PRZETWARZANIE JEZYKA NATURALNEGO W PYTHONIE

Wprowadzenie

PLAN NA DZISIAJ

- Wstęp
- Wstępne przetwarzanie tekstu: tokenizacja, stemming, lematyzacja
- Praca z plikami
- Korpus, biblioteka gensim
- Model BoW
- Model Tf-idf
- Biblioteka SpaCy
- Biblioteka Polyglot
- Klasyfikacja NLP w sklearn

NLP

WSTEP

Czym jest przetwarzanie języka naturalnego (ang. Natural Language Processing, NLP) ?

Właśnie to robisz. Czytasz napisane tu słowa i tworzysz z nich jakąś semantyczą reprezentacje (zapamiętujesz znaczenie tego co jest napisane, nie zapamiętując dokładnej składni - słowa po słowie). Gdyby ktoś zapytał się teraz Ciebie, o czym był tekst, który przed chwilą przeczytałeś, to bezbłędnie odpowiesz. Ale gdyby ktoś zapytał sie Ciebie jak dokładnie brzmiało zdanie, które przeczytałeś, to odpowiedź, chociaż teoretycznie znacznie prostsza do sformułowania, praktycznie okazałaby się znacznie trudniejsza do udzielenia.

Kiedy podobnego typu zdanie dajemy komputerowi do zrealizowania, to to właśnie nazywamy nlp.

Zadania realizowane w ramach nlp mogą dotyczyć między innymi automatyzacji analizy składniowej lub semantycznej tekstu, automatyzacji rozumienia, tłumaczenia i generowania języka naturalnego przez komputer.

Przykładowe zastosowania:

1. tłumaczenie maszynowe (machine translation)
2. w czasach chat gpt, wspominanie o wirtualnych asystentach (Siri, Alexa) jest pewnie zbędne
3. analiza sentymentu (jakie zabarwienie emocjonalne ma przetwarzany tekst)
4. detekcj spamu
5. i wiele innych

NLP

WSTEP

Nieustrukturyzowane dane (tekst)

Dodaj jajka i mleko do mojej listy zakupów

Ustrukturyzowane dane

```
<shopping list>
    <item>eggs</item>
    <item>milk</item>
</shopping list>
```

NLP

WSTEP

Nieustrukturyzowane dane (tekst)

Dodaj jajka i mleko do mojej listy zakupów

NLG

(**N**atural **L**anguage **G**eneration)

NLU

(**N**atural **L**anguage **U**nderstanding)

Ustrukturyzowane dane

```
<shopping list>
    <item>eggs</item>
    <item>milk</item>
</shopping list>
```

NLP

WSTEP

NLP to nie jest jakiś jeden określony algorytm. To raczej skrzynka z wieloma różnymi narzędziami. Największe osiągnięcia w tej dziedzinie zawdzięczmy obecnie sieciom neuronowym (wcześniej rekurencyjnym sieciom neuronowym, dzisiaj transformerom). My na razie przyjrzymy się prostszym narzędziom.

NLTK 3.6.2 documentation

Natural Language Toolkit

NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active discussion forum.

Thanks to a hands-on guide introducing programming fundamentals alongside topics in computational linguistics, plus comprehensive API documentation, NLTK is suitable for linguists, engineers, students, educators, researchers, and industry users alike. NLTK is available for Windows, Mac OS X, and Linux. Best of all, NLTK is a free, open source, community-driven project.

NLTK has been called "a wonderful tool for teaching, and working in, computational linguistics using Python," and "an amazing library to play with natural language."

Natural Language Processing with Python provides a practical introduction to programming for language processing. Written by the creators of NLTK, it guides the reader through the fundamentals of writing Python programs, working with corpora, categorizing text, analyzing linguistic structure, and more. The online version of the book has been been updated for Python 3 and NLTK 3. (The original Python 2 version is still available at http://nltk.org/book_1ed.)

Some simple things you can do with NLTK

Tokenize and tag some text:

```
>>> import nltk
>>> sentence = """At eight o'clock on Thursday morning
... Arthur didn't feel very good."""
>>> tokens = nltk.word_tokenize(sentence)
>>> tokens
['At', 'eight', "o'clock", 'on', 'Thursday', 'morning',
```


