's default .sentiment function rates an input text as negative or positive on a scale of -1 to 1. The sentiment function returns two numbers in the form of a named tuple:

Sentiment(polarity, subjectivity).

The polarity score is a float within the range [-1.0, 1.0]. Subjectivity is a float within the range [0.0, 1.0] where 0.0 is very objective (ie: does not express strong sentiment) and 1.0 is very subjective.

Code:

```
testimonial = TextBlob("Textblob is amazingly simple to
use. What great fun!")

print(testimonial.sentiment)
print(testimonial.sentiment.polarity)
```

Output:

```
Sentiment(polarity=0.39166666666666666,
subjectivity=0.4357142857142857)

0.39166666666666666
```

The textblob.sentiments module contains two sentiment analysis implementations, PatternAnalyzer (based on the pattern library) and NaiveBayesAnalyzer (an NLTK classifier trained on a movie reviews corpus).

The default implementation is PatternAnalyzer, but you can override the analyzer by passing another implementation into a TextBlob's constructor.

For instance, the NaiveBayesAnalyzer returns its result as a namedtuple of the form: Sentiment(classification, p_pos, p_neg).

Code:

```
from textblob import TextBlob

from textblob.sentiments import NaiveBayesAnalyzer

blob = TextBlob("I love this library",
analyzer=NaiveBayesAnalyzer())

print(blob.sentiment)
```

Output:

```
Sentiment(classification='pos', p_pos=0.7996209910191279,
p_neg=0.2003790089808724)
```

1.4. Tokenization and Lemmatization

You can break TextBlobs into words or sentences.

Code:

```
zen = TextBlob(

    "Beautiful is better than ugly. "

    "Explicit is better than implicit. "

    "Simple is better than complex."

)

print(zen.words)


print(zen.sentences)
```

Output:

```
WordList(['Beautiful', 'is', 'better', 'than', 'ugly',
'Explicit', 'is', 'better', 'than', 'implicit', 'Simple',
'is', 'better', 'than', 'complex'])

[Sentence("Beautiful is better than ugly."),
Sentence("Explicit is better than implicit."),
Sentence("Simple is better than complex.")]
```

Sentence objects have the same properties and methods as TextBlobs.

Code:

```
for sentence in zen.sentences:

    print(sentence.sentiment)
```

Output:

```
Sentiment(polarity=0.2166666666666667,
subjectivity=0.8333333333333334)
Sentiment(polarity=0.5, subjectivity=0.5)
Sentiment(polarity=0.06666666666666667,
subjectivity=0.41904761904761906)
```

Here, polarity represents the sentiment of the text, ranging from -1 to 1:

- -1: Very negative sentiment
- 0: Neutral sentiment
- 1: Very positive sentiment

The closer the polarity is to 1, the more positive the sentiment is. The closer the polarity is to -1, the more negative the sentiment is. If it's close to 0, it suggests neutral sentiment. Subjectivity measures how subjective or objective the text is, with a range from 0 to 1:

0: Very objective (factual)
1: Very subjective (opinion-based)

Higher subjectivity values indicate that the text is more opinion-based or emotional, while lower values indicate that the text is more factual or neutral.

The words and sentences properties are helpers that use the textblob.tokenizers.WordTokenizer and textblob.tokenizers.SentenceTokenizer classes, respectively.

You can use other tokenizers, such as those provided by NLTK, by passing them into the TextBlob constructor then accessing the tokens property.

Code:

```
from textblob import TextBlob
from nltk.tokenize import TabTokenizer
tokenizer = TabTokenizer()
blob = TextBlob("This is\ta rather tabby\tblob.",
tokenizer=tokenizer)
blob.tokens
```

Output:

```
WordList(['This is', 'a rather tabby', 'blob.'])
```

You can also use the tokenize([tokenizer]) method.

Code:

```
from textblob import TextBlob
from nltk.tokenize import BlanklineTokenizer
tokenizer = BlanklineTokenizer()
blob = TextBlob("A token\n\nof appreciation")
blob.tokenize(tokenizer)
```

Output:

```
WordList(['A token', 'of appreciation'])
```

Lemmatization is the process of grouping together the different inflected forms of a word so they can be analyzed as a single item. Words can be lemmatized by calling the lemmatize method.

Code:

```
from textblob import Word
w = Word("octopi")
print(w.lemmatize())
w = Word("went")
print(w.lemmatize("v"))
 # Pass in WordNet part of speech (verb)
```

Output:

```
'octopus'
'go'
```

1.5. Spelling Correction

Use the correct() method to attempt spelling correction.

