How to work with Text

Natural Language Processing

NLP = Natural Language Processing

- NLP is a field in machine learning with the ability of a computer to understand, analyze, manipulate, and potentially generate human language.

- Useful to all big data applications
- Especially useful for mining knowledge about people's behavior, attitude and opinions
- Express directly knowledge about our world: Small text data are also useful!

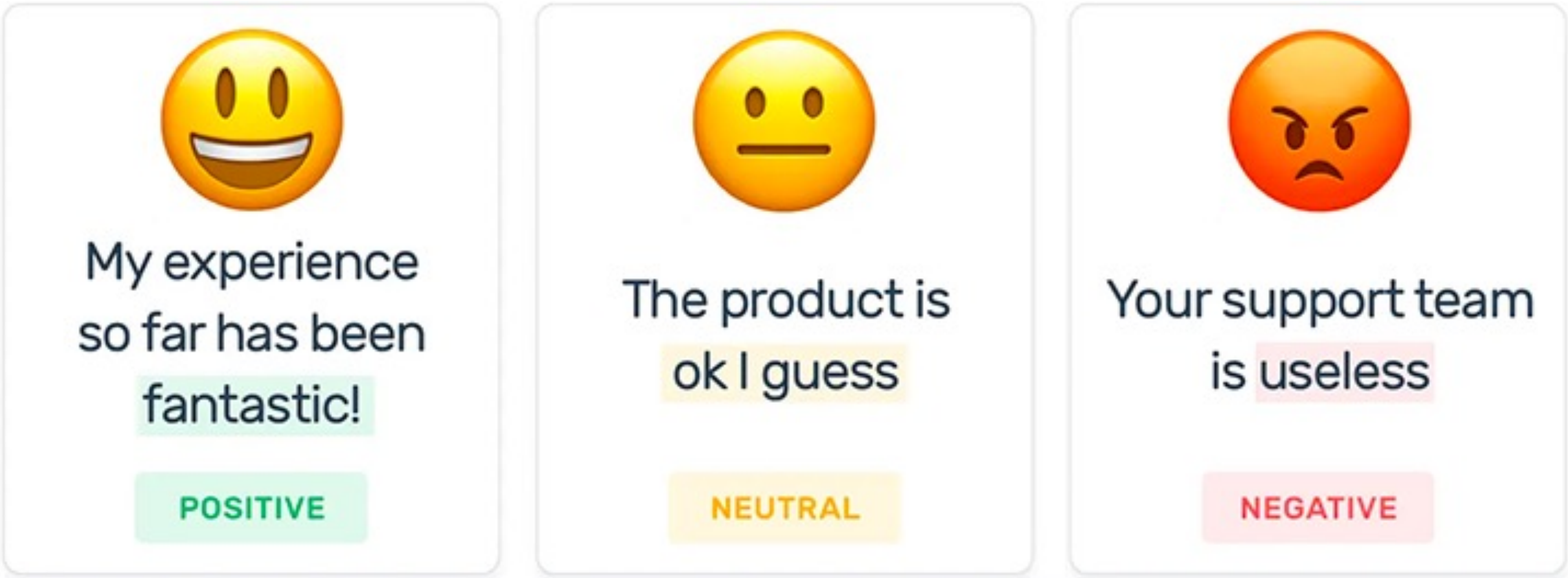
Main NLP Task

- Document classification
  - Associate a label to a document

- Sentiment analysis
  - Associate a label to a sentence

  - Sentiment analysis is the automated process of analyzing text data and classifying opinions as negative, positive or neutral. Usually, besides identifying the opinion, these systems extract attributes of the expression e.g.:
    - Polarity: if the speaker express a positive or negative opinion,
    - Subject: the thing that is being talked about,
    - Opinion holder: the person, or entity that expresses the opinion

