Corpus preprocessing, cleaning and homogeneization with NLTK examples

Jerónimo Arenas-García, Vanessa Gómez-Verdejo, Jesús Cid-Sueiro

Universidad Carlos III de Madrid

jeronimo.arenas@uc3m.es

March 16, 2021

Document representation for Machine Learning

- ML algorithms process numbers, not words
- We need to transform text into meaningful numbers
- Bag-of-Words (BoW) representation: Count number of appearances of each word (in the vocabulary of all documents) in a given document
- In order to have a useful representation, some preprocessing steps are normally required

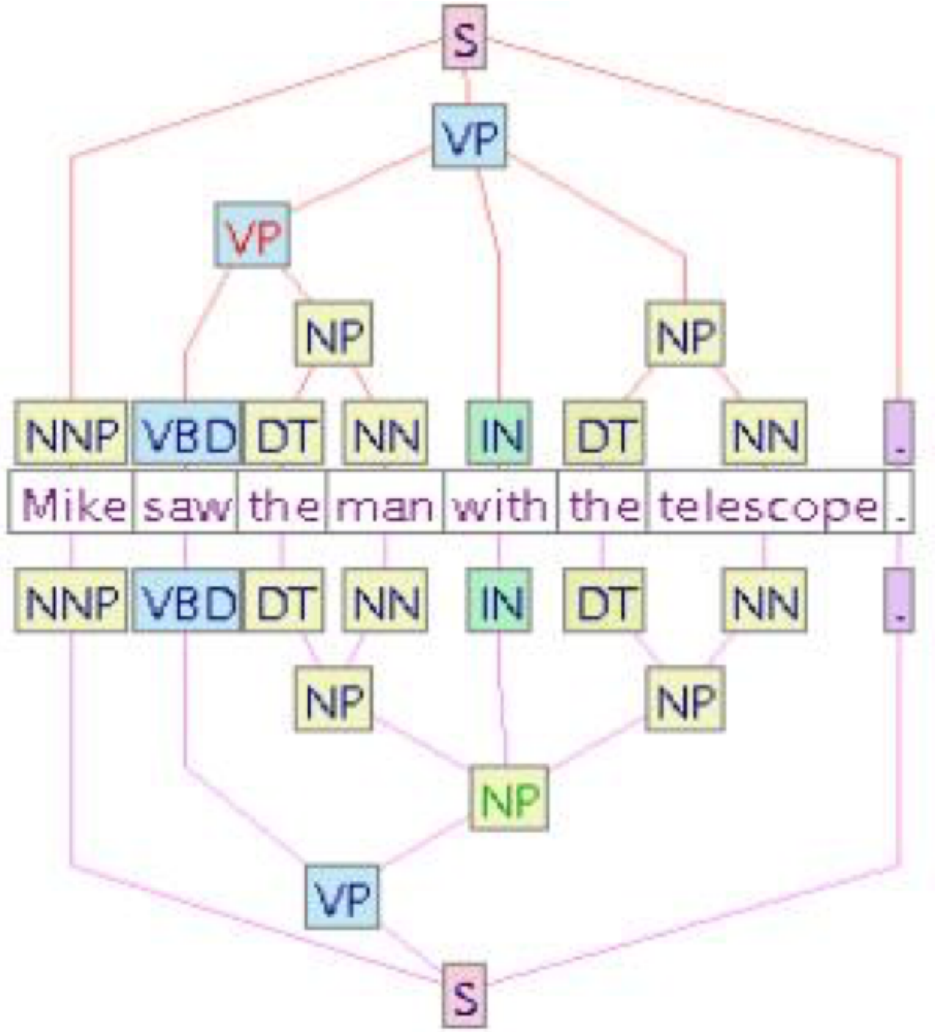
NLP with Python

NLTK:

- Basic classes for representing data relevant to NLP
- Functions for performing common NLP tasks
- Corpuses

Spacy:

- NLP Pipelines ready for production
- Incorporates pre-trained Neural Networks Models
- It is possible to fine tune the models



NLTK modules

- corpora: a package containing collections of example text
- **tokenize**: functions to split text strings into basic elements
- probability: for modeling frequency distributions and probabilistic systems
- **stem**: package of functions to stem words of text
- **wordnet**: interface to the WordNet lexical resource
- chunk: identify short non-nested phrases in text
- **tag**: tagging each word with part-of-speech, sense, etc.
- parse: building trees over text - recursive descent, shift-reduce, probabilistic, etc.
- cluster: clustering algorithms
- draw: visualize NLP structures and processes
- contrib: various pieces of software from outside contributors

