Natural Language Processing

DEFINITION *Natural language processing* is an area of research in computer science

and artificial intelligence (AI) concerned with processing natural languages

such as English or Mandarin. This processing generally involves

translating natural language into data (numbers) that a computer can use to

learn about the world. And this understanding of the world is sometimes used

to generate natural language text that reflects that understanding.

DEFINITION A natural language processing system is often referred to as a

*pipeline* because it usually involves several stages of processing where natural

language flows in one end and the processed output flows out the other.

Corpus

Lexicon

Regular expression

A regular expression search function will search through the corpus, returning all texts that match the pattern. The corpus can be a single document or a collection.

All the possible vectors a machine might create this way is called a *vector space*. And this model of documents and statements and words is called a *vector space model*.

Stemming:

Stemming is the process of reducing words to their root or base form. The root form is not necessarily a word itself but rather the core part of the word that has meaning.

For example, the words "running", "runs", and "runner" all have the same root word "run". By stemming these words, they would all be reduced to the root form "run".

n-grams:

Extacting pairs, triplets, quadruplets and even quintuplets of token is called as n-grams.

Pairs of words are 2-grams (bigrams), triplets are 3-grams

(trigrams), quadruplets are 4-grams, and so on. Using *n*-grams enables your machine

to know about “ice cream” as well as the “ice” and “cream” that comprise it. Another

2-gram that you’d like to keep together is “Mr. Smith.” Your tokens and your vector

representation of a document will have a place for “Mr. Smith” along with “Mr.” and

“Smith,” too.

Tokenizer:

In NLP, *tokenization* is a particular kind of document segmentation. *Segmentation*

breaks up text into smaller chunks or segments, with more focused information content.

Segmentation can include breaking a document into paragraphs, paragraphs

into sentences, sentences into phrases, or phrases into tokens (usually words) and

punctuation. Segmenting text into *tokens*, which is called tokenization.

A tokenizer breaks unstructured data, natural language text,

into chunks of information that can be counted as discrete elements. These counts of

token occurrences in a document can be used directly as a vector representing that

document.

One Hot vectors

Creating a numerical vector representation for each word are called *one-hot vectors.* It represents the position of the active element corresponds to the word's index in the vocabulary. Captures the **presence or absence** of a word in a document, offering no information about its frequency or importance. High dimensionality due to large vocabulary size, potentially leading to data sparsity issues and inefficient computation.

Bag of Word:

Each word's **occurrence frequency** within a document is represented by an integer value in a vector. The vector size is equal to the vocabulary size. Captures the **importance** of each word based on its frequency, but ignores the order of words within the document. Offers some insight into word importance, suitable for tasks like document classification. Ignores word order and relationships between words, may lead to loss of contextual information. Suitable for capturing word importance based on frequency, often used in conjunction with other techniques like TF-IDF (Term Frequency-Inverse Document Frequency) to adjust for document length and word rarity across the corpus

Corpus/Corpora

Corpus refers to a large and structured set of text documents or spoken language recordings that are used for linguistic analysis, language modeling, and machine learning purposes. Corpora (plural of

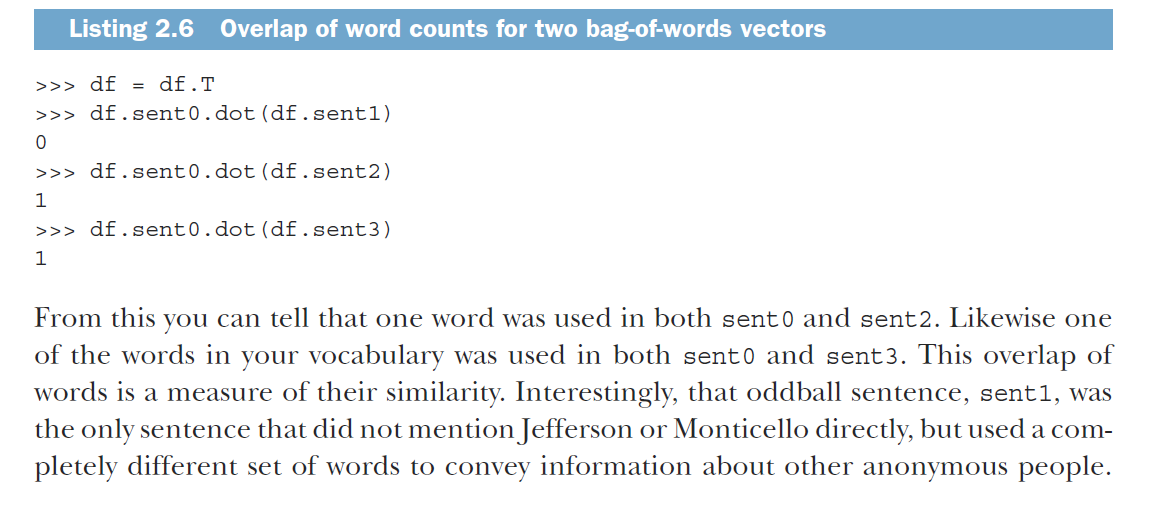
corpus) serve as the primary data source for training and testing NLP models.

***Measuring bag-of-words overlap***

If we can measure the bag of words overlap for two vectors, we can get a good estimate

of how similar they are in the words they use. And this is a good estimate of how similar

they are in meaning.



RegexpTokenizer

>>> from nltk.tokenize import RegexpTokenizer

>>> tokenizer = RegexpTokenizer(r'\w+|$[0-9.]+|\S+')

>>> tokenizer.tokenize(sentence)

['Thomas',

'Jefferson',

'began',

'building',

'Monticello',

'at',

'the',

'age',

'of',

'26',

'.']

