Natural Language Processing

Peter Ganong and Maggie Shi

November 20, 2024

Table of contents I

Language Models

Roadmap

- Define natural language processing and capabilities
- Introduce language models
- `spacy` package and language model: `en_core_web_sm`

Natural Language Processing

- **Natural language processing** allows machines to "understand" human language
- Uses linguistics and machine learning to text into usable data
- There is a wealth of information stored in text: "text as data"
- Examples of use cases: sentiment analysis, text prediction, speech-to-text

Language Models

- **Language models**: statistical model that understands and generates text based on patterns from pre-trained and hand-tagged text data
  - Parts of speech
  - Relationships between words
  - "Base" versions of words (the verb 'to walk' may appear as 'walk', 'walked', 'walks' or 'walking')
- Training data depends on the model: news articles, web pages, books
- After the model has been trained, then it can be applied to any sentence/document and it will make a data-driven guess as to how each word/sentence is structured
- Side note: GPTs are *large* language models (hence the term LLMs) – basic NLP tools consist of *small* language model

Capabilities of Language Models: Technical

Cover these in class

- **Tokenizing**: splits text into **tokens** – usually a word, punctuation, or white space
- **Named entity detection**: identifies people, organizations, locations
- **Dependency parser**: analyzes syntax structure of sentence

Will not cover these for time reasons, but good to know about

- **Parts-of-speech tagger**: assigns parts-of-speech to tokens
- **Lemmatizer**: converts words to their base form (e.g., texts $\rightarrow$ text, coding $\rightarrow$ code, fastest $\rightarrow$ fast)

Capability of Language Models: Conceptual

- We can use NLP methods to answer questions about a piece of text in a data-driven way
- We will discuss 3 use cases below that answer the following questions:
  - What is the sentiment expressed in a document?
  - Who/what is being discussed?
  - What is being said about a particular topic?

SpaCy package

- SpaCy is the most popular toolkit for NLP in Python
- Optimized to handle large volumes of text
- Includes pre-trained libraries in multiple languages – we don't need to train it
  - We will use a popular *English* language model: `"en_core_web_sm"` (English Core Web Small)

spaCy

- From **command line**, install the spaCy package

```
$ pip install spacy
```

spaCy

- From **command line**, install the spaCy package

```
$ pip install spacy
```

- We need to install the *language model* as well (in **command line**)

```
$ python -m spacy download en_core_web_sm
```

spaCy

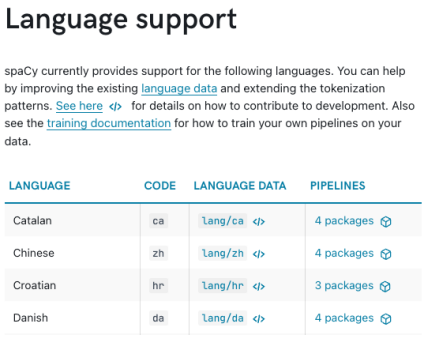
- To start using spaCy, import the package and then load the language library

```
import spacy

nlp = spacy.load("en_core_web_sm")
```

Language Models

- Note that spaCy has many languages beyond English in its library!
- Potentially useful if your projects are not about an English-speaking context



Language support

spaCy currently provides support for the following languages. You can help by improving the existing language data and extending the tokenization patterns. See here </> for details on how to contribute to development. Also see the training documentation for how to train your own pipelines on your data.

