Sentiment Analysis & Recommender Systems - Part 1

One should look for what is and not what he thinks should be. (Albert Einstein)

Who we are

- Data Society's mission is to **integrate Big Data and machine learning best practices across entire teams** and empower professionals to identify new insights
- We provide:
  - High-quality data science training programs
  - Customized executive workshops
  - Custom software solutions and consulting services
- Since 2014, we've worked with thousands of professionals to make their data work for them

Best practices for virtual classes

1. Find a quiet place that is free of distractions. Headphones are recommended.
2. Remove or silence alerts from cell phones, e-mails, etc.
3. Participate in discussions and ask questions. This course is interactive!
4. Give your honest feedback so we can troubleshoot problems and improve the course.

Module completion checklist

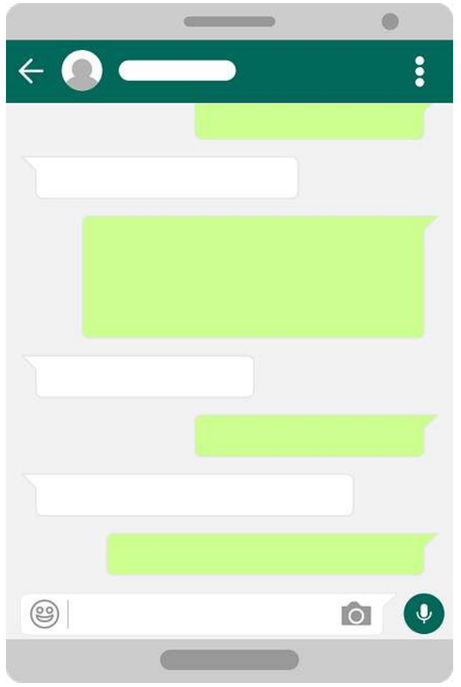
Objective	Complete
Introduction to Sentiment analysis and it’s use cases	
Explain and define the outcome of bag-of-words analysis	
Pre-process the NY Times data and create a DTM	
Summarize the concept of sentiment analysis	
Classify each cleaned sentence from the cleaned text file as positive, negative, or neutral and store labels	

Are the Humans “Happy?”

- Can algorithms pick up on human “feelings”?
- When we are in the realm of text analysis, we can get close with **sentiment analysis**

Sentiment in customer support

- Working with **customer support data**
  - Analyzing positive and negative sentiment, trying to improve customer service
- Working with **chat messages**
  - Analyzing positive and negative sentiment, trying to detect and improve the biggest triggers for unhappy customers



Sentiment in product development

- Working with **consumer reviews**
  - Analyzing positive and negative sentiment, trying to improve product offerings
- Working with **movie reviews**
  - Analyzing positive and negative sentiment, trying to improve movie selection offered with subscription service

Large Text/Historical uses

- After one of the largest releases of email in corporate history due to legal action, thousands of Enron communications were made available for text analysis.
- Hundreds of new studies sprouted up pursuing questions as diverse as social network theory, community and anomaly detection, gender and communication within organizations, behavioral change during an organizational crisis, and insularity and community formation.

You can read about that work ***here***

Sentiment analysis - use cases

- Sentiment mining is used in many areas:
  - opinion mining
  - reputation monitoring
  - business analytics
- It helps businesses understand their customers' experience
- **How do you see sentiment analysis as a helpful tool within your job/field?**

Module completion checklist

Objective	Complete
Introduction to Sentiment analysis and it’s use cases	✔
Explain and define the outcome of bag-of-words analysis	
Pre-process the NY Times data and create a DTM	
Summarize the concept of sentiment analysis	
Classify each cleaned sentence from the cleaned text file as positive, negative, or neutral and store labels	

Directory settings

- In order to maximize the efficiency of your workflow, you should encode your directory structure into `variables`
- We will use the `pathlib` library
- Let the `main_dir` be the variable corresponding to your `skillsoft-sentiment-analysis-2021` folder
- `data_dir` be the variable corresponding to your `data` folder

```
# Set 'main_dir' to location of the project folder
from pathlib import Path
home_dir = Path(".").resolve()
main_dir = home_dir.parent
```

```
data_dir = str(main_dir) + "/data"
```

- We'll be using this variable to load the data present in the data folder!

