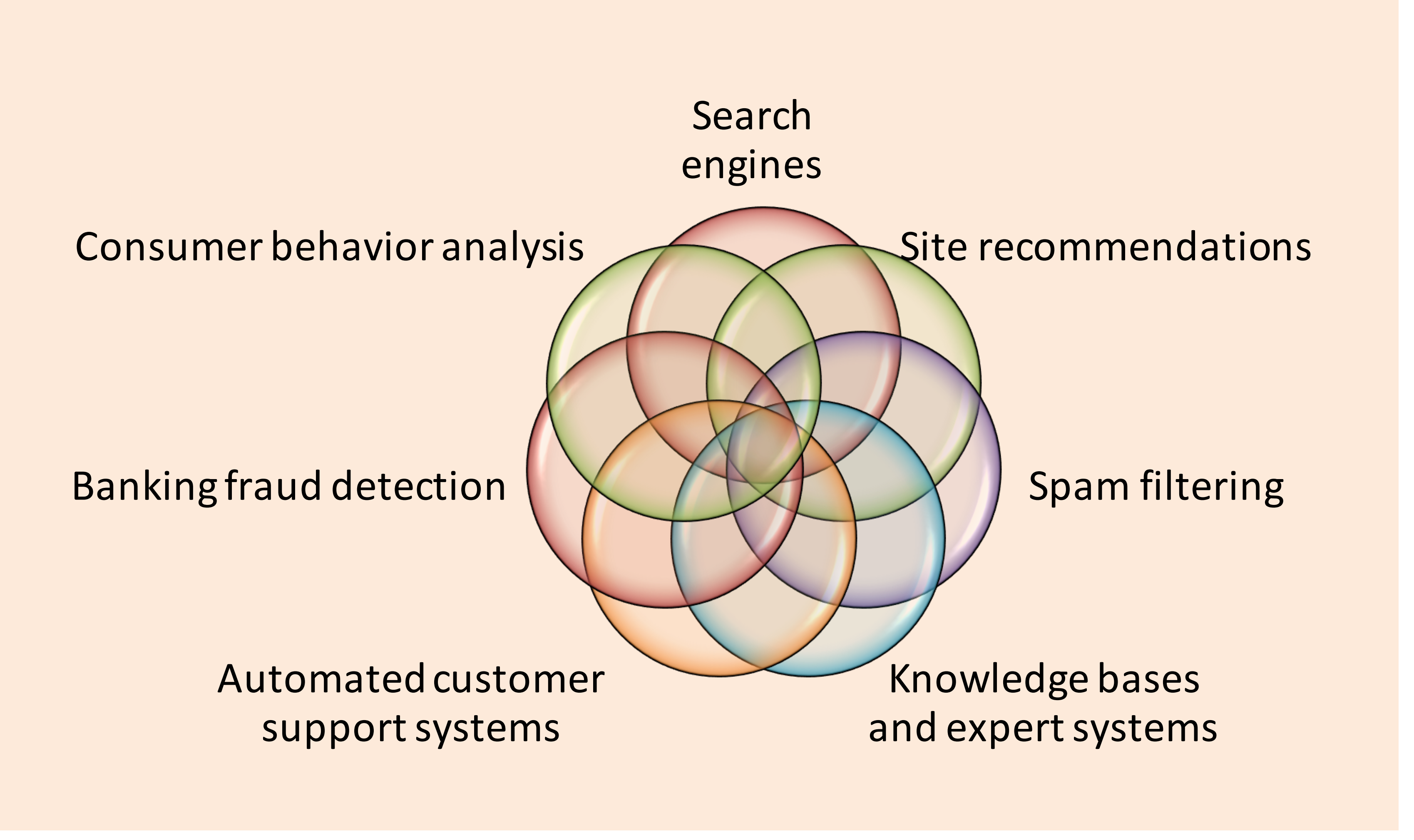
Natural Language Processing with NLTK

Vanessa Gómez Verdejo
Jesús Cid Sueiro

Natural Language Processing

- Natural Language Processing:
  - Computer aided text analysis of human language.
  - The goal is to enable machines to understand human language and extract meaning from text.
  - It is a field of study which falls under the category of machine learning and more specifically computational linguistics.

Applications



Challenges to machines.

Sentiment

Ambiguity

Intent

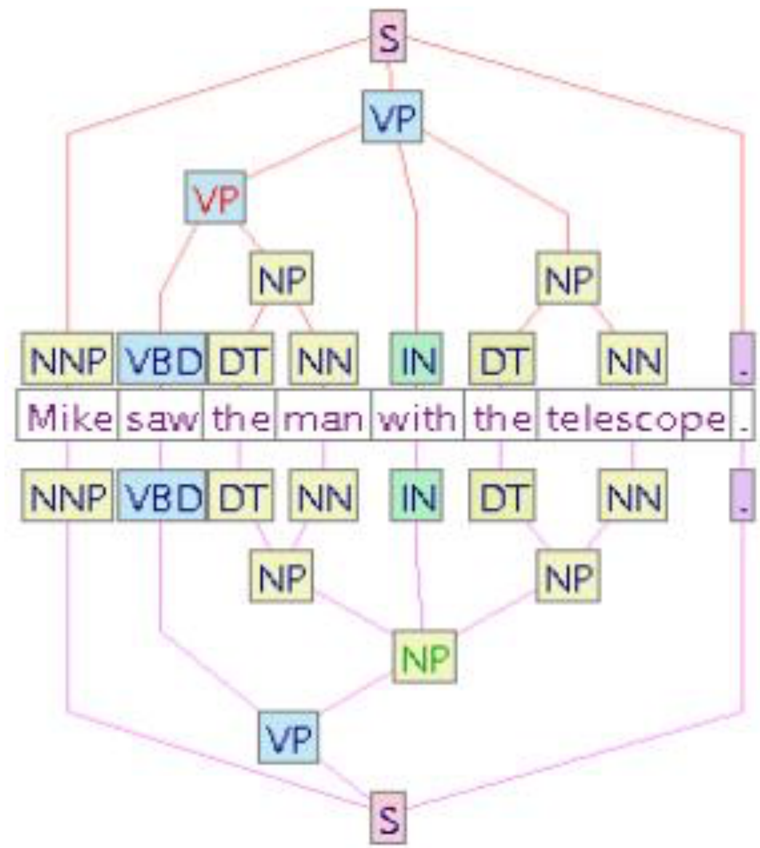
- Sarcasm
- Slang

Context

- Emphasis
- Time and date
  - Since when did “google” become a verb?