Code:

```
b = TextBlob("I havv goood speling!")

print(b.correct())
```

Output:

```
I have good spelling!
```

Word objects have a spellcheck() Word.spellcheck() method that returns a list of (word, confidence) tuples with spelling suggestions.

Code:

```
from textblob import Word

w = Word("falibility")

w.spellcheck()
```

Output:

```
[('fallibility', 1.0)]
```

1.6. Get Word and Noun Phrase Frequencies

There are two ways to get the frequency of a word or noun phrase in a TextBlob.

The first is through the word_counts dictionary.

Code:

```
monty = TextBlob("We are no longer the Knights who say Ni.
"

                 "We are now the Knights who say Ekki
ekki ekki PTANG.")

monty.word_counts['ekki']
```

Output:

```
3
```

If you access the frequencies this way, the search will not be case sensitive, and words that are not found will have a frequency of 0. The second way is to use the count() method. You can specify whether or not the search should be case-sensitive (default is False). Each of these methods can also be used with noun phrases.

Code:

```
print(monty.words.count('ekki'))
print(monty.words.count('ekki', case_sensitive=True))
print(wiki.noun_phrases.count('python'))
```

Output:

```
3
2
1
```

1.7. Parsing

Use the parse() method to parse the text.

Code:

```
b = TextBlob("And now for something completely
different.")

print(b.parse())
```

Output:

```
And/CC/O/O now/RB/B-ADVP/O for/IN/B-PP/B-PNP
something/NN/B-NP/I-PNP completely/RB/B-ADJP/O
different/JJ/I-ADJP/O ././O/O
```

Parser implementations can also be passed to the TextBlob constructor.

Code:

```
from textblob import TextBlob
from textblob.parsers import PatternParser
blob = TextBlob("Parsing is fun.", parser=PatternParser())
blob.parse()
```

Output:

```
'Parsing/VBG/B-VP/O is/VBZ/I-VP/O fun/VBG/I-VP/O ././O/O'
```

2. TextBlobs and Python strings

TextBlobs Are Like Python Strings! You can use Python's substring syntax.

Code:

```
zen[0:19]
```

Output:

```
TextBlob("Beautiful is better")
```

You can use common string methods, and also can make comparisons between TextBlobs and strings.

Code:

```
print(zen.upper())
zen.find("Simple")
```

Output:

```
TextBlob("BEAUTIFUL IS BETTER THAN UGLY. EXPLICIT IS
BETTER THAN IMPLICIT. SIMPLE IS BETTER THAN COMPLEX.")
65
```

Code:

```
apple_blob = TextBlob("apples")
banana_blob = TextBlob("bananas")

print(apple_blob < banana_blob)

print(apple_blob == "apples")
```

Output:

```
True
True
```

You can concatenate and interpolate TextBlobs and strings.

Code:

```
print(apple_blob + " and " + banana_blob)
"{0} and {1}".format(apple_blob, banana_blob)
```

Output:

```
apples and bananas
'apples and bananas'
```

3. N-grams

The TextBlob.ngrams() method returns a list of tuples of n successive words.

Code:

```
blob = TextBlob("Now is better than never.")
blob.ngrams(n=3)
```

Output:

```
[WordList(['Now', 'is', 'better']),
 WordList(['is', 'better', 'than']),
 WordList(['better', 'than', 'never'])]
```

Use sentence.start and sentence.end to get the indices where a sentence starts and ends within a TextBlob.

Code:

```
for s in zen.sentences:
    print(s)
    print("---- Starts at index {}, Ends at index
{}".format(s.start, s.end))
```

Output:

```
Beautiful is better than ugly.
---- Starts at index 0, Ends at index 30
Explicit is better than implicit.
---- Starts at index 31, Ends at index 64
Simple is better than complex.
---- Starts at index 65, Ends at index 95
```

4. Import text data

In real world applications, we usually need to import textual data from a text file, usually a raw text file. Let's try to first create a text file using a text editor (nano, or even Jupyter lab text editor). Open a blank text file and type in a sentence like:

"*This is a simple sentence that I typed for illustration purposes. I do not intend to use it for anything else.*"

Once you finish typing, save the text file with a name mytext.txt and proceed to your notebook again. This is the code we can use to import this text file and use textblob tagging on the data:

Code:

```
from textblob import TextBlob
file=open("mytext.txt")
t=file.read()
print(type(t))
bobo = TextBlob(t)
bobo.tags
```

Output:

```
<class 'str'>

[('This', 'DT'),
 ('is', 'VBZ'),
 ('a', 'DT'),
 ('simple', 'JJ'),
 ('sentence', 'NN'),
 ('that', 'IN'),
 ('I', 'PRP'),
 ('typed', 'VBD'),
 ('for', 'IN'),
 ('illustration', 'NN'),
 ('purposes', 'NNS'),
```

```
('I', 'PRP'),
('do', 'VBP'),
('not', 'RB'),
('intend', 'VB'),
('to', 'TO'),
('use', 'VB'),
('it', 'PRP'),
('for', 'IN'),
('anything', 'NN'),
('else', 'RB')]
```

Exercise 4.1.