Loading packages

```
# Helper packages.
import os
import pandas as pd
import numpy as np
import pickle
import matplotlib.pyplot as plt
```

```
# Packages with tools for text processing.
import nltk
nltk.download('all')
import nltk.data
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem.porter import PorterStemmer
```

```
# Packages for working with text data and analyzing sentiment
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from sklearn.feature_extraction.text import CountVectorizer
```

Datasets for today

In class

- **NY Times article snippets**
- We will analyze the sentiment of NY Times article snippets
- We will classify the snippets into positive or negative sentiments

Exercises

- **Movie reviews**
- We will analyze the sentiment of movie reviews from a popular review website
- We will classify the reviews into positive or negative sentiments

“Bag-of-words” analysis

- How do we quantify and compute on text data? - *We need to translate words into numbers*
- How do we translate words into numbers? - *The simplest solution is to* **count** *them*
- The analysis of text data based on word counts (a.k.a. frequencies) in documents is called **bag-of-words**

Index	Word	Freq	%
A	Apple	5	20
B	Book	7	28
C	Cat	13	52

“Bag-of-words” analysis: key elements

Steps in Bag-of-words analysis
A corpus of documents cleaned and processed in a certain way • Convert all words to lower case • Remove punctuation, numbers, and special characters • Remove stopwords • Stem words to their root form (and sometimes lemmatize)
A Document-Term Matrix (DTM): with counts of each word recorded for each document
A transformed representation of a Document-Term Matrix (i.e. weighted with TF-IDF weights)

- To perform the sentiment analysis, we would start with loading a corpus, process it and convert it into a DTM!

Module completion checklist

Objective	Complete
Introduction to Sentiment analysis and it’s use cases	✔
Explain and define the outcome of bag-of-words analysis	✔
Pre-process the NY Times data and create a DTM	
Summarize the concept of sentiment analysis	
Classify each cleaned sentence from the cleaned text file as positive, negative, or neutral and store labels	

Load text data

```
# Load corpus from a csv (for Mac).
NYT = pd.read_csv(data_dir + '/NYT_article_data.csv')
print(NYT.columns)
```

```
Index(['web_url', 'headline', 'snippet', 'word_count', 'source',
       'type_of_material', 'date'],
      dtype='object')
```

```
# Isolate the snippet column.
NYT_snippet = NYT["snippet"]
# Look at a sample of the snippets column, 0-10.
print(NYT["snippet"][0:10])
```

```
0    Pakistan's struggling batsmen must find a way ...
1    The National Football League is under the micr...
2    Hitting a hot streak at the right time will be...
3    Pope Francis ushered in the New Year with an o...
4    Chris Froome will not defend his Giro d'Italia...
5    Pakistan's former Prime Minister Nawaz Sharif ...
6    Thousands of demonstrators marched in Hong Kon...
7    Nick Kyrgios started his Brisbane Open title d...
8    British police confirmed on Tuesday they were ...
9    Marcellus Wiley is still on the fence about le...
Name: snippet, dtype: object
```

Tokenization: split each snippet into words

- NLTK's functions operate on **tokens**
- A **token** is the smallest unit of text of interest - in our case, it will be a **word**
- We will use `word_tokenize()` method to split each snippet into tokens
- Below is a list comprehension that:
  i. Takes each snippet from the series of snippets we created - `NYT_snippet`
  ii. Iterates through the each snippet using `word_tokenize`
  iii. Outputs a large list of tokenized snippets

```
# Tokenize each snippet into a large list of tokenized snippets.
NYT_tokenized = [word_tokenize(NYT_snippet[i]) for i in range(0, len(NYT_snippet))]
```

“Bag-of-words” analysis: cleaning text flow

1. Convert all characters to lower case
2. Remove stop words
3. Remove punctuation, numbers, and all other symbols that are not letters of the alphabet
4. Stem words
5. Remove extra white space (if needed)

Implementing pre-processing steps on a corpus

- Let's implement those text cleaning steps on the entire corpus of article snippets

```
# Create a list for clean snippets.
NYT_clean = [None] * len(NYT_tokenized)
```

```
# Create a list of word counts for each clean snippet.
word_counts_per_snippet = [None] * len(NYT_tokenized)
```

```
# Process words in all snippets.
for i in range(len(NYT_tokenized)):
    # 1. Convert to lower case.
    NYT_clean[i] = [snippet.lower() for snippet in NYT_tokenized[i]]

    # 2. Remove stop words.
    stop_words = stopwords.words('english')
    NYT_clean[i] = [word for word in NYT_clean[i] if not word in stop_words]

    # 3. Remove punctuation and any non-alphabetical characters.
    NYT_clean[i] = [word for word in NYT_clean[i] if word.isalpha()]

    # 4. Stem words.
    NYT_clean[i] = [PorterStemmer().stem(word) for word in NYT_clean[i]]

    # Record the word count per snippet.
    word_counts_per_snippet[i] = len(NYT_clean[i])
```