SEARCH

Go

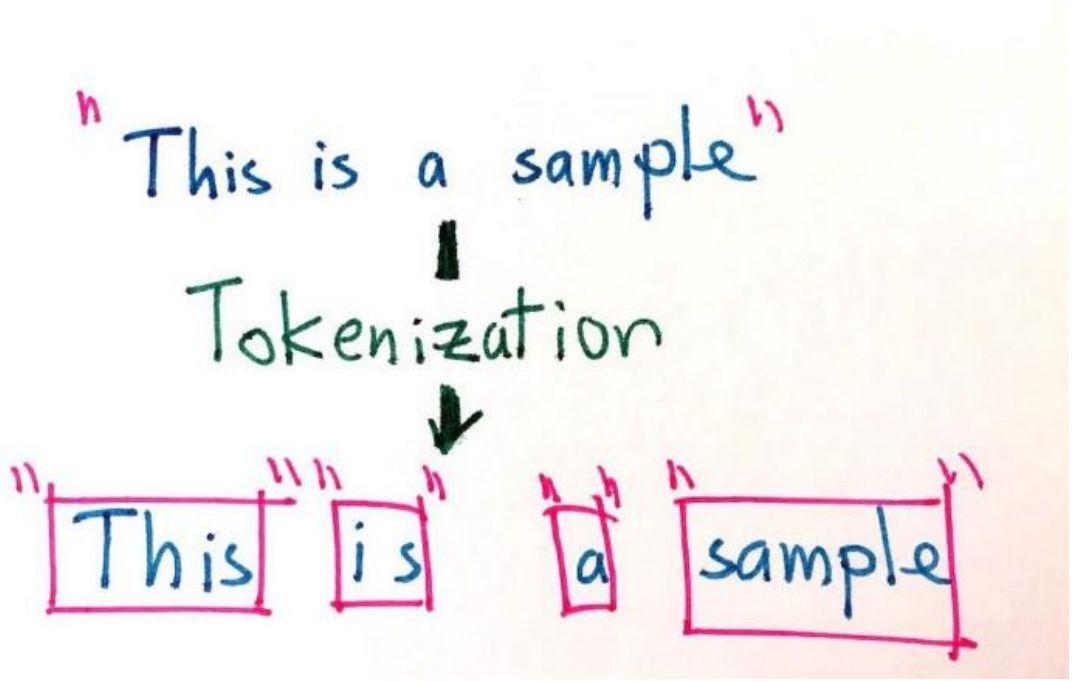
NLP

WSTEP

Przykładowy flow w przetwarzaniu języka naturalnego:

1. Wejściem jest jakiś nieustruktyrozowany tekst (pisany lub mówiony, który został przekonwertowany do pisanego za pomocą jakiegoś algorytmu speech-to-text.
2. Tokenizacja (ang. tokenization) - rozbicie tekstu na tokeny (często po prostu na słowa)
3. Stemming - wyodrębnianie rdzenia z tokenu na podstwie jego struktury (lepiej – lep)
4. Lematyzacja (ang. lemmatization) - wyodrębnienie rdzenia z tokenu na podstawie jego znaczenia (lepiej – dobrze)
5. Tagownie części mowy (ang. part of speech tagging) - identyfikowanie jaką częścią mowy jest dany token
6. Named entity recognition (NER) - identyfikowanie kategorii tokenu (np. słowo Polsk i kategoria kraj)

TOKENIZACJA



NLP

WSTEP

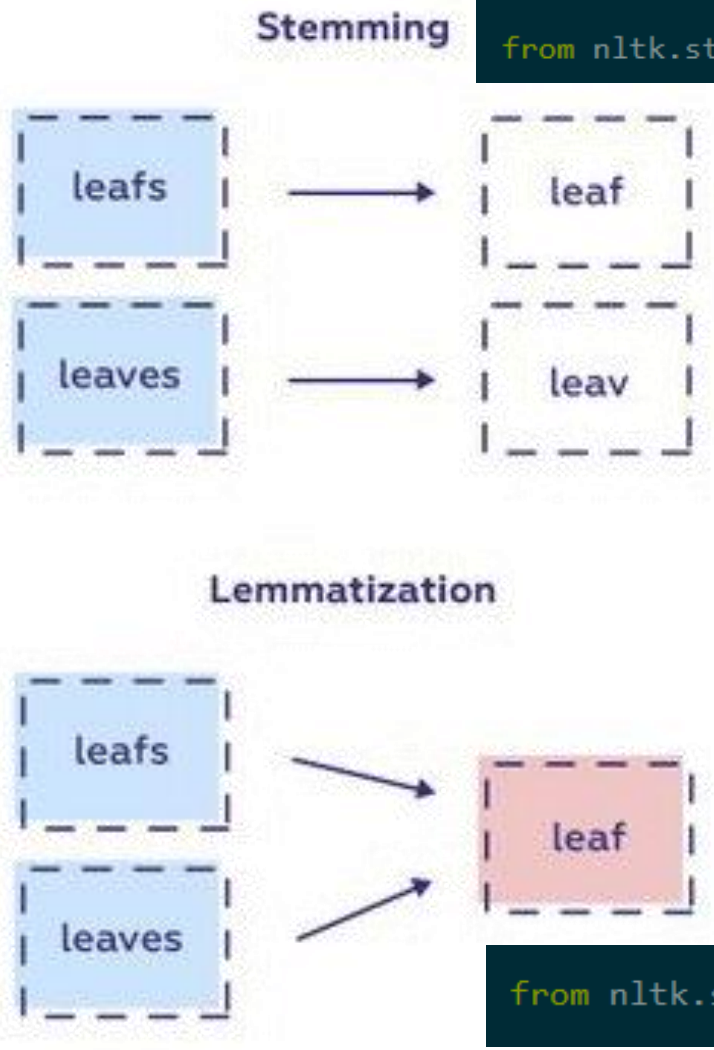
Jednym z częstszych technik prepocessingu w naturalnym przetwarzaniu sygnału jest tokenizacja, czyli rozbicie tekstu na umownie przyjęte tokeny - części składowe (cegiełki) z których składa się tekst.