Create a text file with the following content:

“Set on the fictional continents of Westeros and Essos, Game of Thrones has a large ensemble cast and follows several story arcs throughout the course of the show. The first major arc concerns the Iron Throne of the Seven Kingdoms of Westeros through a web of political conflicts among the noble families either vying to claim the throne or fighting for independence from whoever sits on it. The second major arc focuses on the last descendant of the realm's deposed ruling dynasty, who has been exiled to Essos and is plotting to return and reclaim the throne. The third follows the Night's Watch, a military order defending the realm against threats from beyond the Seven Kingdoms' northern border.”

Save it as Sample.txt. Now, perform a sentiment analysis to calculate the overall positivity or negativity on the Sample.txt.

5. Word cloud visualization

Word cloud is one of the most powerful and straightforward visualization methods when it comes to text data. The size of words is dependent on the occurrence frequency (text’s importance in the context), therefore it comes in handy for Natural Language Processing or machine learning projects or simply

text analysis. We will need the word cloud generator to create the visuals for us, and keep in mind that wordcloud depends on the essential libraries NumPy. We also need matplotlib to display or save the image locally. *Make sure to activate the virtual or conda env first before doing the next step.*

In your terminal (Linux / maxOS) or the command prompt (windows) type in the code below depends on your environment:

pip install wordcloud

If you are using conda:

conda install -c conda-forge wordcloud

For the next part, you need your body of the text from a text file. You also need to perform data preparation by cleaning the text using the following steps:

- Removing special characters
- Tokenization
- Lemmatization

In the previous part of the lesson, we demonstrate how you can perform textual data processing in the TextBlob library. Another useful library for textual data processing is NLTK (Natural Language Toolkit) which is popular among developers. In this part of the lesson, we will use the NLTK library for demonstration. Note that most of the functionalities of TextBlob and NLTK are similar. To learn more about the NLTK library, look at the official documentation here.

To install NLTK using conda, you can use the following command:

conda install anaconda::nltk

In particular, here we will use the 'stopwords' functionality of the NLTK library. Stopwords are common words that are often removed from text before processing for tasks like text analysis, information retrieval, or machine learning. These words, like "the," "and," "of," and "in," typically don't carry much semantic meaning on their own and can clutter the analysis. In textual data analysis, the command *stopwords=['some_word']* Specifies the word "some_word" as a stopword, meaning it will be excluded from the word cloud.

After having all the essential libraries installed and having the body of text ready, we need to ensure to import them.

Code:

```
import numpy as np
from wordcloud import WordCloud
import matplotlib.pyplot as plt
from nltk.corpus import stopwords
```

The body of the text can be saved in a variable called meta_raw, in this example the text from wikipedia about meta platforms company:

“Meta ranks among the largest American information technology companies, alongside other Big Five corporations Alphabet (Google), Amazon, Apple, and Microsoft. The company was ranked #31 on the Forbes Global 2000 ranking in 2023. In 2022, Meta was the company with the third-highest expenditure on research and development worldwide, with R&D expenditure amounting to US$35.3 billion. Meta has also acquired Oculus (which it has integrated into Reality Labs), Mapillary, CTRL-Labs, and a 9.99% stake in Jio Platforms; the company additionally endeavored into non-VR hardware, such as the discontinued Meta Portal smart displays line and partners with Luxottica through the Ray-Ban Stories series of smartglasses.”