Sentiment analysis

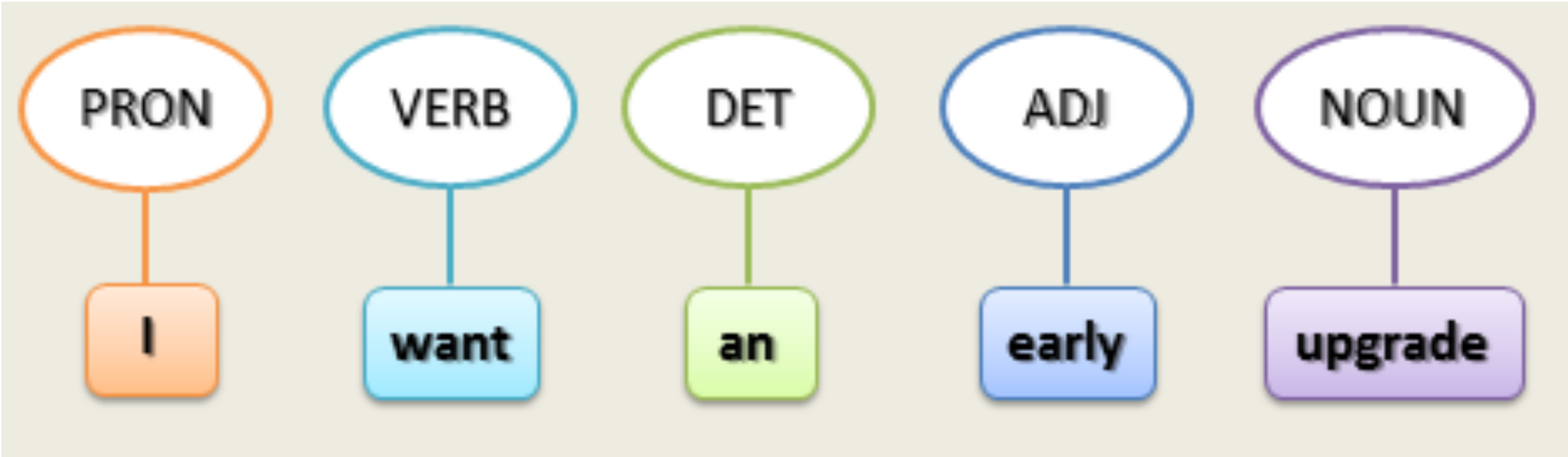


Main NLP Task

- Part of speech tagging
  - Associate a label to a word

  - Part of Speech Marking (POS), also known as grammatical marking or word category disambiguation, involves marking a word according to its relationship to adjacent and related words in a sentence, word group or paragraph.
  - A simplified form of this definition is commonly taught to school-age children in the identification of words such as nouns, verbs, adjectives, adverbs, and so on.

Part of speech tagging (POS)



Main NLP Task

- Naming entity recognition
  - Associate a label to a word

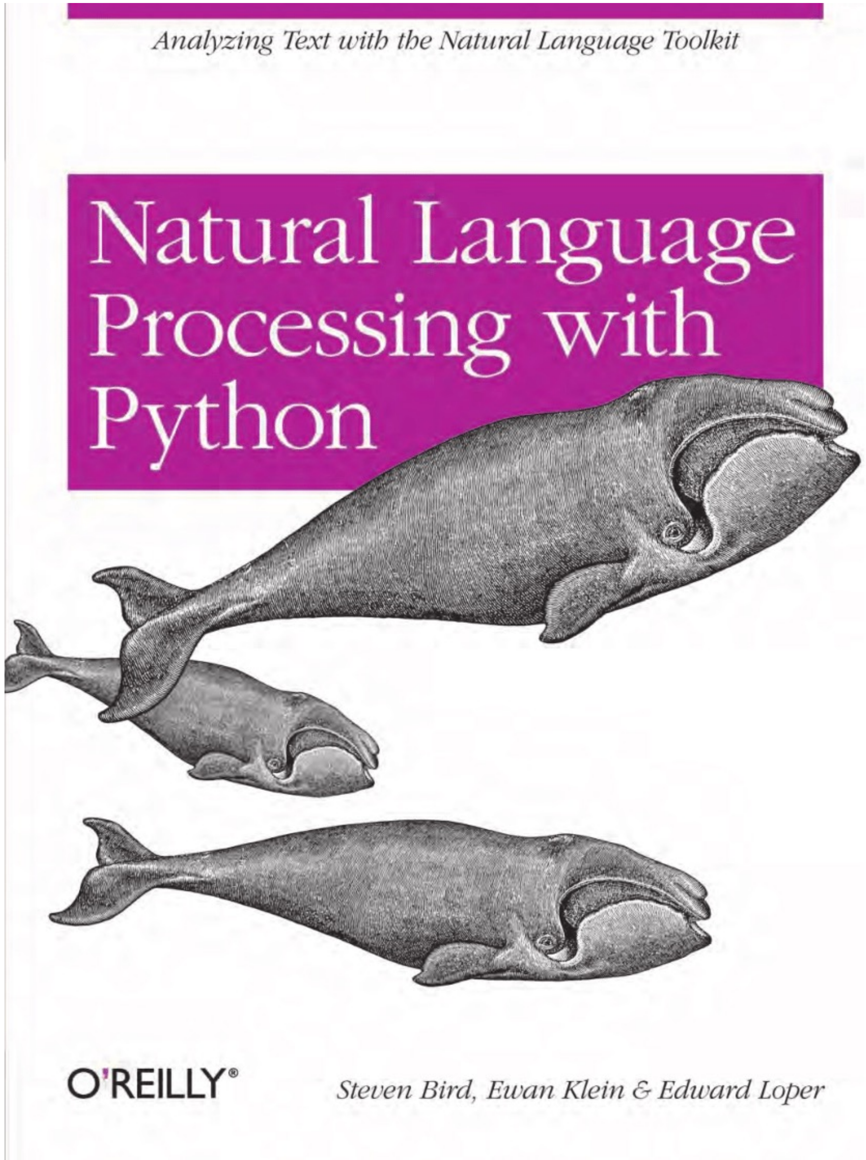
  - Named-entity recognition (NER) seeks to locate and classify named entities in text into pre-defined categories such as the names of persons, organizations, locations, expressions of times, quantities, monetary values, percentages, etc.
  - In a second step, we also try to extract relationships between previously recognized entities