Innymi, często stosowanymi w preprocessingu technikami są:

- Lematyzajca
- Stemming
- Czyszczenie (usuwanie stop words, znaków interpunkcji oraz niechcianych tokenów)
- Osadzanie słów

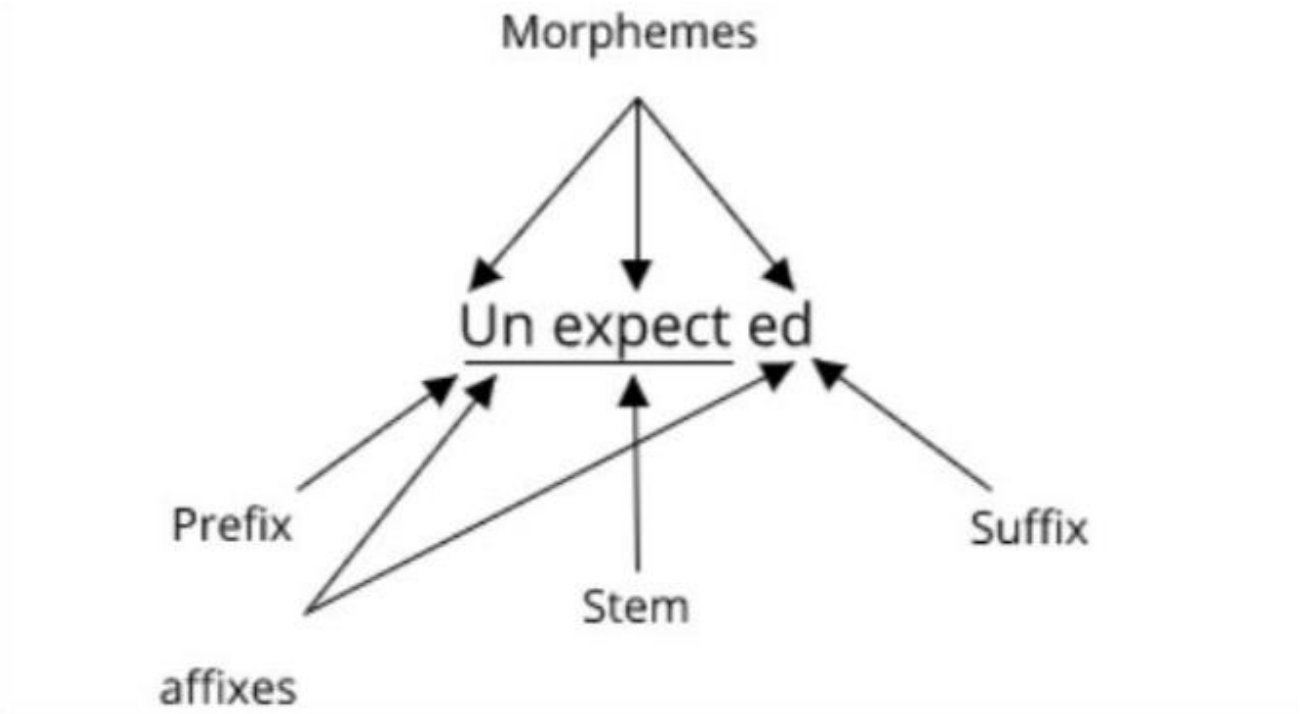
STEMMING
LEMATYZACJA

```
from nltk.stem import PorterStemmer
from nltk.stem import LancasterStemmer
from nltk.stem.snowball import SnowballStemmer
```




```
from nltk.stem import WordNetLemmatizer
```