LANGUAGE	CODE	LANGUAGE DATA	PIPELINES
Catalan	ca	lang/ca </>	4 packages
Chinese	zh	lang/zh </>	4 packages
Croatian	hr	lang/hr </>	3 packages
Danish	da	lang/da </>	4 packages

Intro to using spaCy

- Next, we can load some text into the model using `nlp`

```
text = "The unemployment rate is good right now. I wouldn't want to stay at
↪  this level of inflation, though."

doc = nlp(text)
type(doc)
```

```
spacy.tokens.doc.Doc
```

- Like we saw with `BeautifulSoup`, the `spaCy` package has "parsed" through the text, so we can apply various methods to the text

Intro to using spaCy

- `.sents` property allows us to iterate over sentences
- Output is a "generator" object – like a memory-efficient version of a list
- Since we're working with sentences that are small enough, we can just convert it into a list

```
sents = doc.sents
sents_list = list(doc.sents)
print(sents_list[0])
```

```
The unemployment rate is good right now.
```

```
print(sents_list[1])
```

```
I wouldn't want to stay at this level of inflation, though.
```

```
print(sents_list[0][1])
```

```
unemployment
```

Tokenizing

- A basic NLP functionality is **tokenizing** – splitting text up into units of analysis
- Tokenizing in and of itself isn't a useful function, but many of the NLP functionalities depend on it

```
for token in sents_list[1]:
    print(token.text)
```

```
I
would
n't
want
to
stay
at
this
level
of
inflation
```

Tokenizing

- Tokens are usually a word
- But for `"wouldn't"`, it splits it into `"would"` and `"n't"` to distinguish that they are two different parts of speech, with very different functions within the sentence
  - `"would"`: auxilary verb
  - `"n't"`: negation

Summary

- Language models are pre-trained models that parse through and "understand" text
- In Python: `spacy` package
- Tokenizing is a basic NLP functionality

Use Case 1: Sentiment Analysis

Sentiment Analysis: Roadmap

- Introduce sentiment analysis and `spacytextblob`
- Example with presidential debate
- Do-pair-share

Sentiment Analysis

- *What is the sentiment expressed in a document?*
- One common usage for NLP is extracting *sentiment*: emotional tone behind a body of text
- Widely used in social media monitoring, customer feedback analysis, and market research
  - Can be rule-based using pre-defined rules and lexicons
  - Or derived from machine learning or deep learning models

Sentiment Analysis

- We will use `TextBlob`: sentiment classifier that is trained on labeled dataset of text examples
- `spacytextblob`: wrapper that integrates `TextBlob` with `spacy`
- In command line:

```
$ pip install spacytextblob
```

```
from spacytextblob.spacytextblob import SpacyTextBlob
nlp = spacy.load('en_core_web_sm')
nlp.add_pipe('spacytextblob')
```

```
<spacytextblob.spacytextblob.SpacyTextBlob at 0x12c3ad010>
```

Sentiment Analysis

`spacytextblob` measures 2 dimensions of sentiment:

- **Polarity**: positive or negative (-1 to 1)
- **Subjectivity**: objective vs. subjective (0 to 1)

Sentiment Analysis: polarity

- Polarity: higher number = more positive

```
doc = nlp("The unemployment rate is good right now. I wouldn't want
 ↪  to stay at this level of inflation, though.")
```

- Sentiment can be estimated at the document level

```
doc._.blob.polarity
```

```
0.4928571428571428
```

Sentiment Analysis: polarity

- Polarity: higher number = more positive

```
doc = nlp("The unemployment rate is good right now. I wouldn't want
 ↪  to stay at this level of inflation, though.")
```

- Sentiment can be estimated at the document level

```
doc._.blob.polarity
```

```
0.4928571428571428
```

- Or at the sentence level (do you agree with second sentence polarity?)

```
print([sent._.blob.polarity for sent in doc.sents])
```

```
[0.4928571428571428, 0.0]
```

Sentiment Analysis: polarity

- Or at the token-level

```
print([token._.blob.polarity for token in doc])
```

```
[0.0, 0.0, 0.0, 0.0, 0.7, 0.2857142857142857, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
```

Discussion q: do you agree that second sentence is neutral?