NLP with Python

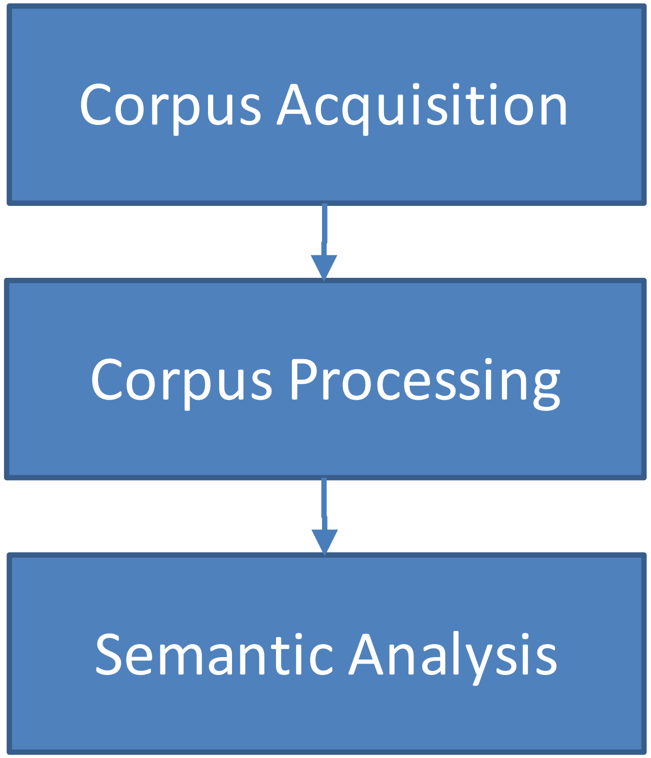
- NLTK: A package that provides
  - basic classes for representing data relevant to Natural Language Processing
  - Standard interfaces for performing NLP tasks such as tokenization, tagging and parsing
  - Standard implementation of each task which can be combined to solve complex problems
  - http://www.nltk.org



NLTK modules

- **corpora**: a package containing modules of example text
- **tokenize**: functions to separate text strings
- 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
- etree: for hierarchical structure over 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

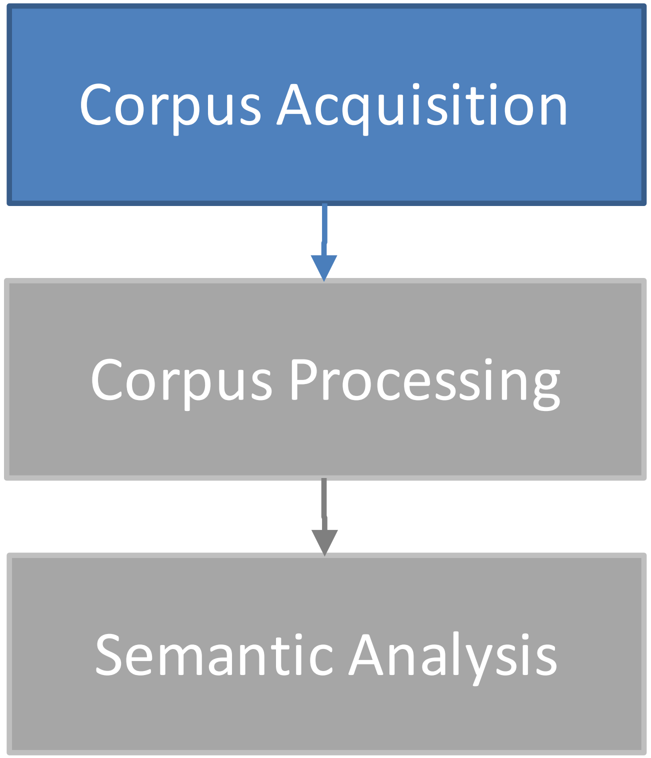
Document Corpus Analysis



Text processing tools:
Natural Language Toolkit (NLTK)

Topic Models: PLSI, LDA

Corpus Acquisition

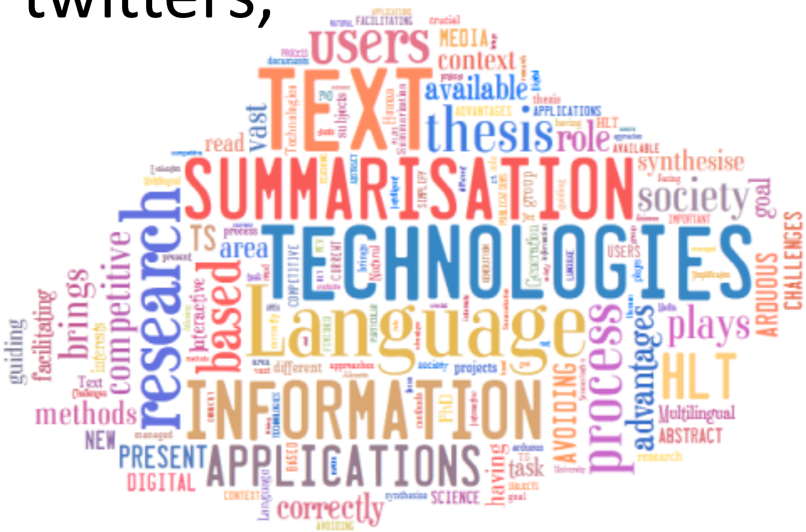


Text processing tools:
Natural Language Toolkit (NLTK)

Topic Models: PLSI, LDA

Text sources

- Any document can be analyzed:
  - Web content: web pages, twitters, blogs, …
    - Crawler
    - Available APIs: wikipedia
  - Local documents
  - Available corpus:
    - NLTK (see http://www.nltk.org/nltk_data/)
    - scikit-learn, gensim,…



Loading a corpus

- From NLTK (pip install nltk)

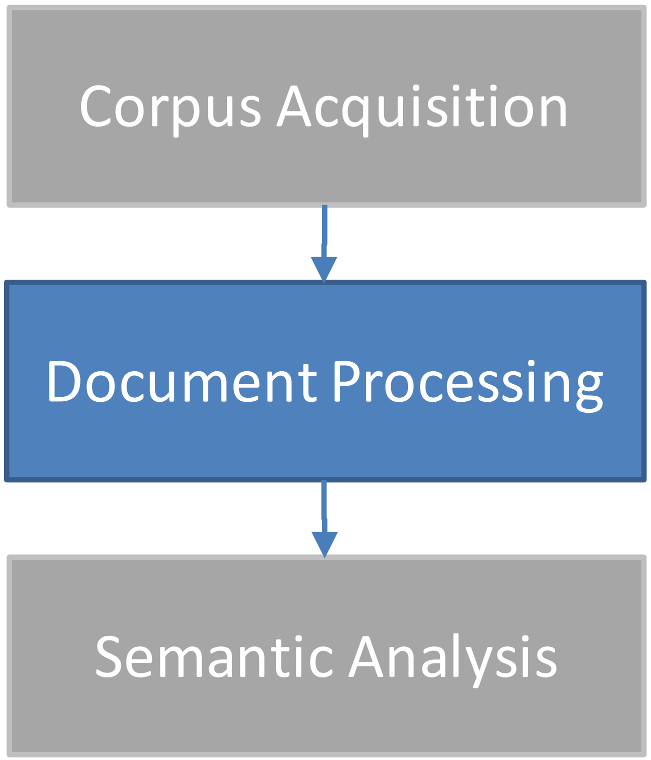
```
import nltk
nltk.download()
Mycorpus = nltk.corpus.gutenberg
text_name = Mycorpus.fileids()[0]
raw = Mycorpus.raw(text_name)
Words = Mycorpus.words(text_name)
```