Inspect results

```
print(NYT_clean[0][:10])
```

```
['pakistan', 'struggl', 'batsmen', 'must', 'find', 'way', 'handl', 'south', 'africa',
'potent']
```

```
print(NYT_clean[5][:10])
```

```
['pakistan', 'former', 'prime', 'minist', 'nawaz', 'sharif', 'appeal', 'convict', 'prison',
'sentenc']
```

```
print(NYT_clean[10][:10])
```

```
['still', 'reckon', 'fallout', 'emmett', 'till', 'paint', 'chasten', 'artist', 'reveal',
'controversi']
```

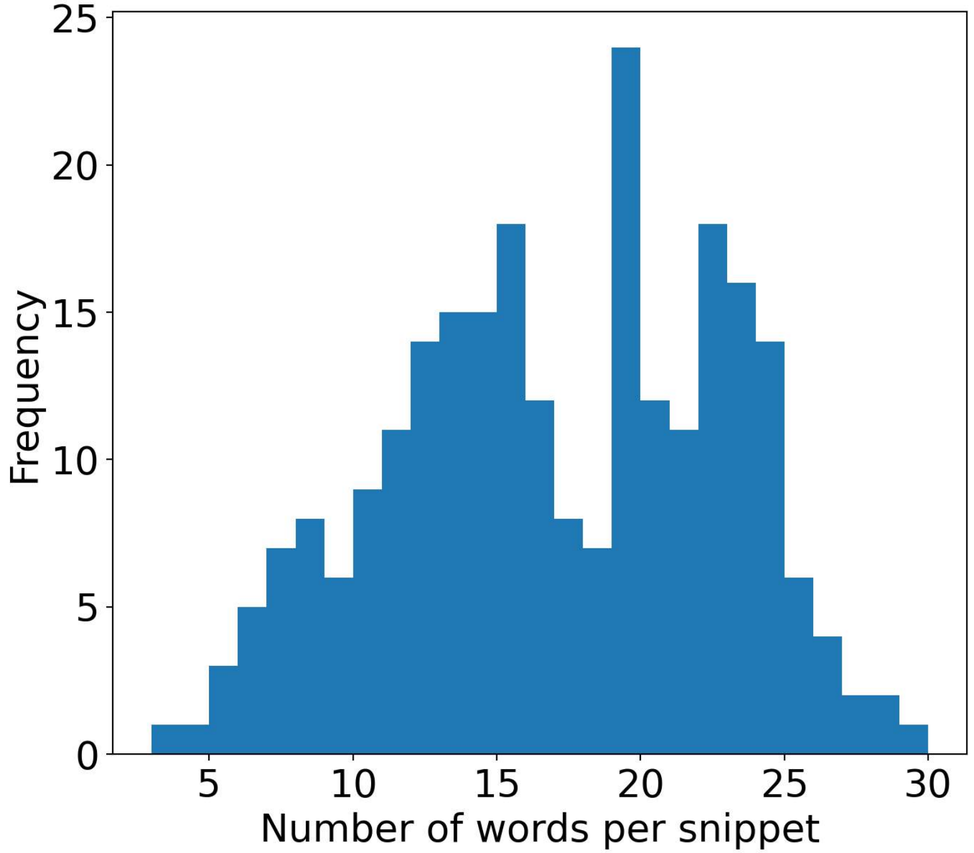
Removing empty and very short snippets

- Let's take a look at total word counts per snippet (for the first 10)

```
print(word_counts_per_snippet[:10])
```

```
[24, 12, 19, 20, 19, 15, 23, 15, 22, 27]
```

```
# Plot a histogram for word counts per snippet,
set bins to number of unique values in the list.
plt.hist(word_counts_per_snippet, bins =
len(set(word_counts_per_snippet)))
plt.xlabel('Number of words per snippet')
plt.ylabel('Frequency')
```



Removing empty and very short snippets (cont'd)