Sentiment Analysis: subjectivity

- Subjectivity: higher number is more subjective

```
doc = nlp("The unemployment rate is good right now. I wouldn't want
 ↪  to stay at this level of inflation, though.")
doc._.blob.subjectivity
```

```
0.5678571428571428
```

Sentiment Analysis: subjectivity

- Subjectivity: higher number is more subjective

```
doc = nlp("The unemployment rate is good right now. I wouldn't want
 ↪  to stay at this level of inflation, though.")
doc._.blob.subjectivity
```

```
0.5678571428571428
```

```
doc = nlp("The unemployment rate is currently low. However, inflation
 ↪  remains higher than last summer.")
doc._.blob.subjectivity
```

```
0.2888888888888889
```

Sentiment Analysis: subjectivity

- Subjectivity: higher number is more subjective

```
doc = nlp("The unemployment rate is good right now. Inflation is
 ↪  really terrible though.")
doc._.blob.subjectivity
```

```
0.7119047619047619
```

Sentiment Analysis: subjectivity

- Subjectivity: higher number is more subjective

```
doc = nlp("The unemployment rate is good right now. Inflation is
 ↪  really terrible though.")
doc._.blob.subjectivity
```

```
0.7119047619047619
```

```
doc = nlp("The unemployment rate is currently low. However, inflation
 ↪  remains higher than last summer.")
doc._.blob.subjectivity
```

```
0.2888888888888889
```

Sentiment Analysis: Under the Hood

- Where do these sentiment values come from?
- `spacytextblob` builds on a lexico (aka library) called `pattern`
  - **Lexicon**: a collection of words and their associated polarity/subjectivity scores
  - `pattern` lexicon is an open-source was developed at the University of Antwerp
  - Developed through manual labeling of words, mostly of *adjectives*
  - This is why "I wouldn't want to stay at this level of inflation, though." gets a polarity of zero
- Depending on your use, focusing on just the adjectives may or may not be sufficient
  - More appropriate for highly emotive text: social media, debates
  - Less appropriate for formal text: academic research, diplomatic communications

Application: Presidential Debates

```
with open("files/clinton_2016.txt", 'r') as file:
    clinton_2016_text = file.read()

print(clinton_2016_text)
```

```
How are you, Donald? [applause]

Well, thank you, Lester, and thanks to Hofstra for hosting us.

The central question in this election is really what kind of country we wa

I want us to invest in you. I want us to invest in your future. That means

I also want to see more companies do profit-sharing. If you help create th

And I want us to do more to support people who are struggling to balance fa
```

Application: Presidential Debates

- We can look at Hillary Clinton's overall sentiment during the 2016 debate

```
doc_clinton = nlp(clinton_2016_text)
print(f"Polarity: {doc_clinton._.blob.polarity:.2f}")
print(f"Subjectivity: {doc_clinton._.blob.subjectivity:.2f}")
```

```
Polarity: 0.17
Subjectivity: 0.48
```

Application: Presidential Debates

- The units of the polarity and subjectivity scores difficult to interpret on their own, so we can compare it to Trump's 2016 sentiment

```
with open("files/trump_2016.txt", 'r') as file:
    trump_2016_text = file.read()

doc_trump = nlp(trump_2016_text)
print(f"Polarity: {doc_trump._.blob.polarity:.2f}")
print(f"Subjectivity: {doc_trump._.blob.subjectivity:.2f}")
```

```
Polarity: 0.12
Subjectivity: 0.53
```

- While they both have positive sentiment overall, Trump's is slightly lower than Clinton's
- Trump is also slightly more subjective than Clinton

Application: Presidential Debates

```
import pandas as pd
import altair as alt
alt.renderers.enable("png", ppi=300) #high ppi increases resolution
```

```
RendererRegistry.enable('png')
```

