Natural Language Processing

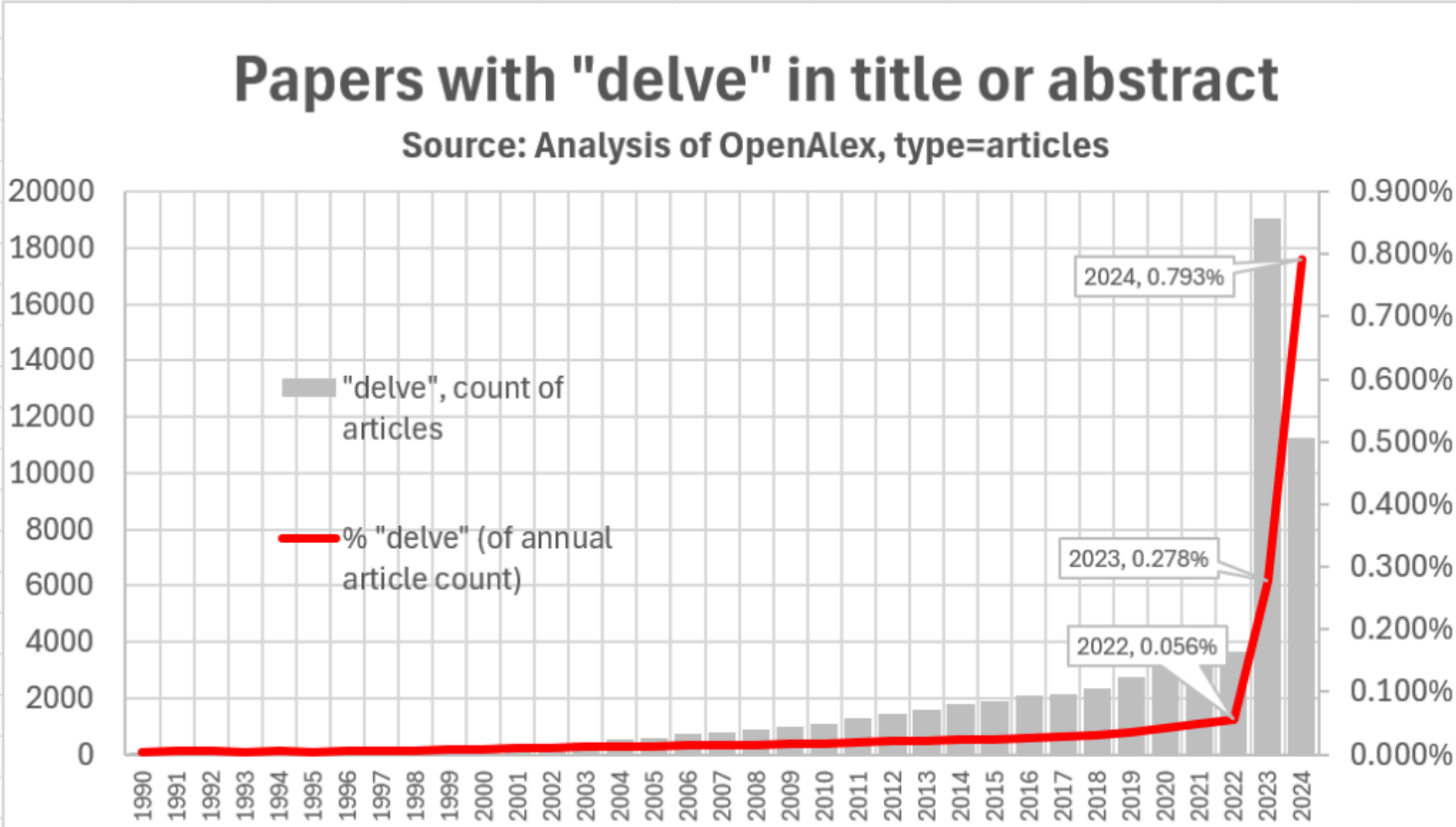
Do Now: Songs w/ Pandas

bigd103.link/pandas-lyrics

Natural Language Processing (NLP) is an interdisciplinary subfield of computer science and artificial intelligence. It is primarily concerned with enabling computers to process data encoded in natural language. NLP is closely related to information retrieval, knowledge representation, and computational linguistics—a subfield of linguistics.

Applications of NLP in Real-World Scenarios

- Language translation (e.g., Google Translate).
- Filtering internet hate speech, fake news, and spam.
- Content recommendation systems.
- *A component of* text-completion suggestions (e.g., autocomplete).
- *A component of* language processing and auto-captioning.
- *A component of* large language models (LLMs), such as ChatGPT.