Install it now and download book content (it takes a while)

```
[Emma by Jane Austen 1816]
VOLUME I
CHAPTER I
Emma Woodhouse, handsome, clever, and
rich, with a comfortable home and happy
disposition, seemed to unite some of
the best blessings of existence; and
had lived nearly twenty-one years in
the world with very little to distress
```

```
[u'[', u'Emma', u'by', u'Jane',
u'Austen', u'1816', u']', u'VOLUME',
u'I', u'CHAPTER', u'I', u'Emma',
u'Woodhouse', u',', u'handsome', u',',
u'clever', u',', u'and', u'rich',
u',', u'with', u'a', u'comfortable',
u'home', u'and', u'happy',
u'disposition', u',', u'seemed',
u'to', u'unite', u'some', u'of', …]
```

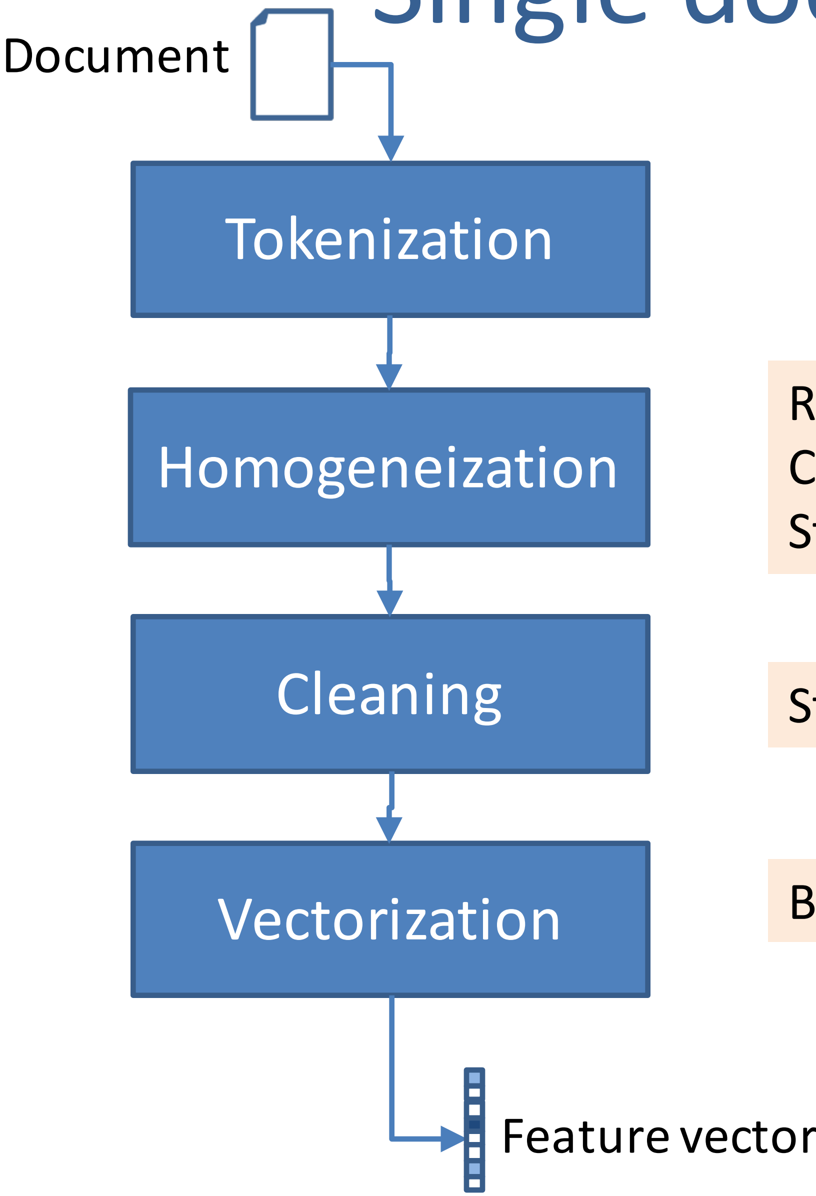
Corpus Processing



Text processing tools:
Natural Language Toolkit (NLTK)

Topic Models: PLSI, LDA

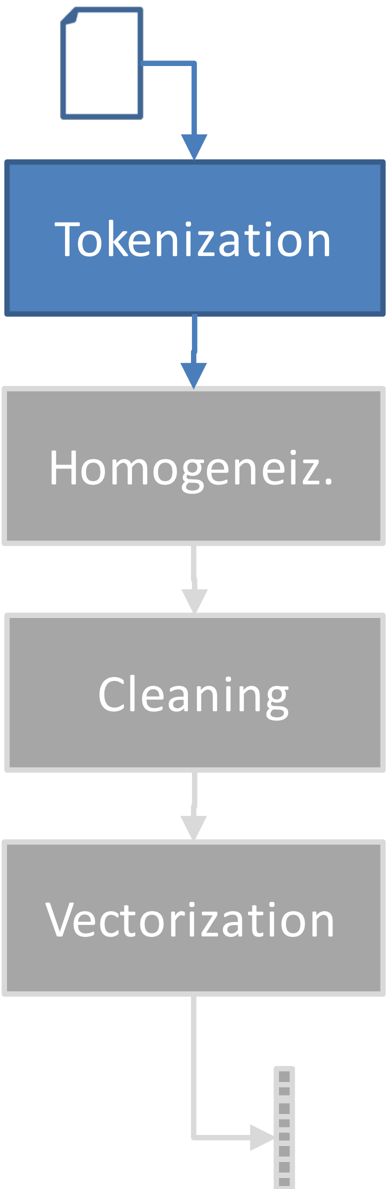
Single document processing



Removing punctuation
Conversion to lowercase
Stemming and lemmatization

Stopword elimination

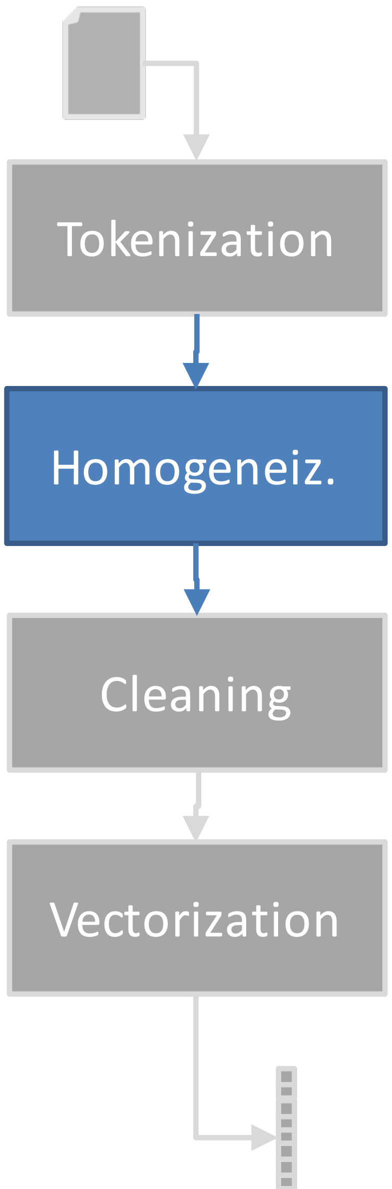
Bag of Words model



Tokenization

- From text to words (elements inside a sentence):

```
>> Words = corpus.words(text_name)
>> sentence = “Hola, mundo.”
>> sentence.split()
[‘Hola,’, ‘mundo.’]
>> from nltk.tokenize import word_tokenize
>> word_tokenize(sentence)
[‘Hola’,  ‘,’, ‘mundo’, ‘.’]
```



