drawing

Application of Deep Learning to Text and Image Data

Module 2, Lab 1: Processing Text

In this notebook, you will learn techniques to analyze and process text data. Text processing is known as *natural language processing (NLP)* and is an important topic because of how much information is communicated through text. Knowing how to handle text will help you build models that perform better and are more useful.

You will learn the following:

- What a word cloud is and how to create one
- How to use stemming and lemmatization
- What part-of-speech tagging is and how it impacts text processing
- How to use named entity recognition to sort data

You will be presented with two kinds of exercises throughout the notebook: activities and challenges.

Activity	**Chal**
No coding is needed for an activity. You try to understand a concept, answer questions, or run a code cell.	Challenges are where you car

---

Index

- [Word cloud]
- [Part-of-speech tagging]
- [Stemming and lemmatization]
- [Named entity recognition]

Initial Setup

First let's put everything in place.

```
!pip install -U -q -r requirements.txt
```

Show hidden output

Install the spaCy library. This will be used to perform some NLP tasks in the lab.

```
!python -m spacy download en_core_web_sm
```

Show hidden output

```
# Install PyStemmer
!pip install PyStemmer

# Import the dependencies
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt
import re, string
from Stemmer import Stemmer # Import from the module
import spacy
from spacy import displacy
import pandas as pd

#The import from Stemmer to Stemmer is correct. The problem was that the library was not ins
```

Show hidden output

Next, you need to create a function to preprocess text so that only real words, not special characters and numbers, are displayed.

```
# Preprocess text
def preProcessText(text):
    # Lowercase and strip leading and trailing white space
    text = text.lower().strip()

    # Remove HTML tags
    text = re.compile("<.*?>").sub("", text)

    # Remove punctuation
    text = re.compile("[%s]" % re.escape(string.punctuation)).sub(" ", text)

    # Remove extra white space
    text = re.sub("\s+", " ", text)

    # Remove numbers
    text = re.sub(r"[0-9]", "", text)

    return text
```

Word cloud

Word clouds, which are also known as *text clouds* or *tag clouds*, help you visualize text data by highlighting the important words or phrases. Word clouds convey crucial information at a glance by making commonly occurring words bigger and bolder. These clouds are commonly used to compare and contrast two pieces of text. Word clouds are also used to identify the topic of a document.

To create a word cloud, you will use WordCloud for Python.