MORFOLOGIA



OSADZANIE SŁÓW

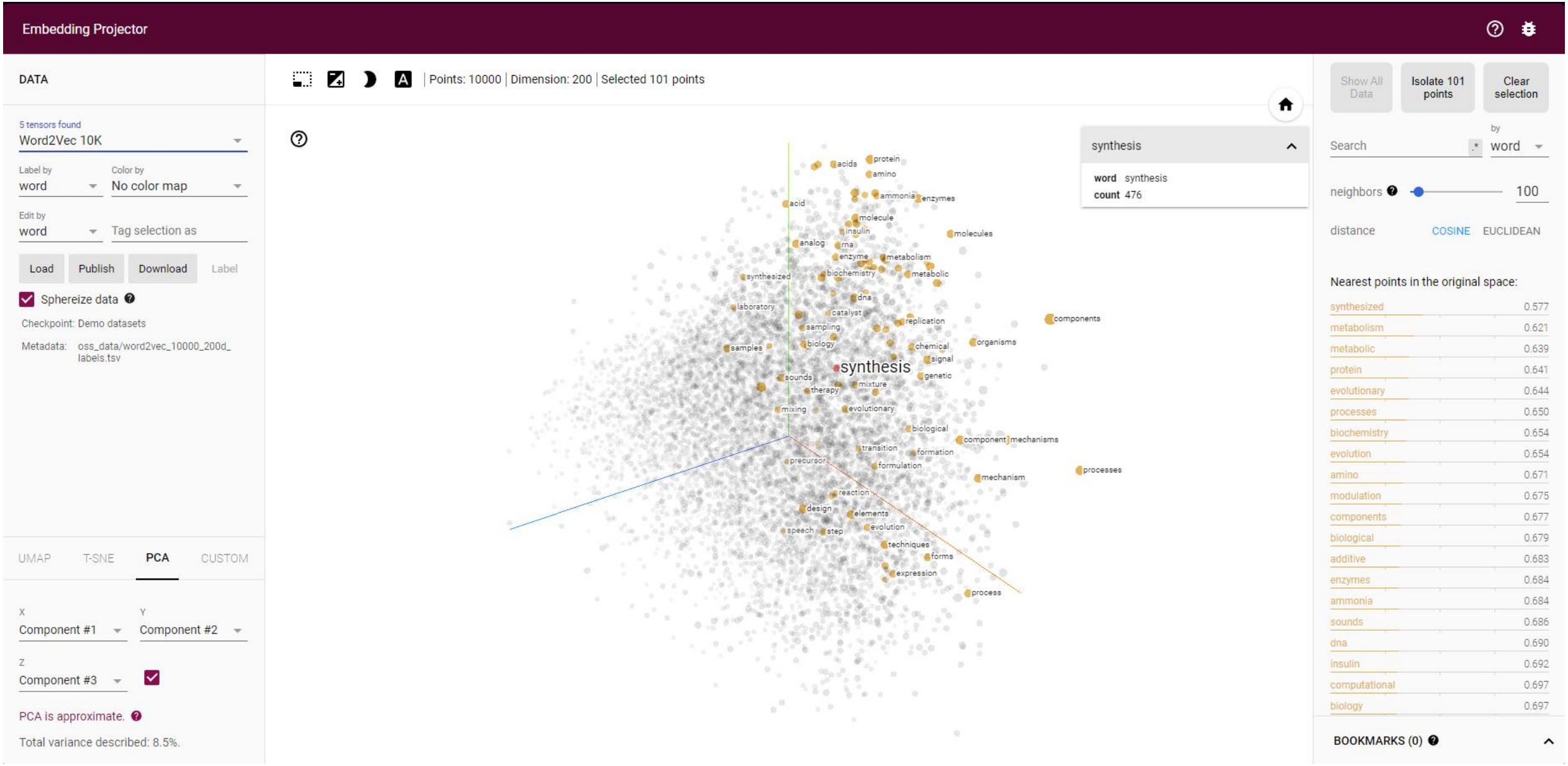
Embeddings

EMBEDDING

OSADZANIA

Po wykonaniu tokenizacji dysponujemy zestawem słów występujących w tekście.

Ale modele uczenia maszynowego działają głównie na wektorach liczbowych a nie na napiach. Po tokenizacji należy zatem zadbać o liczbową reprezentacje tokenów. W jaki sposób możemy reprezentowac słowa za pomocą wektorów ?



EMBEDDING

OSADZANIE

EMBEDDING

OSADZANIA

Jedną z najprostszych metod osadzania słów jest Bag of Word.