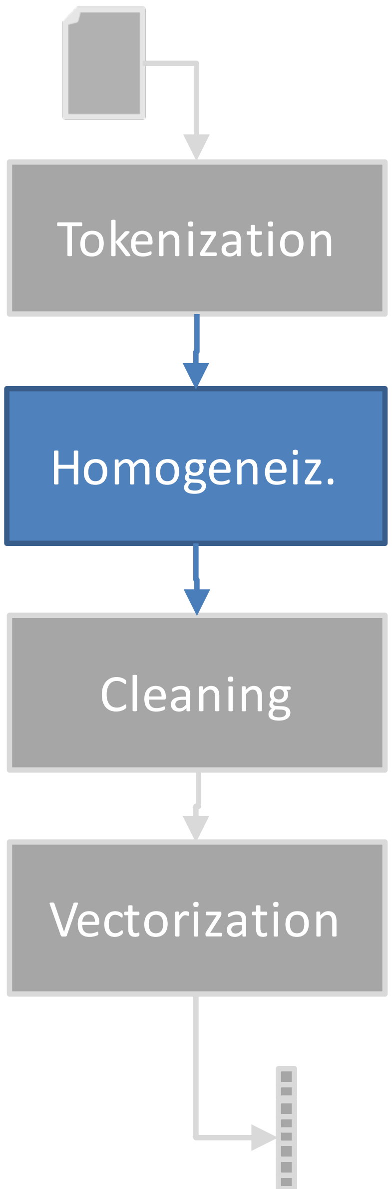
Homogeneization

- EXERCISE 1
  - Convert every word to lowercase

```
>> clean_text = [w.lower() for w in text]
```

- EXERCISE 2
  - Remove punctuation

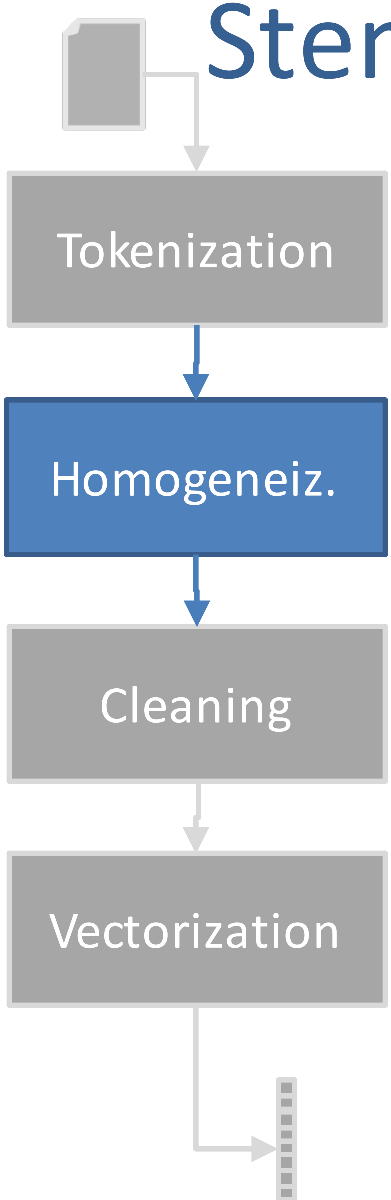
```
>> clean_text = [w for w in text
                 if w.isalnum()]
```



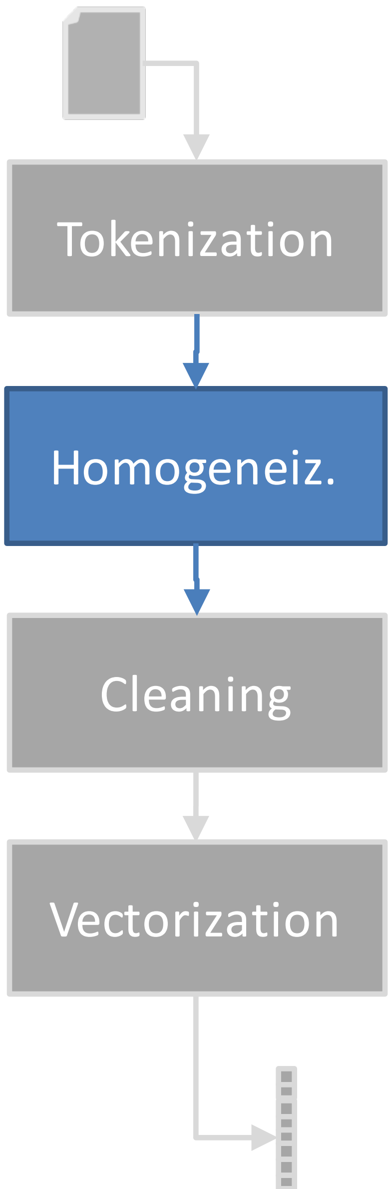
Homogeneization

- The class string in python:
  - s.find(t) index of first instance of string t inside s (-1 if not found)
  - s.rfind(t) index of last instance of string t inside s (-1 if not found)
  - s.join(text) combine the words of the text into a string using s as the glue
  - s.split(t) split s into a list wherever a t is found (whitespace by default)
  - s.lower() a lowercased version of string s
  - s.upper() an uppercased version of string s
  - s.title() a titlecased version of string s
  - s.strip() a copy of s without leading or trailing whitespace
  - s.replace(t, u) replace instances of t with u inside s
  - t in s test if t is contained inside s