- After cleaning, we look to see if there are snippets that might not be meaningful anymore
- In this case, we will look for any snippets that contain under 5 words

```
# Convert word counts list and snippets list to numpy arrays.
word_counts_array = np.array(word_counts_per_snippet)
NYT_array = np.array(NYT_clean, dtype=object)
print(len(NYT_array))
```

```
250
```

```
# Find indices of all snippets where there are greater than or equal to 5 words.
valid_snippets = np.where(word_counts_array >= 5)[0]
print(len(valid_snippets))
```

```
248
```

Removing empty and very short snippets (cont'd)

- We now only keep all snippets over 5 words

```
# Subset the NYT_array to keep only those where there are at least 5 words.
NYT_array = NYT_array[valid_snippets]
print(len(NYT_array))
```

```
248
```

- And convert the array back to a list so we can continue analysis

```
# Convert the array back to a list.
NYT_clean = NYT_array.tolist()
print(NYT_clean[:3])
```

```
[['pakistan', 'struggl', 'batsmen', 'must', 'find', 'way', 'handl', 'south', 'africa',
'potent', 'pace', 'attack', 'claw', 'way', 'back', 'seri', 'second', 'test', 'start',
'like', 'live', 'newland', 'wicket', 'thursday'], ['nation', 'footbal', 'leagu',
'microscop', 'lack', 'minor', 'head', 'coach', 'recent', 'slew', 'fire', 'leagu'], ['hit',
'hot', 'streak', 'right', 'time', 'goal', 'golf', 'top', 'male', 'profession', 'year',
'new', 'calendar', 'cram', 'major', 'championship', 'super', 'busi', 'stretch']]
```

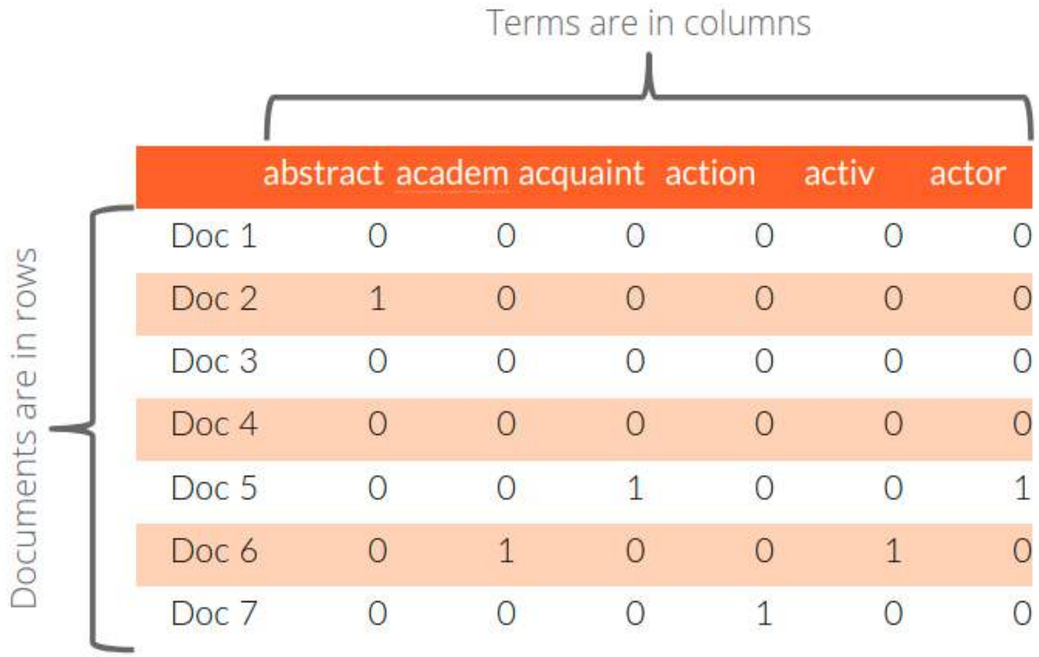
- **Remember, the threshold of max/min words for each instance will vary based on subject matter**

Save processed text to file using .join()

```
# Join words in each snippet into a single character string.
NYT_clean_list = [' '.join(snippet) for snippet in NYT_clean]
print(NYT_clean_list[:5])
```

```
['pakistan struggl batsmen must find way handl south africa potent pace attack claw way back
seri second test start like live newland wicket thursday', 'nation footbal leagu microscop
lack minor head coach recent slew fire leagu', 'hit hot streak right time goal golf top male
profession year new calendar cram major championship super busi stretch', 'pope franci usher
new year ode motherhood tuesday remind faith mother exampl embrac best antidot today
disjoint world solitud miseri', 'chri froom defend giro titl year choos focu win fifth tour
de franc crown instead team sky announc tuesday']
```