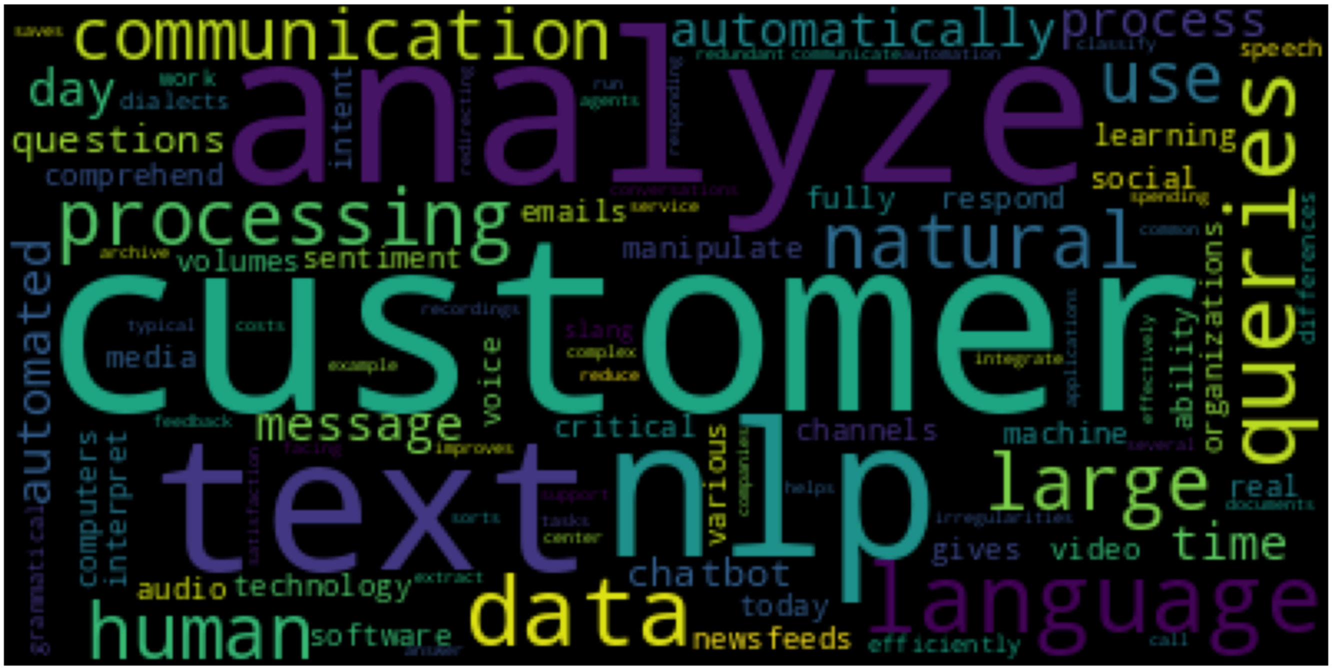
The following text is from the What Is Natural Language Processing (NLP)? page on aws.amazon.com.

```
text = "Natural language processing (NLP) is a machine learning technology that gives comput
ability to interpret, manipulate, and comprehend human language. Organizations today have la
of voice and text data from various communication channels like emails, text messages, socia
newsfeeds, video, audio, and more. They use NLP software to automatically process this data,
the intent or sentiment in the message, and respond in real time to human communication. \
Natural language processing (NLP) is critical to fully and efficiently analyze text and spee
It can work through the differences in dialects, slang, and grammatical irregularities typic
day-to-day conversations. \
Companies use it for several automated tasks, such as to: \
<li>Process, analyze, and archive large documents</li> \
<li>Analyze customer feedback or call center recordings</li> \
<li>Run chatbots for automated customer service</li> \
<li>Answer who-what-when-where questions</li> \
<li>Classify and extract text</li> \
You can also integrate NLP in customer-facing applications to communicate more effectively w
customers. For example, a chatbot analyzes and sorts customer queries, responding automatica
common questions and redirecting complex queries to customer support. This automation helps
costs, saves agents from spending time on redundant queries, and improves customer satisfact
```

```
# Remove stop words before generating the word cloud
wordcloud = WordCloud(stopwords=STOPWORDS, background_color="black", max_words=300)

# Clean up the text to prevent plotting punctuation and duplicate words (for example, 'Natur
wordcloud.generate(preProcessText(text))

plt.figure(figsize=(15, 10))
plt.axis("off")
plt.imshow(wordcloud);
```



Now that you have created a word cloud, do you see how it can help you quickly identify key words?

Note that the stop words were removed before the graphic was created. This is important so that words that don't impact the meaning of the text aren't overemphasized. Can you think of some examples of stop words?

In this example, you used the precompiled list of stop words that were curated by the WordCloud for Python project. You can print a list of the stop words to make sure that they cover the stop words that you expect.

```
# Show the list of stop words
", ".join(list(STOPWORDS))
```

```
'can't, if, when, that's, do, only, you've, itself, again, by, which, weren't, should,
we, not, doesn't, than, under, being, above, be, there's, since, what's, they've, each,
about, shan't, both, wouldn't, up, i'd, he'll, however, else, my, so, shouldn't, at, do
es, the, get, he, they, i'll, he'd, such, with, yours, into, they'll, like, while, nor,
was, she, we've, him, ours, themselves, then, are, doing, or, she'd, that, how, when's,
```

Part-of-speech tagging

The process of classifying words into their corresponding part of speech based on definition and context is called *part-of-speech tagging*, which is also known as *POS tagging*. A part-of-speech tagger processes a sequence of words and attaches a part-of-speech tag to each word.

For this lab, you will use the the Natural Language Toolkit [spaCy](). The **nlp() function** creates different token attributes, among them the one representing the token tag: **token.pos**. For example, the following tagged token combines the word *fly* with a noun part of speech tag, *NN*: `tagged_tok = ('fly', 'NOUN')`.

The following table provides the meanings for the tags from a lst of Universal POS tags:

Tag	Meaning
ADJ	Adjective
ADV	Adposition
ADP	Adverb
AUX	Auxiliary
CCONJ	Coordinating Conjuction
DET	Determiner
INTJ	Interjection
NOUN	Noun
NUM	Numeral
PART	Particle
PRON	Pronoun
PROPN	Proper Noun
PUNCT	Punctuation
SCONJ	Subordinating Conjuction
SYM	Symbol
VERB	Verb
X	Other

Now you can use the tagger to tag each token or word in the following text.