Stemming and Lemmatization



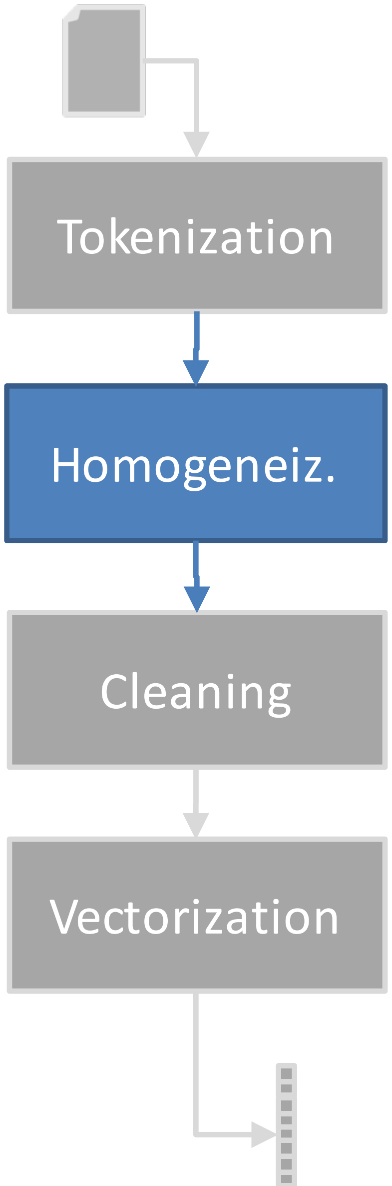
- Motivation:
  - Different forms of a word
    - organize, organizes, organizing.
  - Derivationally related words with similar meanings:
    - democracy, democratic, democratization.
- Goal: to reduce inflectional or derivationally related forms of a word to a common base form.
  - am, are, is ➔ be
  - car, cars, car's, cars' ➔ car
- Stemming:
  - Chops off the ends of words in the hope of achieving this goal correctly most of the time.
  - See, saw ➔ s
- 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.
  - ‘saw’ ➔ ‘see’ if verb, ‘saw’ if noun.
- Stemming vs lemmatization
  - Stemming commonly collapses derivationally related words, whereas lemmatization commonly only collapses the different inflectional forms of a lemma.



Stemming

- We count similar words in different variants as different words
- We need a function that reduces words to their specific word stem.

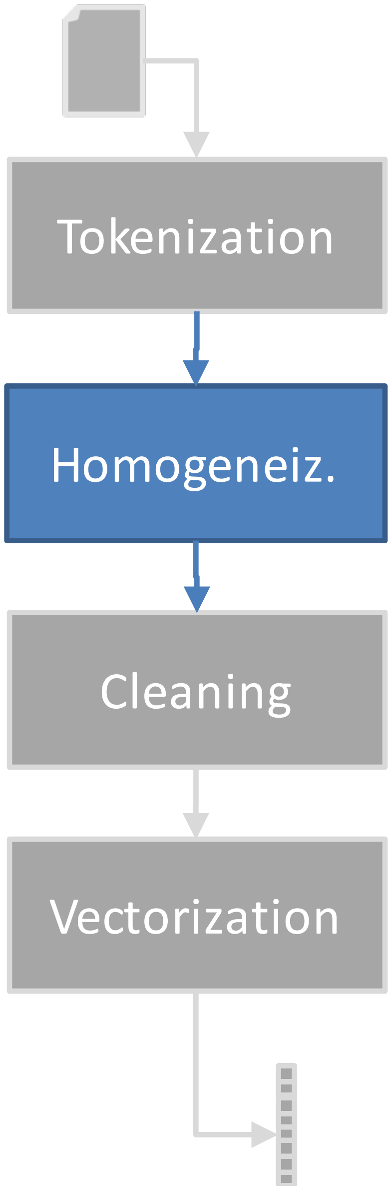
```
import nltk.stem
s = nltk.stem.SnowballStemmer('english')
s.stem(“imaging”)  # → u'imag'
s.stem(“image")    # → u'imag’
```



Lemmatization

- Lemmatization is a more complex process.
- Lemmatization in NLTK uses WordNet.

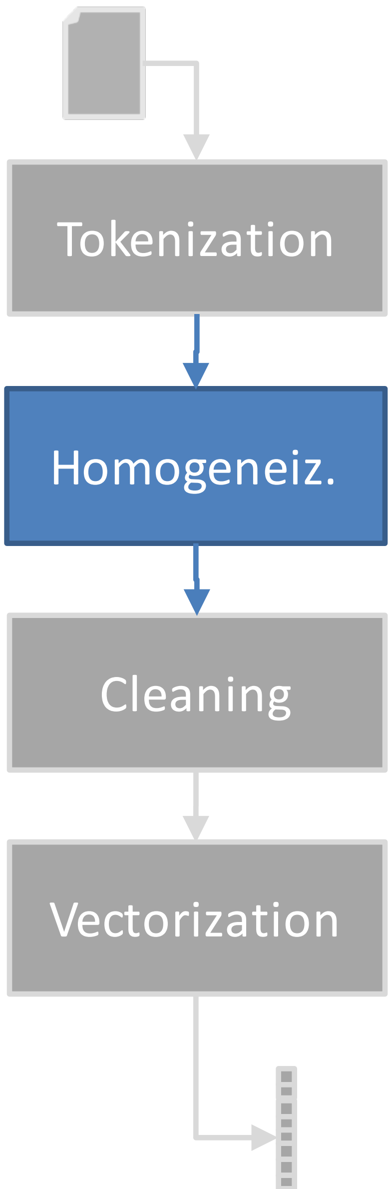
```
from nltk.stem import WordNetLemmatizer
>> wnl = WordNetLemmatizer()
>> print(wnl.lemmatize('dogs'))
dog
>> print(wnl.lemmatize('churches'))
church
>> print(wnl.lemmatize('abaci'))
abacus
```



Lemmatization

- With contextual information (the grammatical role of the word) .lemmatize() can filter grammatical differences.

```
from nltk.stem import WordNetLemmatizer
>> wnl = WordNetLemmatizer()
>> print(wnl.lemmatize(’is'))
is
>> print(wnl.lemmatize(’is’, pos=‘v’))
be
```