What is a DTM?



	abstract	academ	acquaint	action	activ	actor
Doc 1	0	0	0	0	0	0
Doc 2	1	0	0	0	0	0
Doc 3	0	0	0	0	0	0
Doc 4	0	0	0	0	0	0
Doc 5	0	0	1	0	0	1
Doc 6	0	1	0	0	1	0
Doc 7	0	0	0	1	0	0

- **Document-term matrix** is simply a matrix of unique words counted in each document:
  - Documents are arranged in the rows
  - Unique terms are arranged in columns
- The corpus **vocabulary** consists of all of the unique terms (i.e. column names of DTM) and their total counts across all documents (i.e. column sums)

Create DTM with CountVectorizer

- To create a Document-Term matrix, we will use `CountVectorizer` from `scikit-learn` library's `feature_extraction` module for working with text
- It takes a `list` of character strings that represents the documents (i.e. our snippets) as the main argument passed to its `fit_transform()` method:
  - `.fit_transform(list_of_documents)`
- It returns a 2D array (i.e. a matrix) with documents in rows and terms in columns - the **DTM**

Create a DTM

```
# Initialize `CountVectorizer`.
vec = CountVectorizer()
```

```
# Transform the list of snippets into DTM.
X = vec.fit_transform(NYT_clean_list)
print(X.toarray()) #<- to show output as a matrix
```

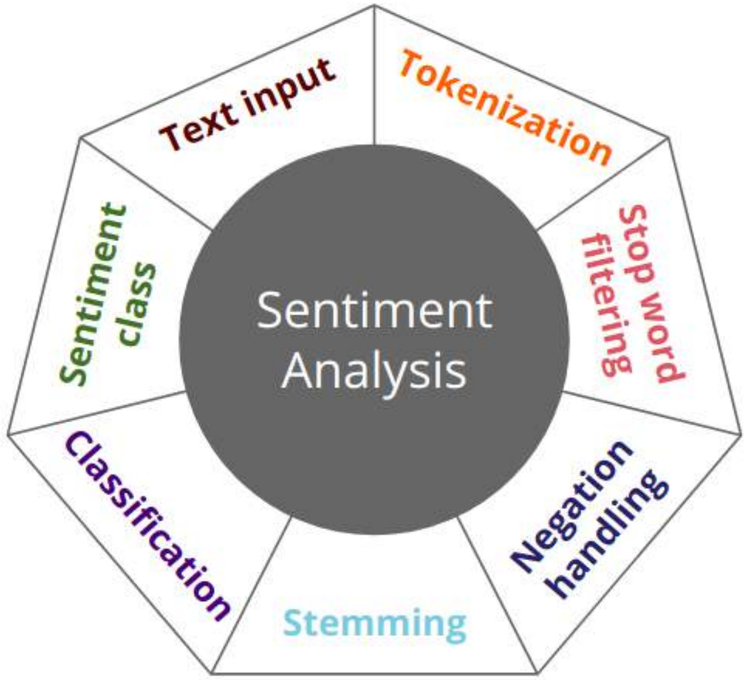
```
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 1 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
```

Module completion checklist

Objective	Complete
Introduction to Sentiment analysis and it’s use cases	✔
Explain and define the outcome of bag-of-words analysis	✔
Pre-process the NY Times data and create a DTM	✔
Summarize the concept of sentiment analysis	
Classify each cleaned sentence from the cleaned text file as positive, negative, or neutral and store labels	