Tokenization

- The goal is to find the basic elements (tokens) of a given string.
- Python `split()` function makes a list from a string using a given substring as a separator
- NLTK tokenizers are better suited to this task, and more efficient since they use regular expressions.
- Includes both sentence and word tokenizers

Example

```
≫ sentence = 'Hello, world.'
≫ sentence.split()
['Hello,', 'world.']
≫ from nltk.tokenize import word_tokenize
≫ tokens = word_tokenize(sentence)
≫ tokens = [el for el in tokens if el.isalnum()]
```

Homogenization

- Collapse semantically equivalent words into a unique representative
  - Different forms of a word:
    organize, organizes, organizing, ... $\rightarrow$ organize
  - Derivationally related words with similar meanings:
    democracy, democratic, democratization, ... $\rightarrow$ democracy
- Stemming vs lemmatization
  - **Stemming** chops of the ends of words using a list of suffixes
  - **Lemmatization** usually refers to doing things properly with a vocabulary and morphological analysis of words, aiming to return the base or dictionary form (lemma) of a word.
  - A practical difference is that lemmatizers output complete words.
- An additional benefit is a smaller vocabulary size (vector length)

Stemming

- Porter algorithm: developed by Martin Porter in 1980. Not very aggressive
- Based on a set of expert rules. Ignores words roles
- NLTK incorporates SnowBallStemmer that fixes some issues of the Porter algorithm
- Output not necessarily 'proper' words

```
≫ import nltk.stem
≫ s = nltk.stem.SnowballStemmer('english')
≫ s.stem('image') → 'imag'
≫ s.stem('images') → 'imag'
≫ s.stem('organize') → 'organ'
≫ s.stem('organizing') → 'organ'
≫ s.stem('organ') → 'organ'
```

Lemmatization

- NLTK recurs to WordNet, a lexical database for the English language. Wordnet groups English words into sets of synonyms called synsets.
- With contextual information (the grammatical role of the word) `lemmatize()` can filter grammatical differences

```
>> from nltk.stem import WordNetLemmatizer
>> wnl = WordNetLemmatizer()
>> wnl.lemmatize('image') → 'image'
>> wnl.lemmatize('images') → 'image'
>> wnl.lemmatize('organize') → 'organize'
>> wnl.lemmatize('organizing') → 'organizing'
>> wnl.lemmatize('organizing', pos='v') → 'organize'
>> wnl.lemmatize('organ') → 'organ'
```

NLP lemmatizers for Python

- Different algorithm and performances
- Different computational requirements

```
sentence = """Following mice attacks, caring farmers were marching to Delhi for bet
Delhi police on Tuesday fired water cannons and teargas shells at protesting farmers
break barricades with their cars, automobiles and tractors."""

# NLTK
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
pprint(" ".join([lemmatizer.lemmatize(w, get_wordnet_pos(w)) for w in nltk.word_toke
# ('Following mouse attack care farmer be march to Delhi for well living '
#  'condition Delhi police on Tuesday fire water cannon and teargas shell at '
#  'protest farmer a they try to break barricade with their car automobile and '
#  'tractor')

# Spacy
import spacy
nlp = spacy.load('en', disable=['parser', 'ner'])
doc = nlp(sentence)
pprint(" ".join([token.lemma_ for token in doc]))
# ('follow mice attack , care farmer be march to delhi for good living condition '
#  '. delhi police on tuesday fire water cannon and teargas shell at protest '
#  'farmer as -PRON- try to break barricade with -PRON- car , automobile and '
#  'tractor .')
```

```
# TextBlob
pprint(lemmatize_with_postag(sentence))
# ('Following mouse attack care farmer be march to Delhi for good living '
#  'condition Delhi police on Tuesday fire water cannon and teargas shell at '
#  'protest farmer a they try to break barricade with their car automobile and '
#  'tractor')

# Pattern
from pattern.en import lemma
pprint(" ".join([lemma(wd) for wd in sentence.split()]))
# ('follow mice attacks, care farmer be march to delhi for better live '
#  'conditions. delhi police on tuesday fire water cannon and tearga shell at '
#  'protest farmer a they try to break barricade with their cars, automobile and '
#  'tractors.')

# Stanford
pprint(lemmatize_corenlp(conn_nlp=conn_nlp, sentence=sentence))
# ('follow mouse attack care farmer be march to Delhi for better living '
#  'condition Delhi police on Tuesday fire water cannon and tearga shell at '
#  'protest farmer as they try to break barricade with they car automobile and '
#  'tractor')
```