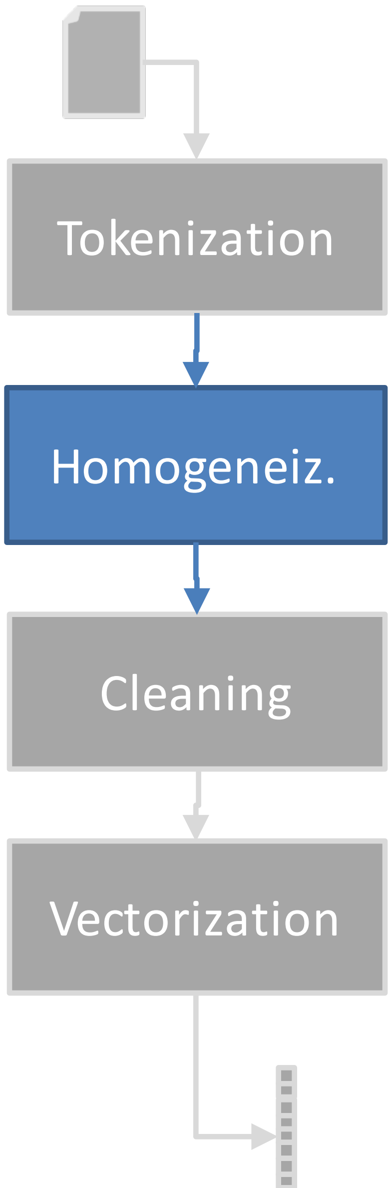


Lemmatization

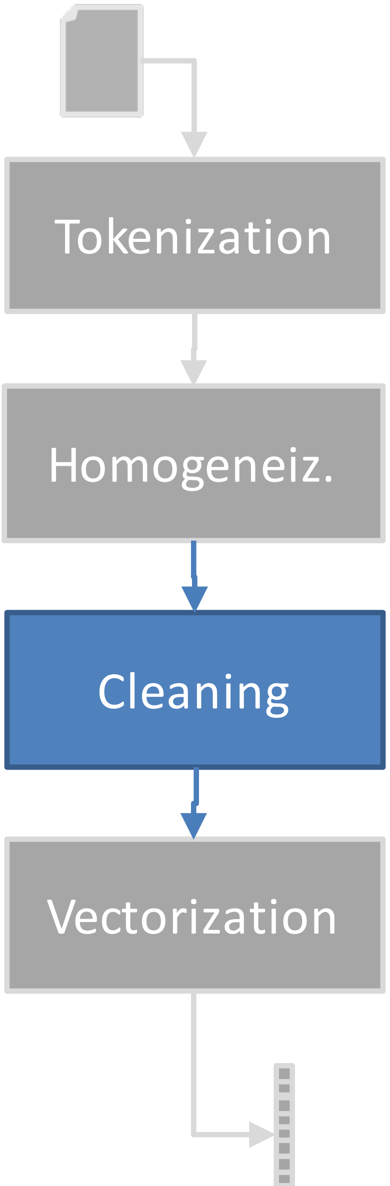
- Part of Speech Tagging.
  - Part-of-speech (POS) tagging is the process of assigning a word to its grammatical category (noun, verb, adverb,…), in order to understand its role within the sentence.
  - POS taggers typically take a sequence of words (i.e. a sentence) as input, and provide a list of tuples (word, pos) as output.
  - POS tagging is what provides the contextual information to .lemmatize() to filter grammatical differences.

```
>> from nltk import pos_tag
>> s = "This is a simple sentence"
>> tokens = word_tokenize(s)
>> tokens_pos = pos_tag(tokens)
>> print(tokens_pos)
[('This', 'DT'), ('is', 'VBZ'), ('a', 'DT'),
('simple', 'JJ'), ('sentence', 'NN')]
```

Homogeneization

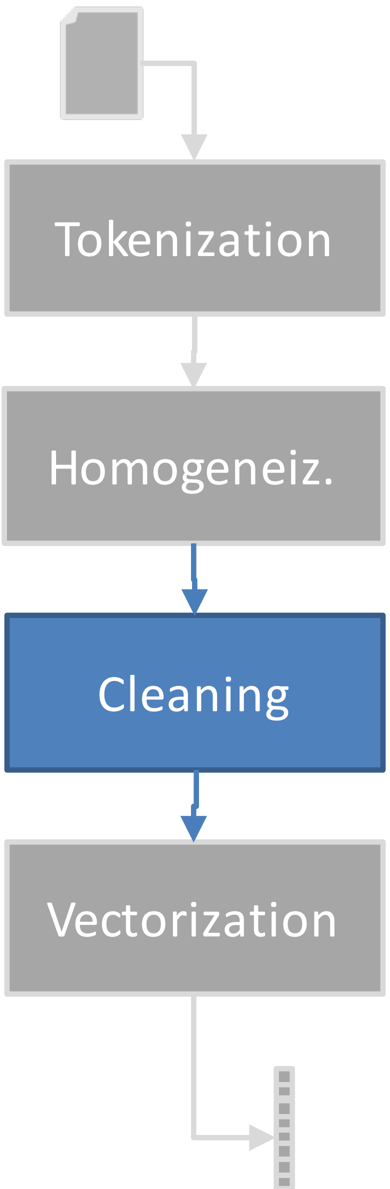


- *N*-grams
  - Some words tend to occur in groups
    - information processing, machine learning...
  - It can be useful that they are analyzed in groups
  - There are routines to detect them, but the easiest way is...
    - informationprocessing
    - machinelearning



Cleaning

- Removing least relevant words
  - Some words appear very often in all sorts of different contexts.
  - They are so frequent that they do not help to distinguish between different texts.
  - These words are called stop words.
  - The best option would be to remove them

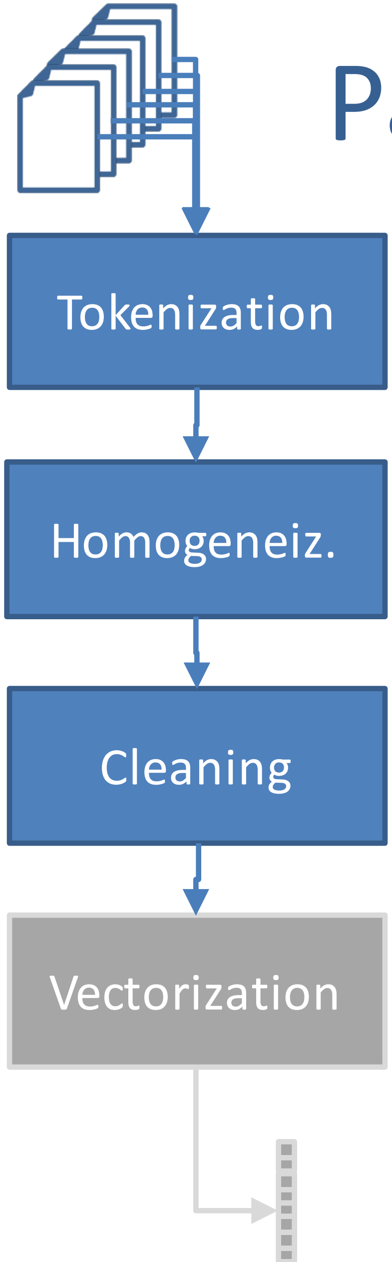


Cleaning

- Removing stopwords:

```
from nltk.corpus import stopwords
sw = stopwords.words('english')
clean_text = [word for word in document
                  if not word in sw]
```