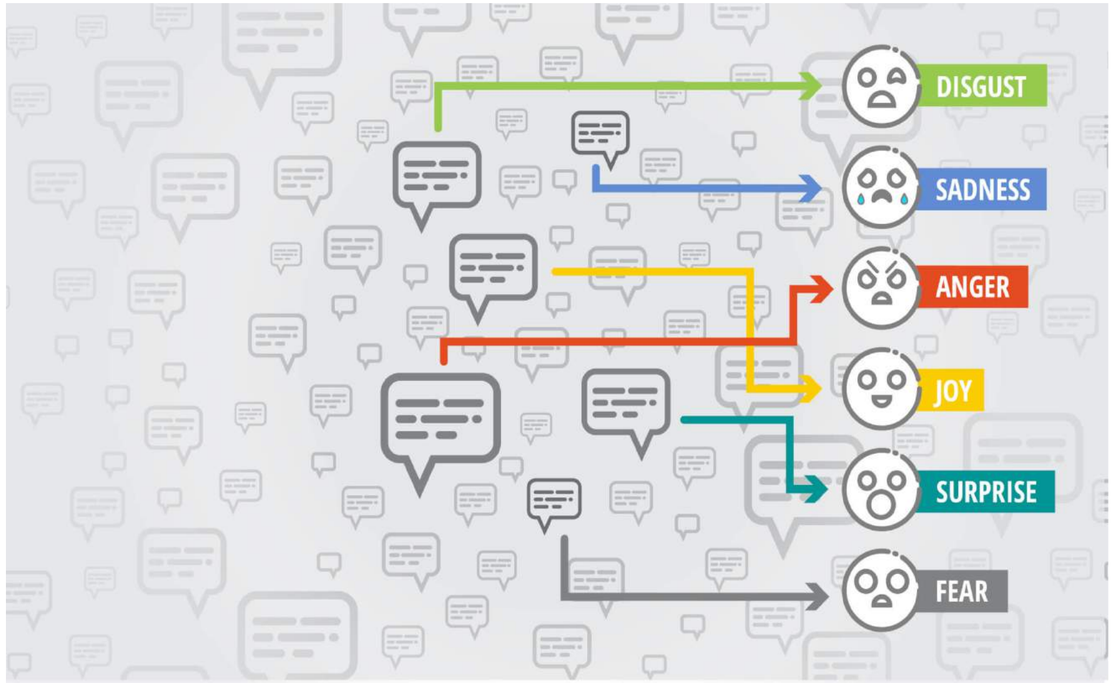
Sentiment analysis - steps

- There are **many ways to build out sentiment analysis**, including what we will use today
- Some of the processes will use more complex methodology / algorithms, but **today, we want to understand the basis of sentiment analysis**
- **Using these base steps, you will be able to build upon it** on your own as you continue using sentiment analysis



Sentiment analysis in Python

- **Python is very powerful when it comes to text mining, as you have seen**
- We will split the process we will be using into two parts for simplicity - text classification and model building
- **We will perform text classification today using `NLTK` and assign the labels to the text sentences in our dataset**



Sentiment analysis in Python

Text classification

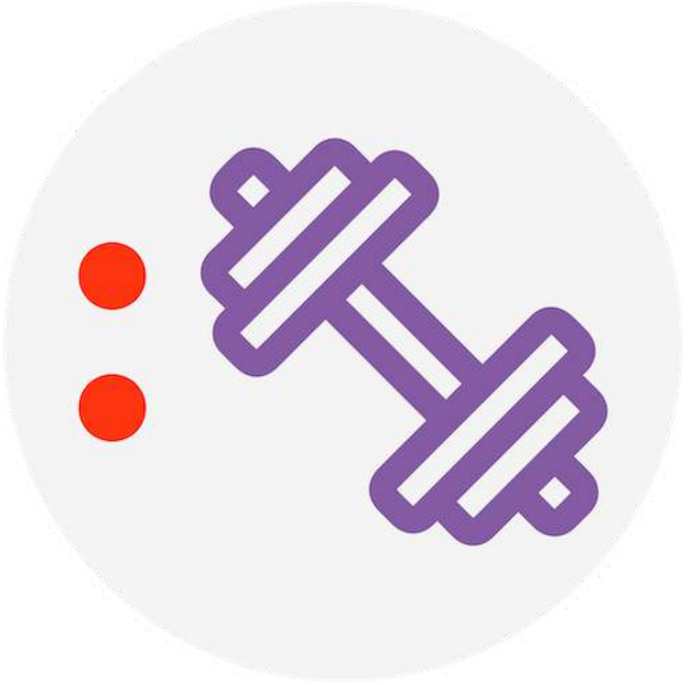
- **Clean** the NYTimes data and **prepare** a DTM
- **Classify** articles using NLTK's `SentimentIntensityAnalyzer`

Model building

- **Split the DTM into train and test datasets**, including the sentiment labels for each article
- **Build a logistic regression model** on the train dataset that classifies texts as one of the two categories, "negative" or "positive"
- **Analyze results** by predicting on the test set and on new data
- **Optimize the model** after inspecting results
- **Update the model** when new data comes in to have a continuously updated sentiment model