MODEL BOW

$$w_{i,j} = tf_{i,j}$$

$$w_{i,j} = \text{tf weight for token } i \text{ in document } j$$

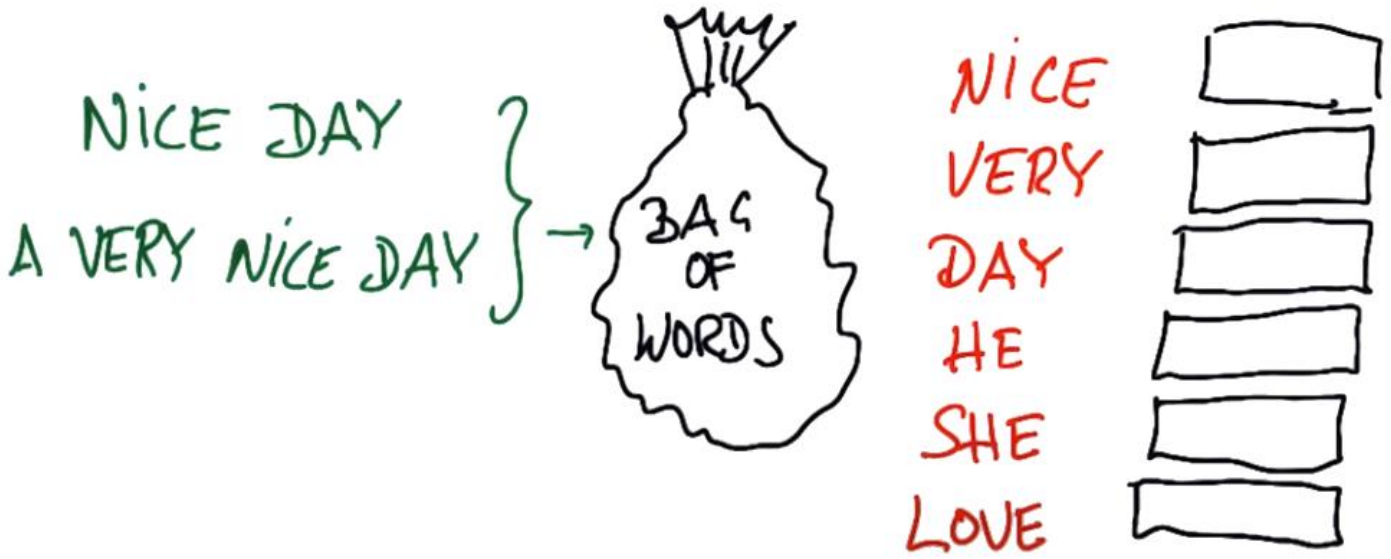
$$tf_{i,j} = \text{number of occurences of token } i \text{ in document } j$$

BOW

BAG OF WORDS

Bag of Wods to metoda na identifykacji znaczących słów w tekście na podstawie liczby ich wystąpień.

Po wykonaniu tokenizacji, możemy zliczyć wszystkie zidentyfikowane tokeny. Zgodnie z BoW im więcej razy wystąpiło jakieś słowo w tekście, tylko bardziej centralne (ważniejsze) jest dla tego tekstu.