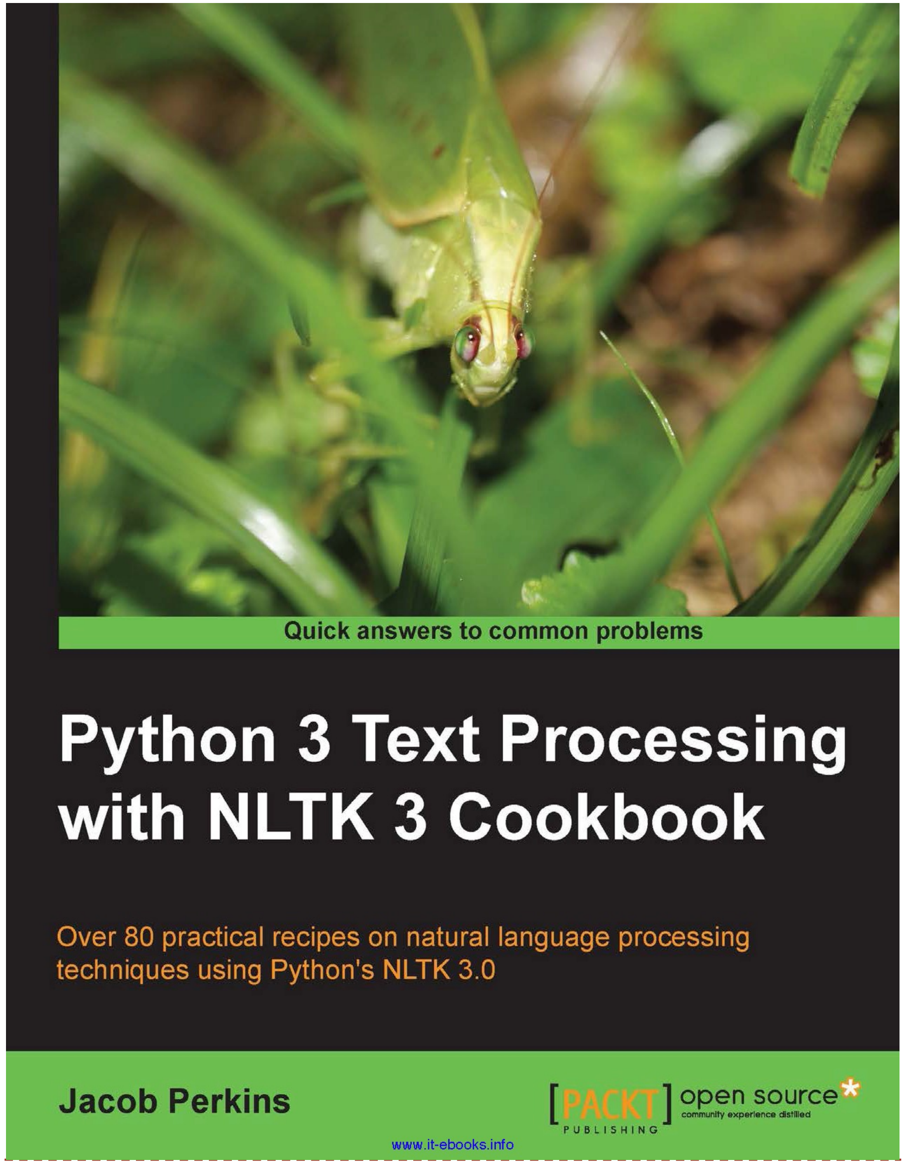
Naming Entity Recognition (NER)

In fact, the Chinese NORP market has the three CARDINAL most influential names of the retail and tech space – Alibaba GPE , Baidu ORG , and Tencent PERSON (collectively touted as BAT ORG), and is betting big in the global AI GPE in retail industry space . The three CARDINAL giants which are claimed to have a cut-throat competition with the U.S. GPE (in terms of resources and capital) are positioning themselves to become the 'future AI PERSON platforms'. The trio is also expanding in other Asian NORP countries and investing heavily in the U.S. GPE based AI GPE startups to leverage the power of AI GPE . Backed by such powerful initiatives and presence of these conglomerates, the market in APAC AI is forecast to be the fastest-growing one CARDINAL , with an anticipated CAGR PERSON of 45% PERCENT over 2018 - 2024 DATE .