Important: Always remember to preprocess the text before tagging, as we have done before in this notebook.

```
# Text sample
text
```

```
'Natural language processing (NLP) is a machine learning technology that gives computer
s the ability to interpret, manipulate, and comprehend human language. Organizations to
day have large volumes of voice and text data from various communication channels like
emails, text messages, social media newsfeeds, video, audio, and more. They use NLP sof
tware to automatically process this data, analyze the intent or sentiment in the messag
```

Try it yourself!

Activity

To use the spaCy part-of-speech tagger, run the following cell.

Observe the tags that are assigned to each word, and use the table from a previous cell to understand the meaning of each tag.

```
# Part-of-speech tagging
nlp = spacy.load("en_core_web_sm")
doc = nlp(text)
token_text = [token.orth_ for token in doc]
token_lemma = [token.lemma_ for token in doc]
token_pos = [token.pos_ for token in doc]
pd.DataFrame(zip(token_text, token_pos),
             columns=['token_text', 'token_lemma'])
```

	token_text	token_lemma
0	Natural	ADJ
1	language	NOUN
2	processing	NOUN
3	(	PUNCT
4	NLP	PROPN
...	...	...
248	and	CCONJ
249	improves	VERB
250	customer	NOUN
251	satisfaction	NOUN
252	.	PUNCT

253 rows × 2 columns

Refer to the table in a previous cell to identify the tags that the spaCy tagger produces.

```
# uncomment the code bellow to display the dependency parse or named entities and their tags
#displacy.render(doc, style="dep")
```

Stemming and lemmatization

Stemming and lemmatization are two ways to process words so that a model will be more efficient. Both methods remove parts of words so that they can be grouped together.

For example, in the following sentence, "ning" would be removed from "running" so that "running" and "run" would be categorized the same.

The child enjoys **running**, so they **run** every day.

What could make stemming and lemmatization difficult to do properly?

Try it yourself!

Activity

In the next few sections, you will compare stemming and lemmatization.

Consider which text processing method is more suitable for the use case that is provided.

Stemming

Stemming is a rule-based system to convert words into their root forms by removing suffixes. This method helps to enhance similarities (if any) between sentences.

Examples:

"jumping", "jumped" -> "jump"

"cars" -> "car"

```
# Install PyStemmer
!pip install PyStemmer

# Import the dependencies
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt
import re, string
from Stemmer import Stemmer # Import from the module
import spacy
from spacy import displacy
import pandas as pd

#The import from Stemmer to Stemmer is correct. The problem was that the library was not ins
# Preprocess text
def preProcessText(text):
    # Lowercase and strip leading and trailing white space
    text = text.lower().strip()

    # Remove HTML tags
    text = re.compile("<.*?>").sub("", text)

    # Remove punctuation
    text = re.compile("[%s]" % re.escape(string.punctuation)).sub(" ", text)

    # Remove extra white space
    text = re.sub("\s+", " ", text)

    # Remove numbers
    text = re.sub(r"[0-9]", "", text)

    return text
# No longer needed. There is no function to list the algorithms, just instantiate the class
#print(Stemmer.algorithms())
```

```
# we will use english for this example
stemmer = Stemmer('english') #Instantiates the english stemmer

original_text = "   This is a message to be cleaned. It may involve some things like: <br>,
print(original_text)

# Cleaned text
cleaned_text = preProcessText(original_text)
print(cleaned_text)

# Use a tokenizer (nlp) and stemmer from the PyStemmer library
nlp = spacy.load("en_core_web_sm")
doc = nlp(original_text)
stemmed_sentence = []
original_sentence = []
# Tokenize the sentence
for token in doc:
    original_sentence.append(token.text)
    stemmed_sentence.append(stemmer.stemWord(token.text))

pd.DataFrame(zip(original_sentence, stemmed_sentence),
             columns=['token_text', 'token_stemmer'])

stemmed_text = " ".join(stemmed_sentence)
print(stemmed_text)
```

```
Requirement already satisfied: PyStemmer in /usr/local/lib/python3.11/dist-packages (2.2
   This is a message to be cleaned. It may involve some things like: <br>, ?, :, ''  adj
this is a message to be cleaned it may involve some things like adjacent spaces and tabs
    This is a messag to be clean . It may involv some thing like : < br > , ? , : , ''
```

```
# In the cell that failed, remove Stemmer.Stemmer
# Before: stemmer = Stemmer.Stemmer('english')
# After:
stemmer = Stemmer('english')


original_text = "   This is a message to be cleaned. It may involve some things like: <br>,
print(original_text)
```

```
   This is a message to be cleaned. It may involve some things like: <br>, ?, :, ''  adj
```

```
# Cleaned text
cleaned_text = preProcessText(original_text)
print(cleaned_text)
```

```
this is a message to be cleaned it may involve some things like adjacent spaces and tabs
```

```
# Use a tokenizer (nlp) and stemmer from the PyStemmer library
from Stemmer import Stemmer
import spacy
import pandas as pd
nlp = spacy.load("en_core_web_sm")
doc = nlp(original_text)
stemmed_sentence = []
original_sentence = []
# Create a new stemmer
stemmer = Stemmer('english')
# Tokenize the sentence
for token in doc:
    original_sentence.append(token.text)
    stemmed_sentence.append(stemmer.stemWord(token.text))

pd.DataFrame(zip(original_sentence, stemmed_sentence),
             columns=['token_text', 'token_stemmer'])
```