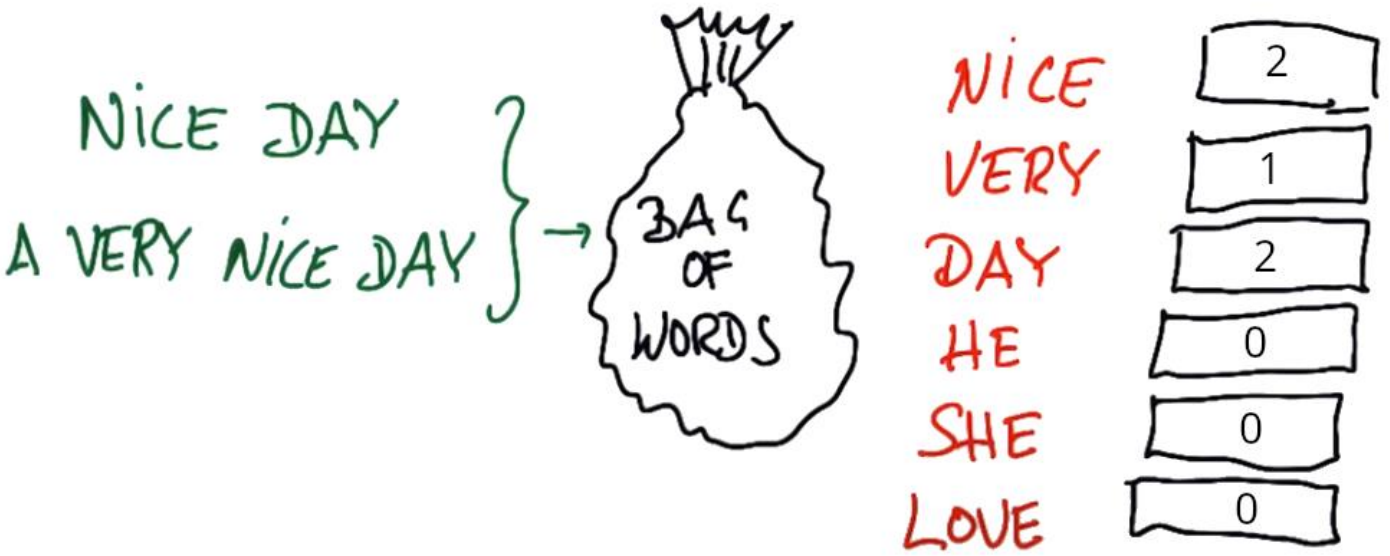


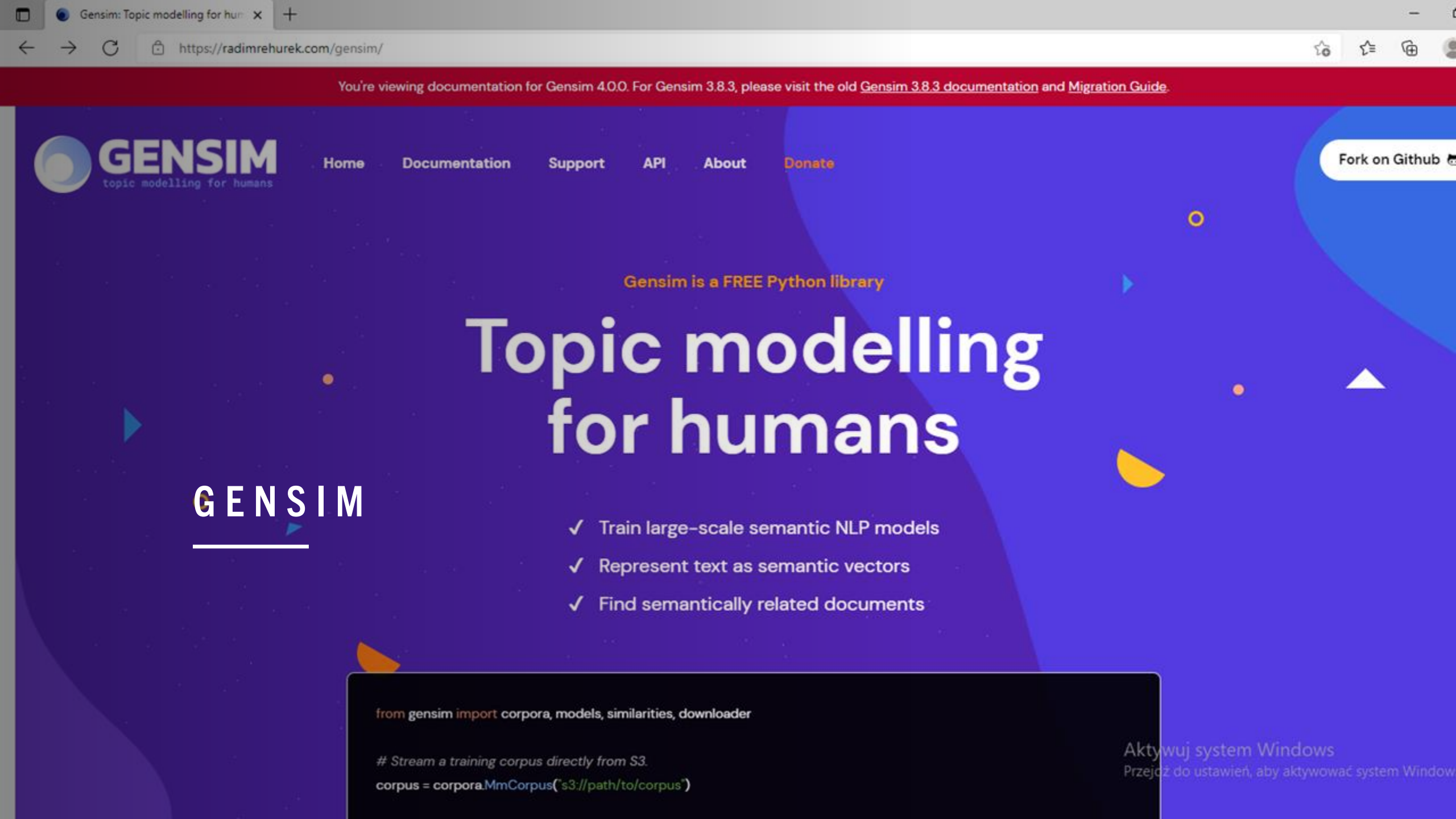
BOW

BAG OF WORDS

Bag of Wods to metoda na identifykacji znaczących słów w tekście na podstawie liczby ich wystąpień.

Po wykonaniu tokenizacji, możemy zliczyć wszystkie zidentyfikowane tokeny. Zgodnie z BoW im więcej razy wystąpiło jakieś słowo w tekście, tylko bardziej centralne (ważniejsze) jest dla tego tekstu.




Gensim: Topic modelling for hur
https://radimrehurek.com/gensim/
You're viewing documentation for Gensim 4.0.0. For Gensim 3.8.3, please visit the old Gensim 3.8.3 documentation and Migration Guide.
GENSIM
topic modelling for humans
Home
Documentation
Support
API
About
Donate
Fork on Github
Gensim is a FREE Python library
Topic modelling for humans
GENSIM
✓ Train large-scale semantic NLP models
✓ Represent text as semantic vectors
✓ Find semantically related documents
from gensim import corpora, models, similarities, downloader
# Stream a training corpus directly from S3.
corpus = corpora.MmCorpus("s3://path/to/corpus")
Aktywuj system Windows
Przejdź do ustawień, aby aktywować system Windows

MODEL TF-IDF

$$w_{i,j} = tf_{i,j} * \log(\frac{N}{df_i})$$

$w_{i,j}$ = tf-idf weight for token i in document j

$tf_{i,j}$ = number of occurences of token i in document j

df_i = number of documents that contain token i

N = total number of documents

TF-IDF

TIME FREQUENCY-INVERSE DOCUMENT FREQUENCY

tf-idf to często wykorzystywany w przetwarzaniu języka naturalnego model, którego zadaniem jest zidentyfikowanie najbardziej istotnych tokenów w każdym dokumencie korpusu.

tf-idf opiera się na założeniu, że poza samymi stopwords w dokumencie mogą znajdować się inne, mało znaczące, a często powtarzające się tokeny. Jak je zidentyfikować ?