Knowledge check 1

Exercise 1



Module completion checklist

Objective	Complete
Introduction to Sentiment analysis and it’s use cases	✔
Explain and define the outcome of bag-of-words analysis	✔
Pre-process the NY Times data and create a DTM	✔
Summarize the concept of sentiment analysis	✔
Classify each cleaned sentence from the cleaned text file as positive, negative, or neutral and store labels	

Text classification

- We are going to classify each of our newspaper snippets using the `SentimentIntensityAnalyzer` function from the `vader` package
- They will be classified by document and classified as either:
  - “negative”
  - “positive”

Text classification - introducing vader

- As mentioned earlier, we will be using the `SentimentIntensityAnalyzer` function from the `vader` package, which is in the `NLTK` library
- We loaded this function earlier, let's take a quick look at what it does:
  - **VADER** = **V**alence **A**ware **D**ictionary for s**E**ntiment **R**easoning
  - `vader` sentiment analysis relies on a dictionary which maps lexical features to emotion intensities called *sentiment scores*
  - `vader.SentimentIntensityAnalyzer` will return a score in the range -1 to 1 from most negative to positive
  - The sentiment score is calculated by summing up the sentiment scores of each `vader` dictionary listed word in the sentence
  - `vader` works best on short documents, like this example, and tweets and other types of messages
  - You can go to the ***nltk.sentiment.vader module documentation*** page for the source code

Text classification - classify

- We will now use `vader.SentimentIntensityAnalyzer` to classify our loaded and cleaned snippets
- Within this function, we will be specifically using `SentimentIntensityAnalyzer.polarity_scores`
- This will give us a score between -1 and 1 and then compound the negative, neutral, and positive valences to give us a combined score, which is the score we will use today

Text classification - classify (cont'd)

class
nltk.sentiment.vader.**SentimentIntensityAnalyzer**(*lexicon_file='sentiment/vader_lexicon.zip/vader_lexicon/vader_lexicon.txt'*) [source]

Bases: `object`

Give a sentiment intensity score to sentences.

make_lex_dict() [source]

Convert lexicon file to a dictionary

polarity_scores(*text*) [source]

Return a float for sentiment strength based on the input text. Positive values are positive valence, negative value are negative valence.

score_valence(*sentiments*, *text*) [source]

sentiment_valence(*valence*, *sentitext*, *item*, *i*, *sentiments*) [source]

nltk.sentiment.vader.**allcap_differential**(*words*) [source]

Check whether just some words in the input are ALL CAPS

Parameters: **words** (*list*) – The words to inspect
Returns: *True* if some but not all items in *words* are ALL CAPS

nltk.sentiment.vader.**negated**(*input_words*, *include_nt=True*) [source]

Determine if input contains negation words

nltk.sentiment.vader.**normalize**(*score*, *alpha=15*) [source]

Normalize the score to be between -1 and 1 using an alpha that approximates the max expected value

nltk.sentiment.vader.**scalar_inc_dec**(*word*, *valence*, *is_cap_diff*) [source]

Check if the preceding words increase, decrease, or negate/nullify the valence

Text classification - classify (cont'd)

- Let's take a look at what the function outputs when iterating through the first 5 cleaned sentences

```
# Initialize the `SentimentIntensityAnalyzer().`
sid = SentimentIntensityAnalyzer()

# Iterate through each sentence printing out the scores for each.
for sentence in NYT_clean_list[:5]:
     print(sentence)
     ss = sid.polarity_scores(sentence)
     for k in ss:
         print('{0}: {1}, '.format(k, ss[k]), end='')
```

```
pakistan struggl batsmen must find way handl south africa potent pace attack claw way back
seri second test start like live newland wicket thursday
neg: 0.112, neu: 0.797, pos: 0.091, compound: -0.1531, nation footbal leagu microscop lack
minor head coach recent slew fire leagu
neg: 0.32, neu: 0.68, pos: 0.0, compound: -0.5719, hit hot streak right time goal golf top
male profession year new calendar cram major championship super busi stretch
neg: 0.0, neu: 0.65, pos: 0.35, compound: 0.8225, pope franci usher new year ode motherhood
tuesday remind faith mother exampl embrac best antidot today disjoint world solitud miseri
neg: 0.0, neu: 0.72, pos: 0.28, compound: 0.7906, chri froom defend giro titl year choos
focu win fifth tour de franc crown instead team sky announc tuesday
neg: 0.0, neu: 0.826, pos: 0.174, compound: 0.5859,
```