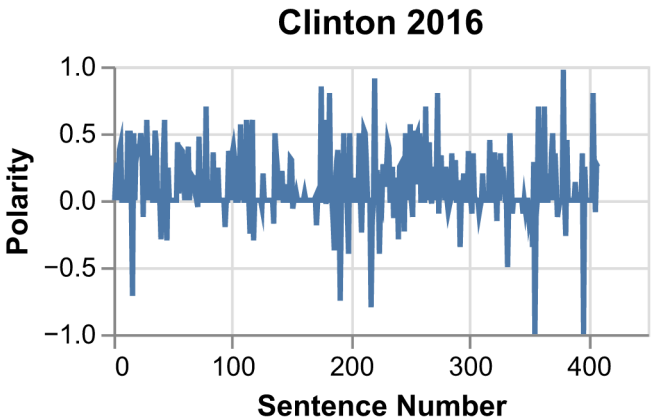
```
clinton_sentence_polarities = []
for i, sentence in enumerate(doc_clinton.sents):
    polarity = sentence._.blob.polarity
    clinton_sentence_polarities.append({"n": i + 1,
 ↪  "clinton_polarity": polarity})
df_clinton = pd.DataFrame(clinton_sentence_polarities)
```

- `clinton_sentence_polarities` is initially an empty lists
- We then iterate through `doc_clinton.sents` and fill in the polarity values

Application: Presidential Debates

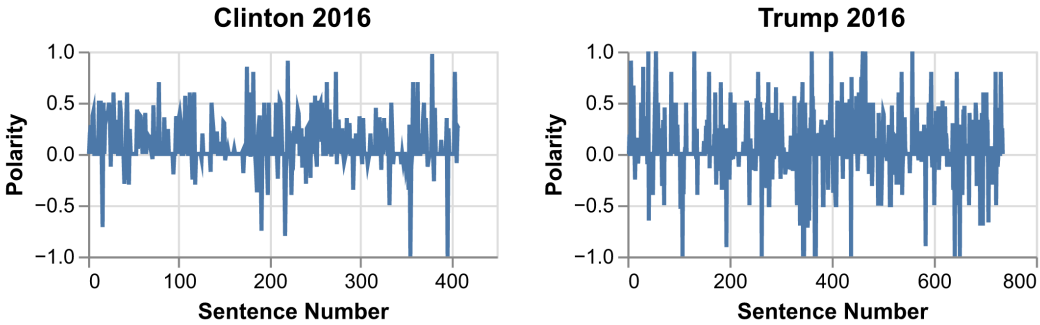
```
chart1 = alt.Chart(df_clinton).mark_line().encode(
    x=alt.X('n', title='Sentence Number'),
    y=alt.Y('clinton_polarity', title='Polarity')
).properties(
    title = "Clinton",
    width=100,
    height=100
)
chart1
```

Application: Presidential Debates



Application: Presidential Debates

- Repeating it for Trump and plotting them side by side:



- We can see that while Trump's average polarity was lower than Clinton's, there was also greater *variance*
- He also had over twice as many sentences than Clinton

Do-pair-share

```
$ pip install --upgrade spacy spacytextblob numpy pandas h5py
$ python -m spacy download en_core_web_sm
```

```
import spacy
nlp = spacy.load("en_core_web_sm")
from spacytextblob.spacytextblob import SpacyTextBlob
nlp = spacy.load('en_core_web_sm')
nlp.add_pipe('spacytextblob')
alt.renderers.enable("png")
```

1. Load in the Trump 2020 debate text (`files/`) and conduct polarity sentiment analysis on it
2. Plot time series for Trump 2020. Compare it to Trump 2016 plot in the slides.

Sentiment Analysis: Summary

- One model for sentiment analysis: `spacytextblob`
- Trained on a lexicon of tagged words
- Sentiment analysis can be conducted at the word, sentence, or document level, for two dimensions: polarity and subjectivity

Use Case 2: Named Entity Detection

Named Entity Detection: Roadmap

- Introduce named entity detection
- Discuss named entity types and how the model was trained
- Application: presidential debate
- Exercise (if we have time): try it out with your own name

Named Entity Detection

- *Question: who/what is being discussed?*
- Language models are also trained to detect **named entities** – places, people, organizations

```
doc = nlp("Jerome Powell said the Federal Reserve Board wouldn't raise the
 ↪  federal funds rate. This signals the U.S. is nearing a long-awaited
 ↪  shift toward cutting interest rates.")
print(doc.ents)
```

```
(Jerome Powell, the Federal Reserve Board, U.S.)
```