Można zestawić ze sobą częstość występowania tokenu w dokumencie, z częstością występowania tego tokenu w pozostałych dokumentach korpusu. Jeżeli te dwie częstości są podobne oznacza to, że ten token najprawdopodbniej jest właśnie takim stopwords nie uwzględnionym na liście stopwords. A przynajmniej jego waga jest mniejsza niż wynikałoby z analizy tylko jednego dokumnetu.

Model tf-ifd zapewnia, że najczęstsze słowa w całym korpusie nie będą wyświetlane jako słowa kluczowe dokumentu. Tf-ifd pomaga zachować specyfikę dokumentu, ważąc nisko słowa często występujące w całym korpusie.

Z modelu tf-idf możemy łatwo skorzystać za pośrednictwem biblioteki Gensim

TF-IDF

TIME FREQUENCY-INVERSE DOCUMENT FREQUENCY

tf – miara częstości występowania tokenu w dokumencie korpusu (jak często)

$$tf(t,d) = \frac{\#\text{wystąpień } t \text{ w } d}{\#\text{słów w } d}$$

gdzie: t – token, d – dokument

idf – miara częstości występowania tokenu we wszyskich dokumentach korpusu (jak rzadko)

$$idf(t,d) = \log \frac{\#\text{dokumentów}}{\#\text{dokumentów z } t}$$

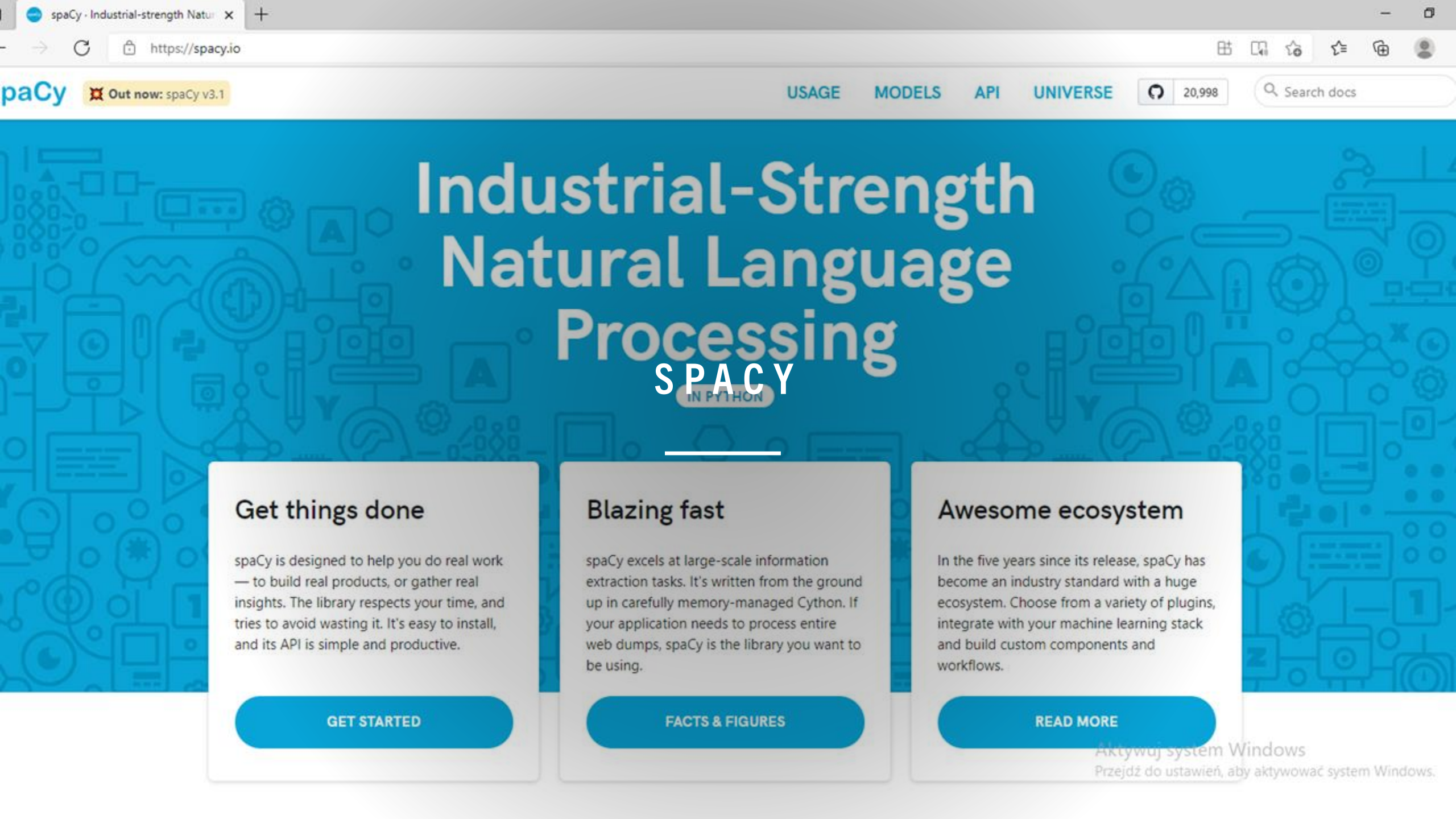
, gdzie t – token, d – dokument

NER

In 1917, Einstein applied the general theory of relativity to model the large-scale structure of the universe. He was visiting the United States when Adolf Hitler came to power in 1933 and did not go back to Germany, where he had been a professor at the Berlin Academy of Sciences. He settled in the U.S., becoming an American citizen in 1940. On the eve of World War II, he endorsed a letter to President Franklin D. Roosevelt alerting him to the potential development of "extremely powerful bombs of a new type" and recommending that the U.S. begin similar research. This eventually led to what would become the Manhattan Project. Einstein supported defending the Allied forces, but largely denounced using the new discovery of nuclear fission as a weapon. Later, with the British philosopher Bertrand Russell, Einstein signed the Russell-Einstein Manifesto, which highlighted the danger of nuclear weapons. Einstein was affiliated with the Institute for Advanced Study in Princeton, New Jersey, until his death in 1955.

Tag colours:

LOCATION TIME PERSON ORGANIZATION MONEY PERCENT DATE


spaCy · Industrial-strength Natur
https://spacy.io
spaCy
Out now: spaCy v3.1
USAGE
MODELS
API
UNIVERSE
20,998
Search docs
Industrial-Strength Natural Language Processing
SPACY
IN PYTHON
Get things done
spaCy is designed to help you do real work — to build real products, or gather real insights. The library respects your time, and tries to avoid wasting it. It's easy to install, and its API is simple and productive.
GET STARTED
Blazing fast
spaCy excels at large-scale information extraction tasks. It's written from the ground up in carefully memory-managed Cython. If your application needs to process entire web dumps, spaCy is the library you want to be using.
FACTS & FIGURES
Awesome ecosystem