Code:

```
meta_raw = "Meta ranks among the largest American
information technology companies, alongside other Big Five
corporations Alphabet (Google), Amazon, Apple, and
Microsoft. The company was ranked #31 on the Forbes Global
2000 ranking in 2023. In 2022, Meta was the company with
the third-highest expenditure on research and development
worldwide, with R&D expenditure amounting to US$35.3
```

```
billion. Meta has also acquired Oculus (which it has
integrated into Reality Labs), Mapillary, CTRL-Labs, and a
9.99% stake in Jio Platforms; the company additionally
endeavored into non-VR hardware, such as the discontinued
Meta Portal smart displays line and partners with
Luxottica through the Ray-Ban Stories series of
smartglasses."
```

We perform tokenization on the text:

Code:

```
meta_blob = TextBlob(meta_raw)
meta_text = meta_blob.words
```

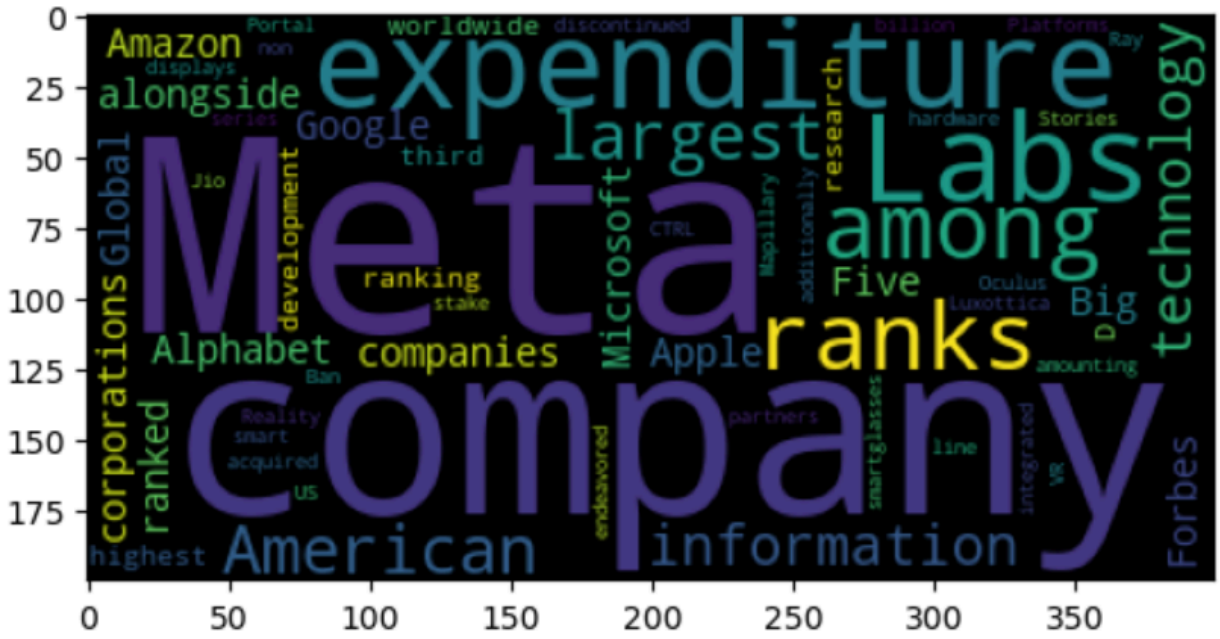
Now we have everything we need for generating word cloud. Create an object of class WordCloud with the name of your choice and call the generate( ) method.

Code:

```
# first, we convert our blob to string
meta_text_str = ' '.join(meta_text)
wc = WordCloud().generate(meta_text_str)
plt.imshow(wc)
```

Wow! Our first wordcloud is generated:

```
<matplotlib.image.AxesImage at 0x22556da8410>
```



As you can see, there are numbers along the axis, and we can turn them off by calling:

Code:

```
plt.axis("off")
```

```
<matplotlib.image.AxesImage at 0x22554ddeb50>
```

Note that WordCloud( ) takes in several arguments and allows us to customize the shape, color, etc. which we will get into in examples below. Here I'm naming my word cloud object "wc". I didn't pass in any arguments inside the WordCloud( ), leaving everything else as default. Let's look at what the simple Meta word cloud looks like by default. WordCloud( ) contains parameters including the following:

- background_color: Color of background
- max_words: The maximum number of unique words used
- stopwords: A stopword list to exclude the words you don't wish to display
- colormap: The color theme
- width: The width of the WordCloud image
- height: The height of the WordCloud image

Code:

```
wc = WordCloud(background_color='white', colormap =
'binary',
     stopwords = ['meta'], width = 800, height =
500).generate(meta_text)
```

```
plt.axis("off")
plt.imshow(wc)
```

```
<matplotlib.image.AxesImage at 0x22554d978d0>
```

Exercise 5.1.

Remove the words like "the," "and," "of," and "in," which typically don't carry much semantic meaning and plot the word cloud again using a new paragraph from Alphabet company wikipedia page.

References and Further Resources

1. Parts of the course material is adapted from TextBlob - Official Documentation

2. NLTK library - The official documentation

3. WordCloud library - The official documentation

4. WordClouds with Python - Meredith Young - Medium.com