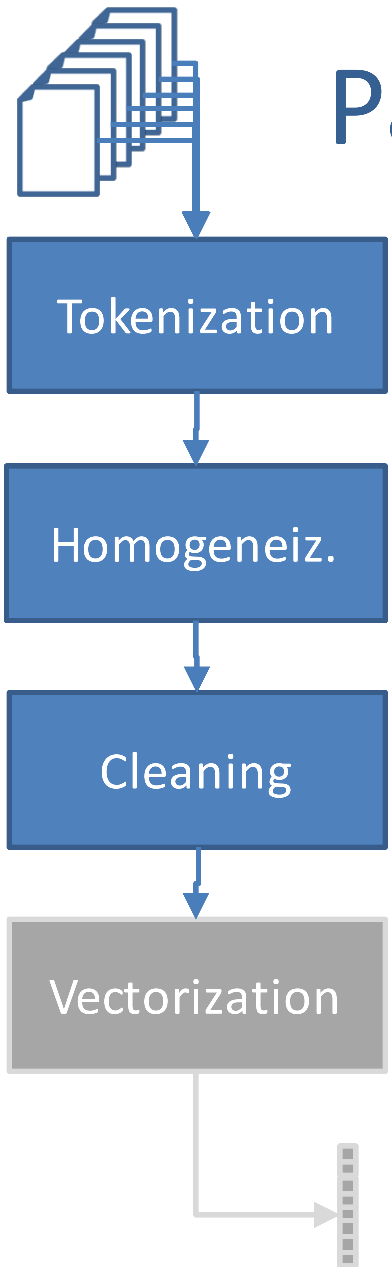
Parallel Document Processing



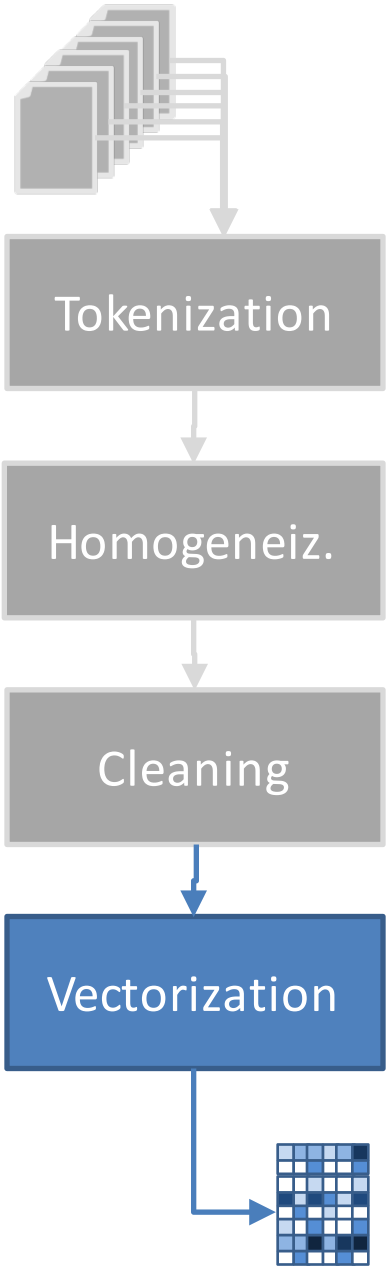
- Until now, we have worked with a single document
- Extend your code to work with all the documents of the corpus
- Create a list of text, where each row is a previously processed text

```
content = [
    [u'fulton', u'counti', u'grand', ..., u'said', u'friday']
    [u'austin', u'texa', u'committe', ..., u'price', u'abandon']
    ....
    [u'dear', u'sir', u'let', u'begin', ..., u'mind', u'address']
    ]
```

Parallel Document Processing

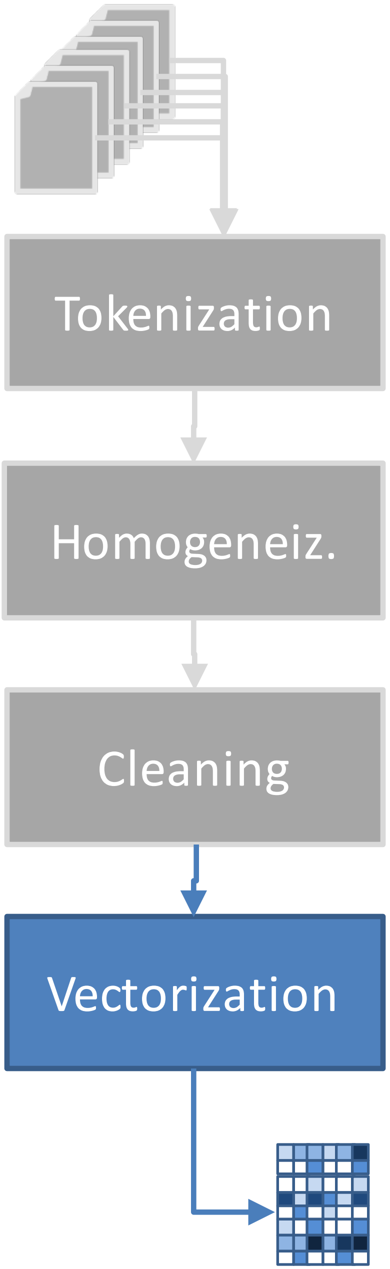



```
content = []
for text_name in corpus.fileids():
    path = nltk.data.find(
        'corpora/brown/'+text_name)
    f = open(path, 'rU')
    raw = f.read()
    # Here you can process your
    # raw text → clean_text
    content.append(clean_text)
    f.close()
```



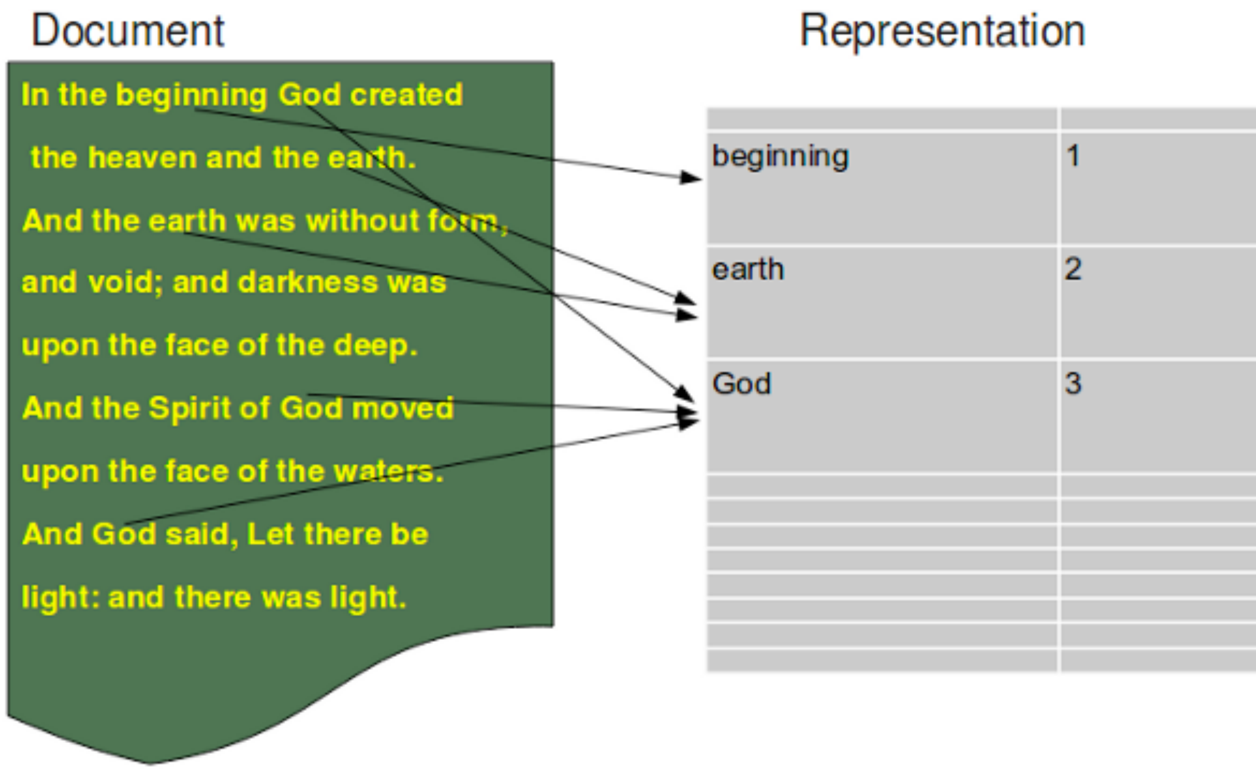
Vectorization

- Bag of words: counting words
  - ML algorithms process numbers, not words.
  - Only if we manage to transform text into meaningful numbers, we can feed it into ML algorithms
  - Bag-of-word approach: for each word in the document, count its occurrence and note it in a vector