In the five years since its release, spaCy has become an industry standard with a huge ecosystem. Choose from a variety of plugins, integrate with your machine learning stack and build custom components and workflows.
READ MORE
Aktywuj system Windows
Przejdź do ustawień, aby aktywować system Windows.

DODATKI

WYRAŻENIA REGULARNE

pattern	matches	example
\w+	word	'Magic'
\d	digit	9
\s	space	' '
.*	wildcard	'username74'
+ or *	greedy match	'aaaaaa'
\S	**not** space	'no_spaces'
[a-z]	lowercase group	'abcdefg'

WYRAŻENIA REGULARNE

pattern	matches	example
[A-Za-z]+	upper and lowercase English alphabet	'ABCDEFghijk'
[0-9]	numbers from 0 to 9	9
[A-Za-z\-\.]+	upper and lowercase English alphabet, - and .	'My-Website.com'
(a-z)	a, - and z	'a-z'
(\s+l,)	spaces or a comma	', '

re — Regular expression operations

Source code: Lib/re.py

This module provides regular expression matching operations similar to those found in Perl.

Both patterns and strings to be searched can be Unicode strings (`str`) as well as 8-bit strings (`bytes`). However, Unicode strings and 8-bit strings cannot be mixed: that is, you cannot match a Unicode string with a byte pattern or vice-versa; similarly, when asking for a substitution, the replacement string must be of the same type as both the pattern and the search string.

BIBLIOTEKA RE

Regular expressions use the backslash character (`'\'`) to indicate special forms or to allow special characters to be used without invoking their special meaning. This collides with Python's usage of the same character for the same purpose in string literals; for example, to match a literal backslash, one might have to write `'\\\\'` as the pattern string, because the regular expression must be `\\`, and each backslash must be expressed as `\\` inside a regular Python string literal. Also, please note that any invalid escape sequences in Python's usage of the backslash in string literals now generate a `DeprecationWarning` and in the future this will become a `SyntaxError`. This behaviour will happen even if it is a valid escape sequence for a regular expression.

The solution is to use Python's raw string notation for regular expression patterns; backslashes are not handled in any special way in a string literal prefixed with `'r'`. So `r"\n"` is a two-character string containing `'\'` and `'n'`, while `"\n"` is a one-character string containing a newline. Usually patterns will be expressed in Python code using this raw string notation.

It is important to note that most regular expression operations are available as module-level functions and methods on compiled regular expressions. The functions are shortcuts that don't require you to compile a regex object first, but miss some fine-tuning parameters.

See also: The third-party regex module, which has an API compatible with the standard library `re` module, but offers additional functionality and a more thorough Unicode support.

Regular Expression Syntax

NLP PIPELINE