Analysis using OpenAlex by Philip Shapira, March 31, 2024.

Popular NLP Libraries

NLTK:

- Natural Language Toolkit, an open-source Python library by Steven Bird, Edward Loper, and Ewan Klein.
- Very comprehensive, widely used, but somewhat older and can be confusing to learn initially.

spaCy:

- Implemented in Cython and developed by Matthew Honnibal.
- Faster, easy to use, and more modern than NLTK but somewhat less comprehensive out-of-the-box.

TextBlob:

- Built on top of NLTK and Pattern, providing a simple API for common NLP tasks.
- Great for beginners and rapid prototyping.

NLP Terminology

Document:

- A piece of text, such as a sentence, a paragraph, a song, or a book.

Corpus:

- A collection of documents, usually in the same language and/or domain.

Text Preprocessing

Tokenization

Word Tokenization

Splitting text into individual words (or “tokens”).

Sentence Tokenization

Splitting text into individual sentences.

```
import re

s = "I love NLP!"
tokens = re.split(r"\s", s)
print(tokens)

s = "I love NLP! It's fascinating."
sentences = re.split(r"[.!?]", s)
print(sentences)
```

Lowercasing and Punctuation Removal

Lowercasing

Converting all characters in the text to lowercase.

```
-NLP is fascinating
+nlp is fascinating
```

Punctuation Removal

Removing punctuation marks from the text.

```
-Hello, world!
+Hello world
```

Stopwords Removal

Common words that do not carry much meaning and can be removed.

- Examples: "the", "is", "and", "a".

```
-This is a sample sentence, showing off the stop words filtr+This sample sentence, showing stop words filtration.
```

Stemming and Lemmatization

Stemming

Reduces words to a simpler base or “root” form.

- Example: "running", "runner", "ran" → "run"

```
-the bats saw the cats with stripes hanging upside down by t+the bat saw the cat with stripes hang upsid down by their fee
```

Lemmatization

Reduces words to their dictionary form (lemma) using vocabulary and morphological analysis.

- Example: "better" → "good" (using a dictionary that knows “better” is a form of “good”)

```
-the bats saw the cats with stripes hanging upside down by t+the bat see the cat with stripe hang upside down by their foo
```

Step 1: To Lower

Python has a built-in method to convert strings to lowercase.

```
# Original sentence
sentence = "The men will be running Wednesday if the weather gets better."
sentence = sentence.lower()
```

```
-The men will be running Wednesday if the weather gets bette+the men will be running wednesday if the weather gets better.
```

Step 2: Punctuation Removal

SpaCy can be used to remove punctuation. We load the English model and iterate through the tokens, removing punctuation marks.

```
import spacy

nlp = spacy.load("en_core_web_lg")  # Load spaCy model

doc = nlp(sentence)  # Initialize as a spaCy object (list of tokens)
words = []
for token in doc:
    if not token.is_punct:
        words.append(token.text)
sentence = ' '.join(words)
```

```
-the men will be running wednesday if the weather gets bette+the men will be running wednesday if the weather gets better
```

Step 3: Stopword Removal

Again, using spaCy to remove stopwords. We iterate through the tokens and filter out common words like "the" or "is."

```
doc = nlp(sentence)
words = []
for token in doc:
    if not token.is_stop:
        words.append(token.text)
sentence = ' '.join(words)
```

```
-the men will be running wednesday if the weather gets bette+men running wednesday weather gets better
```

Step 4: Stemming

NLTK provides a stemmer that can handle stemming:

```
from nltk.stem import PorterStemmer

doc = sentence.split()
stemmer = PorterStemmer()
words = []
for token in doc:
    words.append(stemmer.stem(token))
sentence = ' '.join(words)
```

```
-men running wednesday weather gets better
+men run wednesday weather get better
```

Step 5: Lemmatization

Lemmatization can also be done using spaCy. We iterate through the tokens and replace them with their lemma.

```
doc = nlp(sentence)
words = []
for token in doc:
    words.append(token.lemma_)
sentence = ' '.join(words)
```

```
-men run wednesday weather get better
+man run wednesday weather get well
```

*(Often, you choose to do either stemming **or** lemmatization, not both.)*

All Together Now

```python
import spacy
nlp = spacy.load("en_core_web_sm")  # Load small English model

sentence = "The men will be running Wednesday if the weather gets better."

preprocessed_tokens = []
doc = nlp(sentence)
for token in doc:
    # Filter out punctuation, stopwords, and spaces
    if not token.is_punct and not token.is_stop and not token.is_space:
        text = token.lemma_   # Lemmatized
        text = text.lower()   # Lowercase
        preprocessed_tokens.append(text)

print(sentence)
print(" ".join(preprocessed_tokens))
```

```
-The men will be running Wednesday if the weather gets bette+man run wednesday weather get well
```