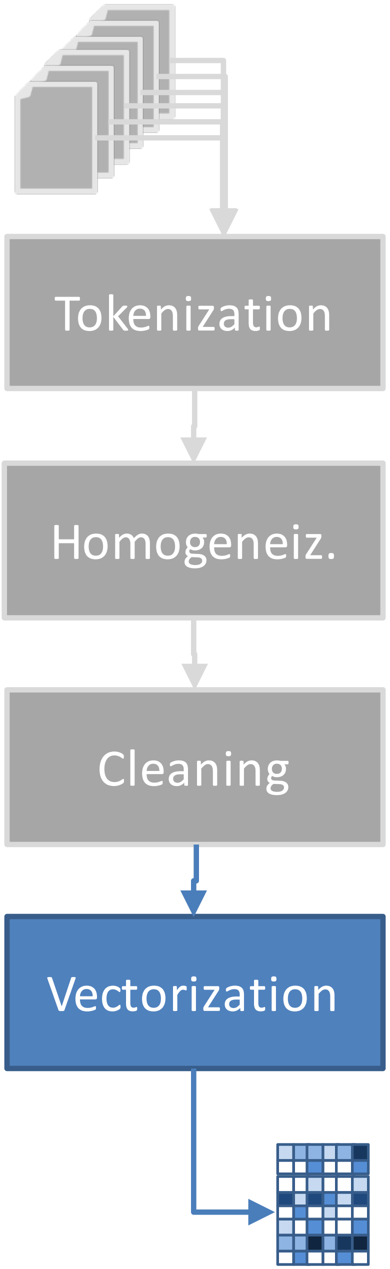


Vectorization

- Bag of words: counting words



Vectorization

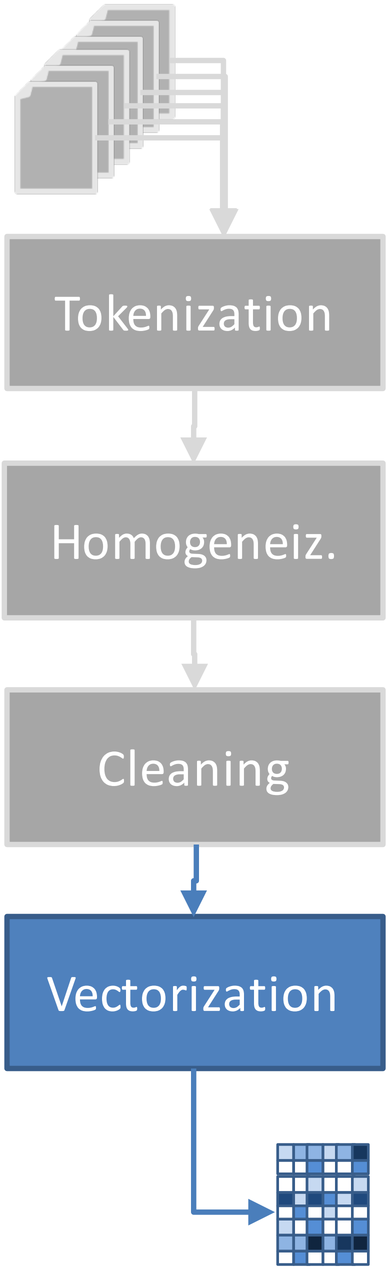


- Bag of words: counting words

[Hadoop, is, great, python, is, great]

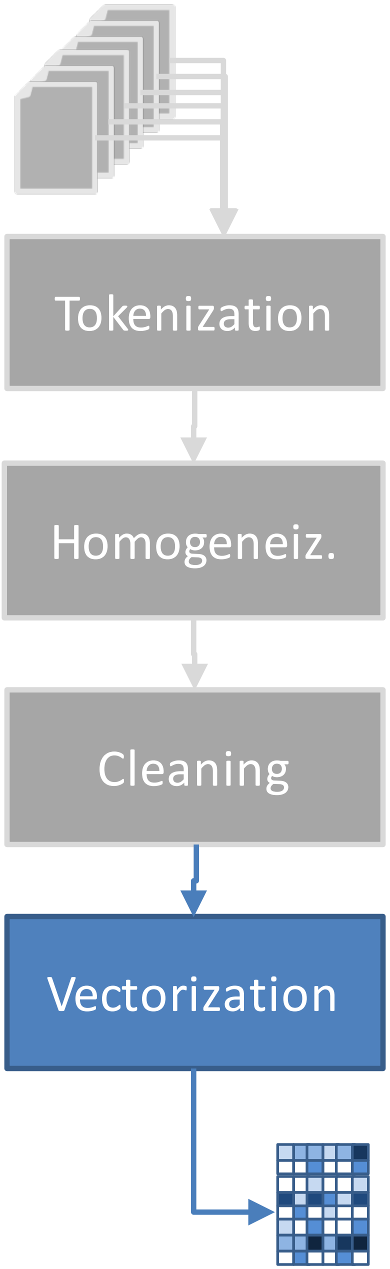
[(Hadoop, 1), (is, 2), (great, 2), (python, 1)]

[(12342, 1), (3423, 2), (5676, 2), (6768, 1)]



Vectorization

- Term frequency - Inverse document frequency (TF-IDF)
  - BoW: the feature values simply count occurrences of terms in a document.
  - High occurrence terms?? They appear in all documents → NON RELEVANT.
  - Low occurrence terms?? They appear in very few documents → RELEVANT
  - This can only be solved by:
    - counting term frequencies for each document
    - discounting those that appear in many posts



Vectorization

- 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
    - $\mathrm{TF}(w, d) = \frac{bow(w,d)}{\#\ words\ in\ doc}$
    - $\mathrm{IDF}(w, d) = \log \frac{\#\ docs}{\#\ docs\ with\ w}$
    - $\mathrm{TF{-}IDF}(w, d) = \mathrm{TF}(w, d) \times \mathrm{IDF}(w, d)$
  - IDF → 0 in common words & IDF increases in rare words