Text classification - classify (cont'd)

- We use `SentimentIntensityAnalyzer` to write a function that:
  - takes the cleaned text, runs the `SentimentIntensityAnalyzer` on each snippet
  - outputs results in four scores: **"neg","neu","pos","compound"**
  - takes the "compound" score and uses that to analyze if the snippet is negative or positive
  - Returns the labels for each snippet in a list

Text classification - classify (cont'd)

```
# This function outputs a list of labels for snippet:
def sentiment_analysis(texts):
        list_of_scores = []
        for text in texts:
            sid = SentimentIntensityAnalyzer()
            compound = sid.polarity_scores(text)["compound"]
            if compound >= 0:
                list_of_scores.append("positive")
            else:
                list_of_scores.append("negative")
        return(list_of_scores)
```

```
score_labels =
sentiment_analysis(NYT_clean_list)
```

```
print(score_labels[1:5])
```

```
['negative', 'positive',
'positive', 'positive']
```

- You now have a list `score_labels` that contains a sentiment classification for each snippet
- These will be used as your `y` variable when you build your train and test datasets

Pickle - what?

- We are going to pause for a moment to learn about the library `pickle`
- When we have objects we want to carry over and do not want to rerun code, we can `pickle` these objects
- In other words, `pickle` will help us **save objects** from one script/ session and **pull them up** in new scripts
- How do we do that? We use a function in Python called `pickle`
- It is similar to **flattening** a file
  - **Pickle/saving:** a Python object is converted into a byte stream
  - **Unpickle/loading:** the inverse operation where a byte stream is converted back into an object

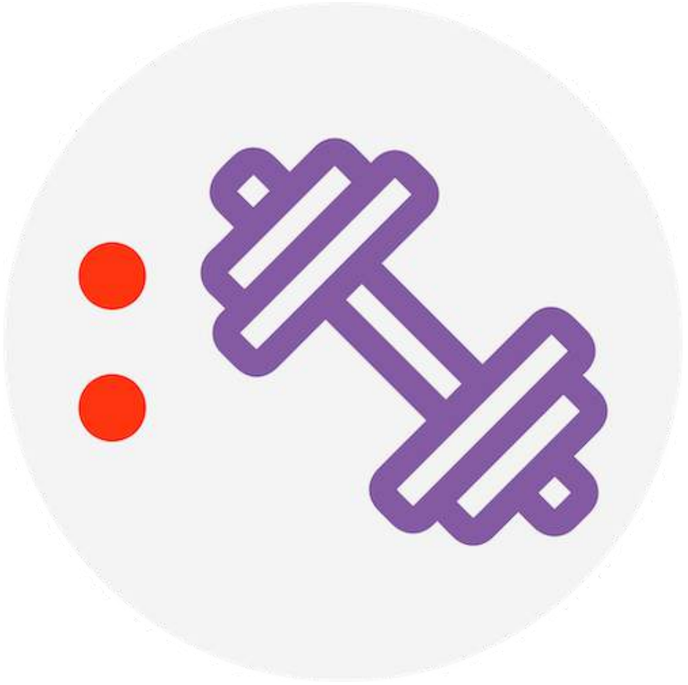
Save results as a pickle

- **Note:** This is applicable only when you have access to your file management system or local data directories
  - Nevertheless, it's a standard practice to save intermediate data to prevent re-running code
- Pickle the generated data for next steps

```
pickle.dump(NYT_clean_list, open(data_dir + '/NYT_clean_list.sav', 'wb'))
pickle.dump(score_labels, open(data_dir + '/score_labels.sav', 'wb'))
pickle.dump(X, open(data_dir + '/DTM_matrix.sav', 'wb'))
```

Knowledge check 2

Exercise 2



Module completion checklist

Objective	Complete
Introduction to Sentiment analysis and it’s use cases	✔
Explain and define the outcome of bag-of-words analysis	✔
Pre-process the NY Times data and create a DTM	✔
Summarize the concept of sentiment analysis	✔
Classify each cleaned sentence from the cleaned text file as positive, negative, or neutral and store labels	✔

Summary

- So far, we have
  - learned the concept of sentiment analysis and its use cases
  - classified sentences and labeled them using `vader` package
- In the next session, we'll learn about the concept of logistic regression and use it to classify our text sentences based on the labels we generated today!
- Until then, stay excited!

This completes our module