Named Entity Detection

- *Question: who/what is being discussed?*
- Language models are also trained to detect **named entities** – places, people, organizations

```
doc = nlp("Jerome Powell said the Federal Reserve Board wouldn't raise the
 ↪  federal funds rate. This signals the U.S. is nearing a long-awaited
 ↪  shift toward cutting interest rates.")
print(doc.ents)
```

```
(Jerome Powell, the Federal Reserve Board, U.S.)
```

- Furthermore, it assigns an entity *type*

```
entities = [(entity.label_, entity.text) for entity in doc.ents]
print(entities[0])
print(entities[1])
```

```
('PERSON', 'Jerome Powell')
('ORG', 'the Federal Reserve Board')
```

Named Entity Detection

Entity Type	Description
‘PERSON‘	People, including fictional.
‘NORP‘	Nationalities or religious or political groups.
‘FAC‘	Buildings, airports, highways, bridges, etc.
‘ORG‘	Companies, agencies, institutions, etc.
‘GPE‘	Countries, cities, states.
‘LOC‘	Non-GPE locations, mountain ranges, bodies of water.
‘PRODUCT‘	Objects, vehicles, foods, etc. (Not services.)
‘EVENT‘	Named hurricanes, battles, wars, sports events, etc.

Named Entity Detection

Entity Type	Description
‘WORK_OF_ART‘	Titles of books, songs, etc.
‘LAW‘	Named documents made into laws.
‘LANGUAGE‘	Any named language.
‘DATE‘	Absolute or relative dates or periods.
‘TIME‘	Times smaller than a day.
‘PERCENT‘	Percentage, including ”%“.
‘MONEY‘	Monetary values, including unit.
‘QUANTITY‘	Measurements, as of weight or distance.
‘ORDINAL‘	“first”, “second”, etc.
‘CARDINAL‘	Numerals that do not fall under another type.

Named Entity Detection: Under the Hood

- How did the language model "know" that these were known entities and what type they were?
- **Statistical learning**: `en_core_web_sm` language model was trained on large datasets with annotated examples of entities
- **Contextual analysis**: model also uses contextual clues to determine if a word/phrase is a known entity

Named Entity Detection

```
apple_text = "Apple announced in March that it was releasing a new iPhone.
↪  Apples are a delicious snack."

apple_doc = nlp(apple_text)
entities = [(entity.label_, entity.text) for entity in apple_doc.ents]
print(entities)
```

Named Entity Detection

```
apple_text = "Apple announced in March that it was releasing a new iPhone.
 ↪  Apples are a delicious snack."

apple_doc = nlp(apple_text)
entities = [(entity.label_, entity.text) for entity in apple_doc.ents]
print(entities)
```

```
[('ORG', 'Apple'), ('DATE', 'March')]
```

- Known entity detection can differentiate between “Apple” the company and “Apples” the fruit (XXX code is not working here XXX)
- But it’s also imperfect - did not detect “iPhone” as a known entity

Exercise

- Write a sentence with *your name*, and use named entity detection to try to pick it up.
- Was it able to detect your name as a named entity?
- What does that tell us about the uses and limitations of this language model?

Exercise Examples

```
name_doc = nlp("Peter is teaching a class.")
print(name_doc.ents)
entities = [(entity.label_, entity.text) for entity in name_doc.ents]
print(entities[0])
```

```
(Peter,)
('PERSON', 'Peter')
```

Exercise Examples

```
name_doc = nlp("Peter is teaching a class.")
print(name_doc.ents)
entities = [(entity.label_, entity.text) for entity in name_doc.ents]
print(entities[0])
```

```
(Peter,)
('PERSON', 'Peter')
```

```
name_doc = nlp("Maggie is teaching a class.")
print(name_doc.ents)
entities = [(entity.label_, entity.text) for entity in name_doc.ents]
print(entities[0])
```

```
(Maggie,)
('PERSON', 'Maggie')
```