Treebank Word Tokenizer

>>> from nltk.tokenize import TreebankWordTokenizer

>>> sentence = """Monticello wasn't designated as UNESCO World Heritage\

... Site until 1987."""

>>> tokenizer = TreebankWordTokenizer()

>>> tokenizer.tokenize(sentence)

['Monticello',

'was',

"n't",

'designated',

'as',

'UNESCO',

'World',

'Heritage',

'Site',

'until',

'1987',

'.']

CONTRACTIONS

You might wonder why you would split the contraction wasn’t into was and n’t. For

some applications, like grammar-based NLP models that use syntax trees, it’s important

to separate the words was and not to allow the syntax tree parser to have a consistent,

predictable set of tokens with known grammar rules as its input. There are a

variety of standard and nonstandard ways to contract words.

***Extending your vocabulary with n-grams***

An *n*-gram is a sequence containing up to *n* elements that have been extracted from a

sequence of those elements, usually a string.

For example, the meaning-inverting word “not” will remain

attached to its neighboring words, where it belongs. Without *n*-gram tokenization, it

would be free floating. Its

STOP WORDS

Stop words are common words in any language that occur with a high frequency but

carry much less substantive information about the meaning of a phrase. Examples of

some common stop words include8

 a, an

 the, this

 and, or

 of, on

Historically, stop words have been excluded from NLP pipelines in order to reduce

the computational effort to extract information from a text. Even though the words

themselves carry little information, the stop words can provide important relational

information as part of an *n*-gram.

***Normalizing your vocabulary***

So you’ve seen how important vocabulary size is to the performance of an NLP pipeline.

Another vocabulary reduction technique is to normalize your vocabulary so that

tokens that mean similar things are combined into a single, normalized form. Doing

so reduces the number of tokens you need to retain in your vocabulary and also

improves the association of meaning across those different “spellings” of a token or *n*gram

in your corpus. And as we mentioned before, reducing your vocabulary can

reduce the likelihood of overfitting.

CASE FOLDING

Words can become case “denormalized” when they are capitalized because of their presence at the beginning of a sentence, Undoing this denormalization is called *case normalization*, or more commonly, *case folding*. Normalizing word and character capitalization is one way to reduce your vocabulary size and

generalize your NLP pipeline.

STEMMING

Another common vocabulary normalization technique is to eliminate the small meaning

differences of pluralization or possessive endings of words, or even various verb

forms. This normalization, identifying a common stem among various forms of a

word, is called stemming. For example, the words housing and houses share the same

stem, house. Stemming removes suffixes from words in an attempt to combine words

with similar meanings together under their common stem. A stem isn’t required to be

a properly spelled word, but merely a token, or label, representing several possible

spellings of a word.

LEMMATIZATION

If you have access to information about connections between the meanings of various

words, you might be able to associate several words together even if their spelling is

quite different. This more extensive normalization down to the semantic root of a

word—its lemma—is called lemmatization.

USE CASES

When should you use a lemmatizer or a stemmer? Stemmers are generally faster to

compute and require less-complex code and datasets. But stemmers will make more

errors and stem a far greater number of words, reducing the information content or

meaning of your text much more than a lemmatizer would. Both stemmers and lemmatizers

will reduce your vocabulary size and increase the ambiguity of the text. But

lemmatizers do a better job retaining as much of the information content as possible

based on how the word was used within the text and its intended meaning. Therefore,

some NLP packages, such as spaCy, don’t provide stemming functions and only offer

lemmatization methods.

***Sentiment***

Whether you use raw single-word tokens, *n*-grams, stems, or lemmas in your NLP pipeline,

each of those tokens contains some information. An important part of this information

is the word’s sentiment—the overall feeling or emotion that the word invokes.

This *sentiment analysis*—measuring the sentiment of phrases or chunks of text—is a

common application of NLP. In many companies it’s the main thing an NLP engineer

is asked to do.

Say you want your NLP pipeline and sentiment analysis algorithm to

output a single floating point number between -1 and +1. Your algorithm would output

+1 for text with positive sentiment like, “Absolutely perfect! Love it! :-) :-) :-).” And

your algorithm should output -1 for text with negative sentiment like, “Horrible! Completely

useless. :(.” Your NLP pipeline could use values near 0, like say +0.1, for a statement

like, “It was OK. Some good and some bad things.”

There are two approaches to sentiment analysis:

 A rule-based algorithm composed by a human

 A *machine learning* model learned from data by a machine

***VADER—A rule-based sentiment analyzer***

Hutto and Gilbert at GA Tech came up with one of the first successful rule-based sentiment

analysis algorithms. They called their algorithm VADER, for **V**alence **A**ware **D**ictionary

for s**E**ntiment **R**easoning.20 Many NLP packages implement some form of this

algorithm.

*Bags of words*—Vectors of word counts or frequencies

*Bags of n-grams*—Counts of word pairs (bigrams), triplets (trigrams), and so on

*TF-IDF vectors*—Word scores that better represent their importance

TF-IDF stands for ***t****erm* ***f****requency times* ***i****nverse* ***d****ocument* ***f****requency*.

It is used to evaluate the importance of word in a document relative to a collection of document.

Term frequency measures how often a word appears in a document which is evaluated by number of term (word) appear in a document by the total number of terms in a document.

Inverse document frequency measures how important a term is across the entire corpus. It is calculated by dividing the total number of documents by the number of documents containing the term, and taking the logarithm of that quotient.

TF-IDF is the product of TF and IDF

It gives higher weight to terms that are frequent in a document but rare in the corpus, thus emphasizing their importance in that particular document.

Term frequencies are the counts of each word in a document, which you

learned about in previous chapters. Inverse document frequency means that

you’ll divide each of those word counts by the number of documents in which

the word occurs.