To further elaborate on the geographical trends, North America LOC has procured more than 50% PERCENT of the global share in 2017 DATE and has been leading the regional landscape of AI GPE in the retail market. The U.S. GPE has a significant credit in the regional trends with over 65% PERCENT of investments (including M&As, private equity, and venture capital) in artificial intelligence technology. Additionally, the region is a huge hub for startups in tandem with the presence of tech titans, such as Google ORG , IBM ORG , and Microsoft ORG .

Two books…


Analyzing Text with the Natural Language Toolkit
Natural Language Processing with Python
O'REILLY
Steven Bird, Ewan Klein & Edward Loper


Quick answers to common problems
Python 3 Text Processing with NLTK 3 Cookbook
Over 80 practical recipes on natural language processing techniques using Python's NLTK 3.0
Jacob Perkins
[PACKT] open source
PUBLISHING
www.it-ebooks.info

And one main library

- Natural Language ToolKit (NLTK)
  - http://www.nltk.org/
  - A comprehensive Python library for natural language processing and text analytics
  - Originally designed for teaching
  - also adopted in the industry for research and development due to its usefulness and breadth of coverage
- NLTK is often used for rapid prototyping of text processing programs
- Demos of select NLTK functionality and production-ready APIs are available at http://text-processing.com
- In Python: use nltk library (http://www.nltk.org/book/)
  - `!pip install -U nltk` in the Jupyter Notebook
  - or `conda install -c conda-forge nltk`

The NLTK Pipeline

Main step for an NLP pipeline

- Text normalization
  - **The set of operation depend on the task**

- Main goal
  - Replace the list of chars (the original text) by a list of tokens
  - Normalize some representation :
    - Date:
      - I'd like to go to Paris tomorrow. I already went there last year between January 12 and January 15, but I didn't get to see much. 3 days wasn't enough.
    - Phone number:
      - 20355555 will be normalized as +4920355555 (E164) or 020355555 (national format)
    - Acronym: U.S.A. → USA
    - Negation: don't → do not
  - Try to reduce the vocabulary size
    - Use only lower character
    - Correct spelling
    - Remove punctuation and stop words
    - Lemmatization (or Stemming)
    - Replace by synonyms (semantic reduction)

NLTK Corpus

- Gutenberg corpus
  - Small selection of texts from the Project Gutenberg electronic text archive, which contains some 25,000 free electronic books
    - Represents established literature
  - nltk.corpus.gutenberg.fileids() identify the files included in the corpus
- Web and Chat Text
  - Contains discussion forum from a Firefox, conversations overheard in New York, the script of Pirates of the Carribea, personal advertisements, and wine reviews
    - Represents more informal language
- Brown Corpus
  - First million-word electronic corpus of English, created in 1961 at Brown University. Contains text from 500 categorized sources (news, editorial, and so on)
    - Represents domain language
- You have to choose the corpus in regards of the task

- All information at: https://www.nltk.org/book/ch02.html

Tokenization

- Tokenization: process of splitting a string into a list of pieces (tokens).
  - A token is a piece of whole
    - A char is a token in a word
    - A word is a token in a sentence
    - A sentence is a token in paragraph
- Token != Words
  - Tokens
    - Substrings
    - Only structural
    - Data
  - Words
    - Objects
    - Contains a 'sense'
    - Meaning
- Not always an easy task
  - ' ' between space ? One or two words
    - cats!
    - San Francisco

Python tokenization

- By default, work with english language
  - from nltk.tokenize import sent_tokenize
  - sent_tokenize(a_text)
    - Return a list of sentences

  - from nltk import word_tokenize
  - word_tokenize(a_text)
    - Return a list of word
    - Ponctuation is a word

- For other language, you have to load specific tokenizer
  - from nltk.data import load
  - spanish_tokenizer = load("tokenizers/punkt/PY3/spanish.pickle")
  - spanish_tokenizer.tokenize(a_text)
- Avalaible tokenizers are on ~/nltk_data/tokenizers/punkt/PY3

Text normalization

- Text normalization is the process of transforming text into a single canonical form
- Text normalization requires being aware of
  - What type of text is to be normalized
  - how it is to be processed afterwards
  - There is no all-purpose normalization procedure
- Easy part
  - Put the text in lower case
    - lower_text = text.lower()
      - Text → "This is the first sentence. A gallon of milk in the U.S. …"
      - Lower text → "this is the first sentence. a gallon of milk in the u.s. …"
  - Suppress '.' in acronym:
    - U.S.A → USA
- Difficult part
  - Phone number: +33 6 10 20 30 40 or +33.(0)6.10.20.30.40
  - Date: 11/01/2018 or 2018-01-11
  - Etc.