Taken from `https://www.machinelearningplus.com/nlp/lemmatization-examples-python/`

Other homogeneization tasks

Part of Speech Tagging

- Part-of-speech (POS) tagging is the process of assigning a word to its grammatical category (noun, verb, adverb,...), in order to understand its role within the sentence.
- POS tagging provides the contextual information to `lemmatize()` to filter grammatical differences.

```
≫ from nltk import pos_tag
≫ tokens = word_tokenize('This is a simple sentence')
≫ pos_tag(tokens)
[('This', 'DT'), ('is', 'VBZ'), ('a', 'DT'), ('simple', 'JJ'),
('sentence', 'NN')]
```

N-gram detection

- Identification of groups of words that occasionally appear together, and have an inherent semantic value
- E.g.: Machine learning, big data, etc

Cleaning

Stopword removal

- Stopwords: Words that appear very often in all sorts of different contexts (language function words). They are removed by using stopwords lists
- Very rare terms: Words that appear in a very reduced number of documents in the collection
- Corpus specific stopwords: Words that are very common for a particular data set, or simply do not have significant semantic value for the application at hand. Either manually created lists, or by word frequency.

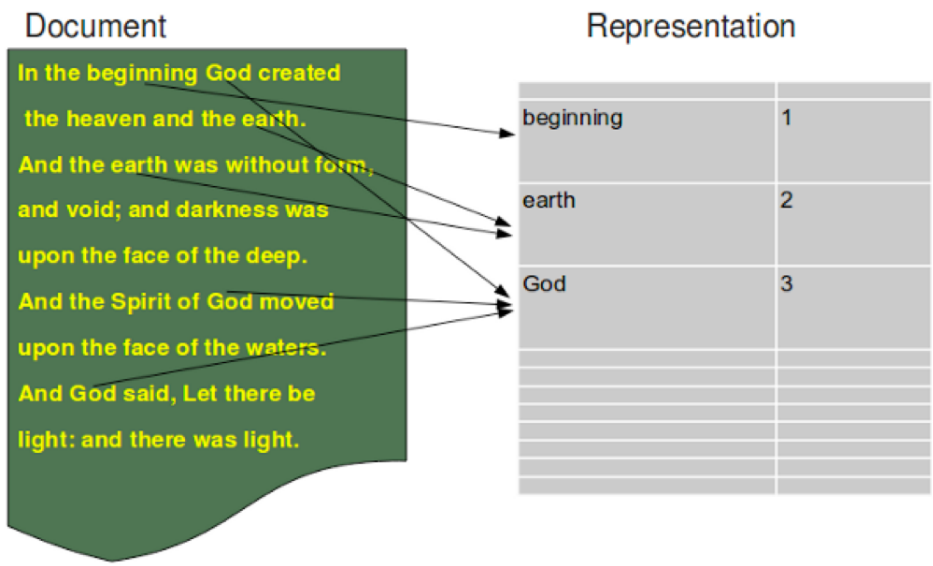
Diccionary of equivalent terms

- Lists of words that should be considered equivalent for the document collection
- Word replacement efficiently implemented using regular expressions `https://www.w3schools.com/python/python_regex.asp`

Vectorization: Bag-of-words representation

The goal is to convert the sequence of homogenized and cleaned tokens into a numeric vector

- Bag-of-words representation: counts how many appearances of each word occur in each document
- Each position in the vector is associated to a unique word in the vocabulary, so vectors are typically very sparse



Term frequency - Inverse document frequency (TF-IDF)

We want a high value for a given term in a given doc if that term occurs often in that particular doc and very rarely anywhere else

- $\mathrm{TF}(w, d) = \frac{\mathrm{BoW}(w,d)}{\#\text{ words in } d}$
- $\mathrm{IDF}(w) = \log \frac{\#\text{ docs}}{\#\text{ docs with term } w}$
- $\mathrm{TF-IDF}(w, d) = \mathrm{TF}(w, d) \times \mathrm{IDF}(w)$

The IDF term emphasizes the words that appear in a reduced number of documents.