Exercise Examples

```
name_doc = nlp("Xiao is teaching a class.")
print(name_doc.ents)
```

```
(Xiao,)
```

Exercise Examples

```
name_doc = nlp("Xiao is teaching a class.")
print(name_doc.ents)
```

```
(Xiao,)
```

- The quality and accuracy of a model depends on what it was trained on
- `en_core_web_sm` was trained on English language news articles, conversational telephone speech, blogs, broadcasts, talk shows
- It will tend to be more correct for names and entities that are disproportionately represented in these types of media

Application: Presidential Debate

```
entities_trump = [ent.text for ent in doc_trump.ents]

print("Top 10 Named Entities Discussed by Trump:")
print(pd.Series(entities_trump).value_counts().head(10))
```

Application: Presidential Debate

```
Top 10 Named Entities Discussed by Trump:
Clinton    22
ISIS       14
Lester     10
NATO       10
one         9
China       9
Chicago     9
Iran        8
Obama       8
Hillary     8
Name: count, dtype: int64
```

Application: Presidential Debate

Repeating this for Clinton:

```
Top 10 Named Entities Discussed by Clinton:
Donald                27
one                   11
American              10
Iran                   9
ISIS                   8
first                  5
Iraq                   5
the United States      5
two                    5
America                5
Name: count, dtype: int64
```

Named Entity Detection: Summary

- `spaCy` can detect named entities (people, places, organizations, etc.) using `.ents`
- Based on training model on text with labeled entities
- Less likely to be correct for names and entities that are less-represented in English-language media

Use Case 3: Dependency Parsing

Dependency Parsing: Roadmap

- Introduce dependency parsing terminology
- Visualize relationships
- Application: presidential debates

Dependency Parsing Terminology

- *Question: what is being said about a particular topic?*
- **Dependency parsing**: analyze grammatical structure of sentence and establish relationship between **head** words and their **dependents** or **children**
- **Head** or **parent**: token (usually a verb or noun) that other words are connected to
- **Children**: words that modifies its head (e.g., subjects, objects, adjectives, adverbs)
- **Ancestor**: any token *higher up* in the dependency tree – i.e., parent, parent's parent

Illustrating Dependencies

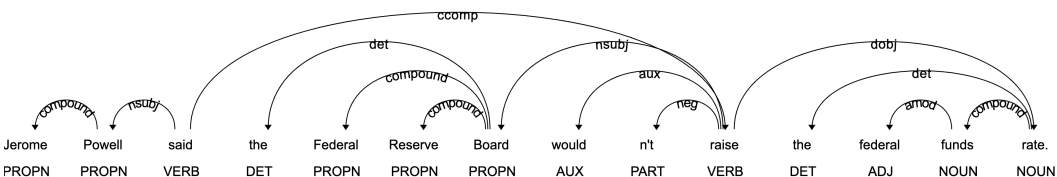
- spacy has a built-in functionality to visualize these relationships

```
from spacy import displacy
doc = nlp("Jerome Powell said the Federal Reserve Board wouldn't
 ↪  raise the federal funds rate.")
displacy.render(doc, style="dep", jupyter=True)
```

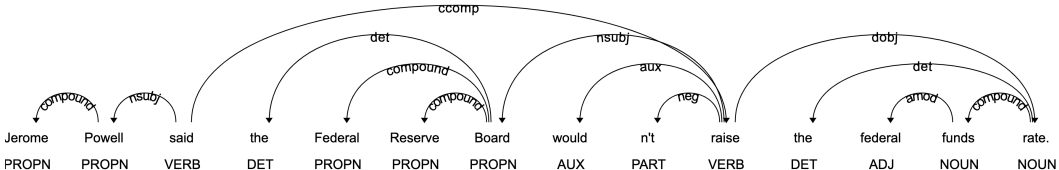
Illustrating Dependencies

- spacy has a built-in functionality to visualize these relationships

```
from spacy import displacy
doc = nlp("Jerome Powell said the Federal Reserve Board wouldn't
 ↪  raise the federal funds rate.")
displacy.render(doc, style="dep", jupyter=True)
```