Correct misspelled words

- It's a really difficult task and there's no specific approach.
- Here are a few proposals, we will see more later.
  - Remove repeating character
    - I looove it
  - Spelling correction using distance between current word and a dictionary
  - Specific Neural Network (same approach than text translation)

Removing repeating character

- In every language, people are often not stricky grammatical
  - I looove it
  - class RepeatReplacer(object):
```
class RepeatReplacer(object):
    def __init__(self):
        self.repeat_regexp=re.compile(r'(\w*)(\w)\2(\w*)')
        self.repl=r'\1\2\3'
    def replace(self, word):
        if word in words.words ():      // if the word exist…
            return word
        repl_word=self.repeat_regexp.sub(self.repl, word)
        if repl_word!=word:
            return self.replace(repl_word)
        else:
            return repl_word
```
  - replacer=RepeatReplacer()
  - replacer.replace("loooove"), replacer.replace(”book")
    - ('love', 'book')

1. Determine if the word exists

- Use a pre-existing list of word
  - Dictionnary
    - Install enchant and PyEnchant for example
  - NLTK have a list of existing English word
    - From nltk.corpus import words
    - 'cooking' in words.words(), 'loove' in words.words()
      - (True, False)
- Use a corpus (One corpus, many corpora)
  - A text corpus is a large body of text
  - Many corpora are designed to contain a careful balance of material in one or more genres
  - Nltk comes from several corpus
    - Each corpus have a specific vocabulary list
    - To list all installing corpus in your machine
      - import os
      - os.listdir(nltk.data.find("corpora"))
        - […, 'brown', 'udhr2', 'webtext', …, 'inaugural', …, 'gutenberg', 'genesis', 'twitter_samples'

2. Find a suggestion list

- Use a dictionnary or a corpus
- Example of dictionary
  1. Install Enchant
     - http://www.abisource.com/projects/enchant
  2. Find dictionaries
     - http://aspell.net
  3. Install PyEnchant library
     - http://pythonhosted.org/pyenchant
     - Use easy_install command
- Use Enchant
  - spell_dict=enchant.Dict(dict_name)
  - spell_dict.check(word) → return True or False
  - spell_dict.suggest(word) → return a list of words

3. Choose a distance

- A distance must satisfy the following three requirements:
  - d(a, a) = 0
  - d(a, b) >= 0
  - d(a, c) <= d(a, b) + d(b, c)
- Import distance
  - From nltk.metrics import <distance_name>
- Edit distance (Levenshtein)
  - The edit distance is the number of characters that need to be substituted, inserted, or deleted, to transform s1 into s2
  - edit_distance('langage', 'language')
    - 1
  - Allows specifying the cost of substitution edits (e.g., "a" -> "b"), because sometimes it makes sense to assign greater penalties to substitutions.
    - *substitution_cost=1 (default)*
  - Allows transposition edits (e.g., "ab" -> "ba")
    - *transpositions=False (default)*

Stop word removal

- Stopwords are common words that generally do not contribute to the meaning of a sentence
  - Examples: the, as, a
- Most search engines will filter out stopwords fom search queries in order to save space in their index
- NLTK comes with a stopword corpus
  - from nltk.corpus import stopwords
  - stopwords.words('english')
    - ['i', 'me', 'my', 'myself', 'we', 'our', …
  - stopwords.words('french')
    - ['au', 'aux', 'avec', 'ce', 'ces', …
- General use
  - tokens = word_tokenize(text)
    - ['this', 'is', 'the', 'first', 'sentence', '.', 'a', …
  - [t for t in tokens if t not in english_stopwords]
    - ['first', 'sentence', '.',

Some experiment with stop word removal

- from nltk.corpus import stopwords
- print(stopwords.words('english'))

['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', **'not'**, 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', **'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"**]

Some experiment with stop word removal

- Let's imagine you are asked to create a model that does sentiment analysis of product reviews. The dataset is fairly small that you label it your self. Consider a few reviews from the dataset.
  1. *The product is really very good. — POSITIVE*
  2. *The products seems to be good. — POSITIVE*
  3. *Good product. I really liked it. — POSITIVE*
  4. *I didn't like the product. — NEGATIVE*
  5. *The product is not good. — NEGATIVE*
- You performed preprocessing on data and removed all stopwords. Now, let us look what happens to the sample we selected above.
  1. *product really good. — POSITIVE*
  2. *products seems good. — POSITIVE*
  3. *Good product. really liked. — POSITIVE*
  4. *like product. — NEGATIVE ?*
  5. *product good. — NEGATIVE ?*
- Scary, right?

Reduce word forms
Stemming and Lemmatisation

Stemming

- The term "stem" generally refers to a crude heuristic process that cuts off the end of words in the hope of achieving a reduction in the forms of a word
- This often results in the removal of suffixes and sometimes prefixes
  - cats, cat → cat
  - looked → look
- May result in an unknown word