	token_text	token_stemmer
0		
1	This	This
2	is	is
3	a	a
4	message	messag
5	to	to
6	be	be
7	cleaned	clean
8	.	.
9	It	It
10	may	may
11	involve	involv
12	some	some
13	things	thing
14	like	like
15	:	:
16	<	<
17	br	br
18	>	>
19	,	,
20	?	?
21	,	,
22	:	:
23	,	,
24	"	"
25		
26	adjacent	adjac
27	spaces	space
28	and	and
29	tabs	tab

30

31 . .

32

```
stemmed_text = " ".join(stemmed_sentence)
print(stemmed_text)
```

From the output of the previous code cell, you can see that stemming isn't perfect. It makes mistakes, such as "messag", "involv", and "adjac". Stemming is a rule-based method that sometimes mistakenly removes suffixes from words. It does run quickly, which makes it appealing to use for massive datasets.

Lemmatization

If you aren't satisfied with the result of stemming, you can use the lemmatization instead. This method usually requires more work but gives better results.

Lemmatization needs to know the correct word position tags, such as "noun", "verb", or "adjective". You need to use another spaCy function to feed this information to the lemmatizer.

The cell below uses part of the full list of position tags listed in the previous session `Part-of-speech tagging`.

```
# get the original and the lemmatized the word/token
token_lemma = [token.lemma_ for token in doc]
token_text = [token.orth_ for token in doc]
pd.DataFrame(zip(token_text, token_lemma),
             columns=['token_text', 'token_lemma'])
```

	token_text	token_lemma
0		
1	This	this
2	is	be
3	a	a
4	message	message
5	to	to
6	be	be
7	cleaned	clean
8	.	.
9	It	it
10	may	may
11	involve	involve
12	some	some
13	things	thing
14	like	like
15	:	:
16	<	<
17	br	br
18	>	>
19	,	,
20	?	?
21	,	,
22	:	:
23	,	,
24	"	"
25		
26	adjacent	adjacent
27	spaces	space
28	and	and
29	tabs	tab

30

31 . .

32

```
lemmatized_text = " ".join(token_lemma)
print(lemmatized_text)
```

```
this be a message to be clean . it may involve some thing like : < br > , ? , : , ''
```

How do the results compare? Is the lemmatized text better than the stemmed text?

Named entity recognition

Named entity recognition involves identification of key information in text and then classifying that information into predefined categories, such as person, organization, place, or date. This is one of the most popular NLP tasks.

For this section, you will use [spaCy](). The following table lists the categories and meanings of the category labels that the spaCy module uses.

Category	Meaning
CARDINAL	Numerals that don't fall under another type
DATE	Absolute or relative dates or periods
EVENT	Named hurricanes, battles, wars, sports events, and so on
FAC	Buildings, airports, highways, bridges, and so on
GPE	Countries, cities, states
LANGUAGE	Any named language
LAW	Named documents made into laws
LOC	Non-GPE locations, mountain ranges, bodies of water
MONEY	Monetary values, including unit
NORP	Nationalities, or religious or political groups
ORDINAL	"first", "second", and so on
ORG	Companies, agencies, institutions, and so on
PERCENT	Percentage, including "%"
PERSON	People, including fictional
PRODUCT	Objects, vehicles, foods, and so on (not services)