Dependency Relationships

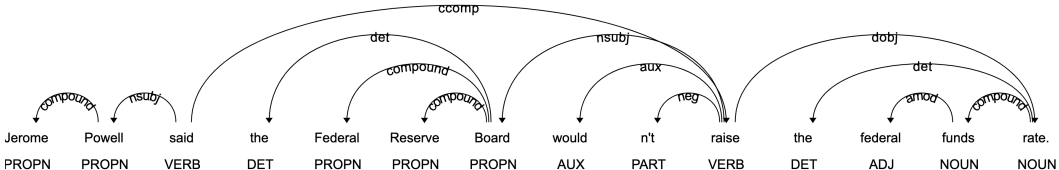


- Use `.ancestors` to find a token's ancestors

```
for token in doc:
    if token.text == "rate":
        for ancestor in token.ancestors:
            print(f"{ancestor.text}")
```

```
raise
said
```

Dependency Relationships



- Use `.children` to find a token's children

```
for token in doc:
    if token.text == "rate":
        for child in token.children:
            print(f"{child.text}")
```

```
the
funds
```

Application: Presidential Debates

- We can apply this to the presidential debates and combine with sentiment analysis
- What is the sentiment when each candidate mentions the economy?
- Steps
  - Identify all tokens in each candidate's transcript that are children or ancestors of "economy"
  - Calculate polarity using `spacytextblob`
  - Take mean of all non-zero polarity values

Application: Presidential Debates

```
tokens_clinton = []
for token in doc_clinton:
    if token.text == "economy":
        for child in token.children:
            tokens_clinton.append((child))
        for ancestor in token.ancestors:
            tokens_clinton.append((ancestor))
```

Application: Presidential Debates

```
[print(token) for token in tokens_clinton]
```

```
an
works
,
those
build
have
the
fairer
make
have
the
grow
is
the
growing
```

Application: Presidential Debates

```
polarities = [token._.blob.polarity for token in tokens_clinton if
 ↪  token._.blob.polarity != 0]
mean_polarity_clinton = sum(polarities) / len(polarities) if
 ↪  polarities else 0
print(f"Clinton's sentiment on the economy:
 ↪  {mean_polarity_clinton:.2f}")
```

```
Clinton's sentiment on the economy: 0.60
```

Application: Presidential Debates

```
tokens_trump = []
for token in doc_trump:
    if token.text == "economy":
        for child in token.children:
            tokens_trump.append((child))
        for ancestor in token.ancestors:
            tokens_trump.append((ancestor))
```

Application: Presidential Debates

```
[print(token) for token in tokens_trump]
```

```
our
is
happened
owe
an
of
revival
have
the
about
talked
```

```
[None, None, None, None, None, None, None, None, None, None, None]
```

Application: Presidential Debates

```
polarities = [token._.blob.polarity for token in tokens_trump if
 ↪  token._.blob.polarity != 0]
mean_polarity_trump = sum(polarities) / len(polarities) if polarities
 ↪  else 0
print(f"Trump's sentiment on the economy: {mean_polarity_trump:.2f}")
```

```
Trump's sentiment on the economy: 0.00
```

Dependency Parsing: Summary

- Dependency parsing analyzes structure of words within sentences to establish "parent-child" relationships
- We can use this to see what words are used in reference to key topics
- Could refine this analysis by combining it with:
  - **Parts of speech** tagging (e.g., keep only adjectives or adverbs)
  - **Lemmatization** (i.e., convert "growing" into "grow")

NLP: Summary

- NLP uses trained language models to parse through text, allowing us to use it as data
- We discussed 3 use cases:
  - Sentiment analysis
  - Named entity detection
  - Dependency parsing
- We have scratched the surface of NLP and the `spacy` package – much more can be done with it!