A More Practical Preprocessor

```
import re

# Hardcoded stop words
STOP_WORDS = {"a", "an", "and", "are", "as", "at", "be", "by", "for", "from", "has", "he", "in", "is", "it", "its", "of", "
```

```
def preprocess(text):
    """Convert text to a list of lowercase words, removing stop words"""
    # Split on punctuation and whitespace
    tokens = re.split(r"[,\.\!\?\s]+", text)

    # Keep only non-empty lower-case tokens that aren't stop words
    processed_tokens = []
    for token in tokens:
        token = token.lower() # Lowercase the token
        if token and token not in STOP_WORDS:
            processed_tokens.append(token)

    return processed_tokens

# Test it
test_text = "Hello! How are you doing today?"
print(preprocess(test_text))  # Should print: ['hello', 'how', 'doing', 'today']
```

```
['hello', 'how', 'doing', 'today']
```

TF-IDF (Term Frequency–Inverse Document Frequency)

Reflects how important a word is to a document within a corpus.

Intuition

The word **"whale"** appears far more often in *Moby Dick* than in most books, so it's quite "unique" there. A common word like **"man"** might appear often, but that doesn't reveal much about the book's main topic. TF-IDF helps highlight words that are uniquely important to a document.

the most venerable of the leviathans, being the one first regularly hunted by man. It yields the article commonly known as whalebone or baleen; and the oil specially known as "whale oil," an inferior article in commerce. Among the fishermen, he is indiscriminately designated by all the following titles: The Whale; the Greenland Whale; the Black Whale; the Great Whale; the True Whale; the Right Whale. There is a deal of obscurity concerning the identity of the species thus multitudinously baptised. What then is the whale, which I include in the second species of my Folios? It is the Great Mysticetus of the English naturalists; the Greenland Whale of the English whalemen; the Baliene Ordinaire of the French whalemen; the Growlands Walfish of the Swedes. It is the whale which for more than two centuries past has been hunted by the Dutch and English in the Arctic seas; it is the whale which the American fishermen have long pursued in the Indian ocean, on the Brazil Banks, on the Nor' West Coast, and various other parts of the world, designated by them Right Whale Cruising Grounds.

the most venerable of the leviathans, being the one first regularly hunted by man. It yields the article commonly known as whalebone or baleen; and the oil specially known as "whale oil," an inferior article in commerce. Among the fishermen, he is indiscriminately designated by all the following titles: The Whale; the Greenland Whale; the Black Whale; the Great Whale; the True Whale; the Right Whale. There is a deal of obscurity concerning the identity of the species thus multitudinously baptised. What then is the whale, which I include in the second species of my Folios? It is the Great Mysticetus of the English naturalists; the Greenland Whale of the English whalemen; the Baliene Ordinaire of the French whalemen; the Growlands Walfish of the Swedes. It is the whale which for more than two centuries past has been hunted by the Dutch and English in the Arctic seas; it is the whale which the American fishermen have long pursued in the Indian ocean, on the Brazil Banks, on the Nor' West Coast, and various other parts of the world, designated by them Right Whale Cruising Grounds.

TF-IDF Calculation

1. **Term Frequency (TF)**: How often a word appears in a specific document (normalized by total words in that doc):

$$\mathrm{tf}(t, d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}$$

   where $f_{t,d}$ is the count of term t in document d, and $\sum_{t' \in d} f_{t',d}$ is the total word count in d.

2. **Inverse Document Frequency (IDF)**: Measures how rare a word is across all documents in the corpus:

$$\mathrm{idf}(t, D) = \log\left(\frac{N}{|\{d : d \in D \text{ and } t \in d\}|}\right)$$

   where N is the total number of documents.

3. **TF-IDF**: The product of the two:

$$\mathrm{tfidf}(t, d, D) = \mathrm{tf}(t, d) \cdot \mathrm{idf}(t, D)$$

Step 1: Define Your Corpus and Pick a Target Document

```
corpus = [
    "John likes to watch movies.",
    "Mary likes movies too.",
    "John also likes to watch football games."  # <-- Target Document
]
tgt_doc = corpus[2]
```

A "corpus" is just a collection of documents. For example:

- “Moby Dick” = one document
- “All English Books” = one corpus

Step 2: Term Frequency (TF)

Calculate the frequency of each term in the *target* document.

```
def calculate_term_frequency(text):
    term_freq = {}
    for term in preprocess(text):
        if term in term_freq:
            term_freq[term] += 1
        else:
            term_freq[term] = 0
    return term_freq
```