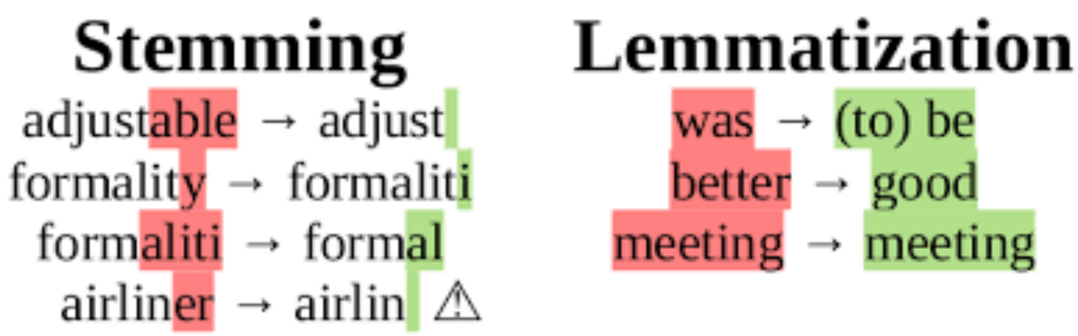
Lemmatization

- Lemmatization generally involves doing things correctly using vocabulary and morphological analysis of words
- Reduce inflections or variant forms to base form
  - am, are, is → be
  - Jack's → Jack
- Always ends up with a known word

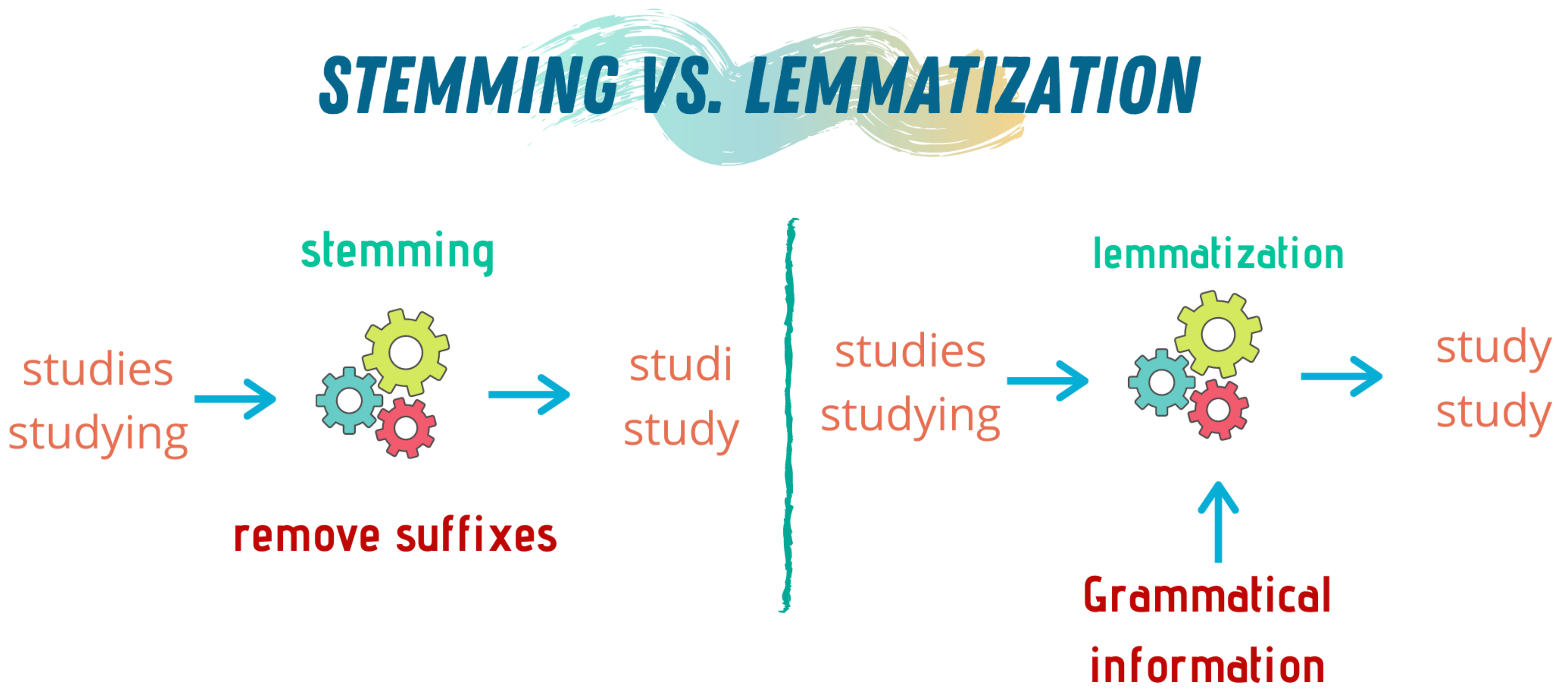
Reduce word forms
Stem and Lem in Python

- Stemming: many algorythms
  - For example, you can use Porter algorithm
    - porter = nltk.PorterStemmer()
    - stemming_form = porter.stem(token)

- Lemmatization:
  - WNlemma = nltk.WordNetLemmatizer()
  - Lemma_form = WNlemma.lemmatize(token)



Stem vs Lem


STEMMING VS. LEMMATIZATION
stemming
studies
studying
studi
study
remove suffixes
lemmatization
studies
studying
study
study
Grammatical
information

Summary
Text normalization in Python

Tokenization

- Usually depends on the language, sometimes on the task
  - tokens = nltk.word_tokenize(sentence)
  - sentences = nltk.sent_tokenize(paragraph)

Normalization

- Use only one form: lowercase for example
  - lower_text = text.lower()

Reduce vocabulary

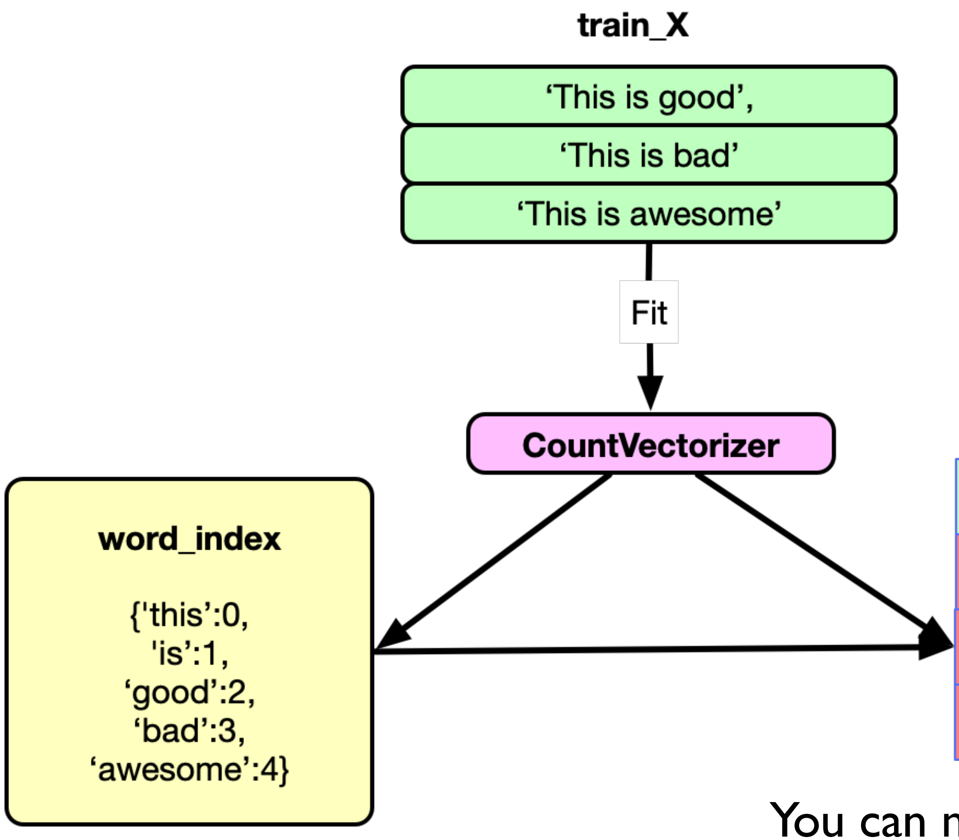
- Stop word removal
  - from nltk.corpus import stopwords
  - tokens = [t for t in tokens if t not in stopwords.words('english')]

- Stemming: user Porter algorithm - Several algorithms available
  - porter = nltk.PorterStemmer()
  - tokens = [porter.stem(t) for t in tokens]

- Lemmatization - Several algorithms available
  - WNlemma = nltk.WordNetLemmatizer()
  - tokens = [WNlemma.lemmatize(t) for t in tokens]

Features extraction

Bag of Word



Features

This	is	good	bad	awesome
1	1	1	0	0
1	1	0	1	0
1	1	0	0	1

You can mark the presence or absence of a word
Or
You can count the number of times it appears.

TF-IDF

train_X

'This is good and awesome',
'This is bad'

Fit

TFIDFVectorizer

Term Frequency
No of time word appears/No of total terms in Document

This	is	good	bad	awesome	and
1/5	1/5	1/5	0	1/5	1/5
1/3	1/3	0	1/3	0	0

Inverse Document Frequency
-log(ratio of documents that include the word)

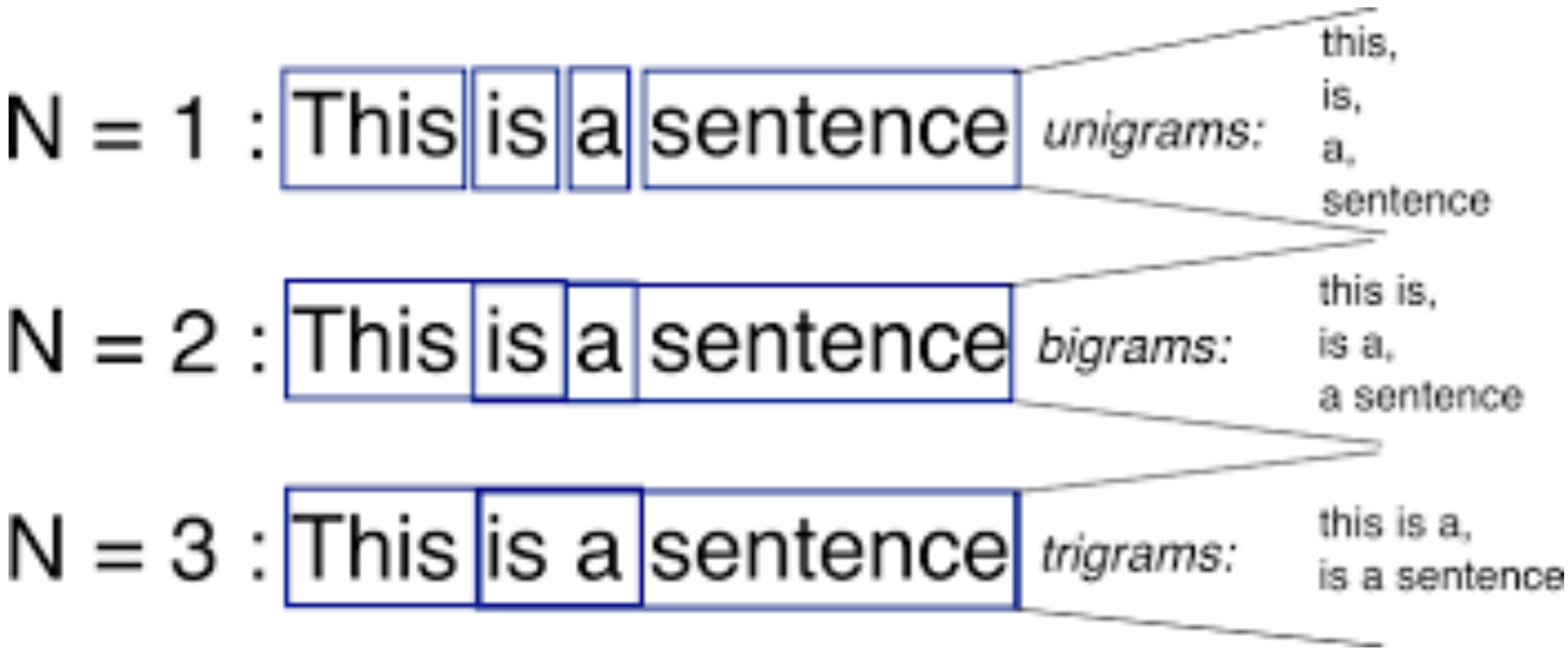
This	is	good	bad	awesome	and
log(2/2)	log(2/2)	log(2/1)	log(2/1)	log(2/1)	log(2/1)

x

Features

This	is	good	bad	awesome	and
0	0	1/5*log(2/1)	0	1/5*log(2/1)	1/5*log(2/1)
0	0	0	1/3*log(2/1)	0	0

N-gram



N = 1 : This is a sentence
unigrams:
this,
is,
a,
sentence
N = 2 : This is a sentence
bigrams:
this is,
is a,
a sentence
N = 3 : This is a sentence
trigrams:
this is a,
is a sentence

N-grams

- *N*-gram of size
  - 1 is referred to as a "unigram";
  - 2 is a "bigram" (or, less commonly, a "digram
  - 3 is a "trigram"
- N-grams in NKTK:
  - from nltk import ngrams
  - For the word "hello"
    - set(ngrams("hello", 2))
      - {('e', 'l'), ('h', 'e'), ('l', 'l'), ('l', 'o')}
  - For the sentence "The cow jumps over the moon"
    - set(ngrams(nltk.word_tokenize("The cow jumps over the moon"), 2))
      - {('The', 'cow'), ('cow', 'jumps'), ('jumps', 'over'), ('over', 'the'), ('the', 'moon')})
- N-grams is also included in
  - sklearn.feature_extraction.text.CountVectorizer
  - sklearn.feature_extraction.text.TfidfVectorizer
  - tf.keras.layers.TextVectorization

Lab

- Build an imbalanced dataset by grouping classes 2 to 4 together and leaving class 1 alone
- Include in a sklearn pipeline an NLP preprocessing and a BOW/DTM representation
  - Use a stemming or lemming step
  - Use TfidfVectorizer
- Observe the impact of the preprocessing for sentiment analysis task with a Logistic Regression Classifier
  - Try different hyperparameter values
    1. binary and use_idf (combination of this two parameters)
    2. lowercase
    3. stop_words (try None, 'english' and your own list)
    4. ngram_range (try (1,1) and (3,3) and (1,3))
    5. max_features (try None, 100, 1000)
    6. max_df (try 1.0, 0.5) and min_df (try 1, 50)
    7. Build you how preprocessor with stemming or lemming
- Evaluate your model
  - ACC / Recall / Precision / F1
  - ROC-AUC curves