

Introduction to Machine Learning: **How to work with Text**

Michel RIVEILL

Michel.Riveill@univ-cotedazur.fr

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!

Natural Language Processing

A lot of applications

- ▶ Information Retrieval ([Google](#) finds relevant and similar results).
- ▶ Information Extraction ([Gmail](#) structures events from emails).
- ▶ Machine Translation ([Google Translate](#) translates language from one language to another).
- ▶ Text Simplification ([Rewordify](#) simplifies the meaning of sentences). Shashi Tharoor tweets could be used (pun intended).
- ▶ Sentiment Analysis ([Hater News](#) gives us the sentiment of the user).
- ▶ Text Summarization ([Smmry](#) or Reddit's [autotldr](#) gives a summary of sentences).
- ▶ Spam Filter (Gmail filters spam emails separately).
- ▶ Auto-Predict (Google Search predicts user search results).
- ▶ Auto-Correct (Google Keyboard and [Grammarly](#) correct words otherwise spelled wrong).
- ▶ Speech Recognition (Google [WebSpeech](#) or [Vocalware](#)).
- ▶ Question Answering (IBM Watson's answers to [a query](#)).
- ▶ Natural Language Generation (Generation of text from image or video [data](#).)

In Python: use nltk library (<http://www.nltk.org/book/>)

- ▶ `!pip install nltk` in the Jupyter Notebook
- ▶ `or conda install -c conda-forge nltk`

Main NLP Task

▶ Sentence level : Sentiment analysis

- ▶ 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

▶ Word level : Naming entity recognition

- ▶ 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

Sentiment analysis



My experience
so far has been
fantastic!

POSITIVE



The product is
ok I guess

NEUTRAL



Your support team
is useless

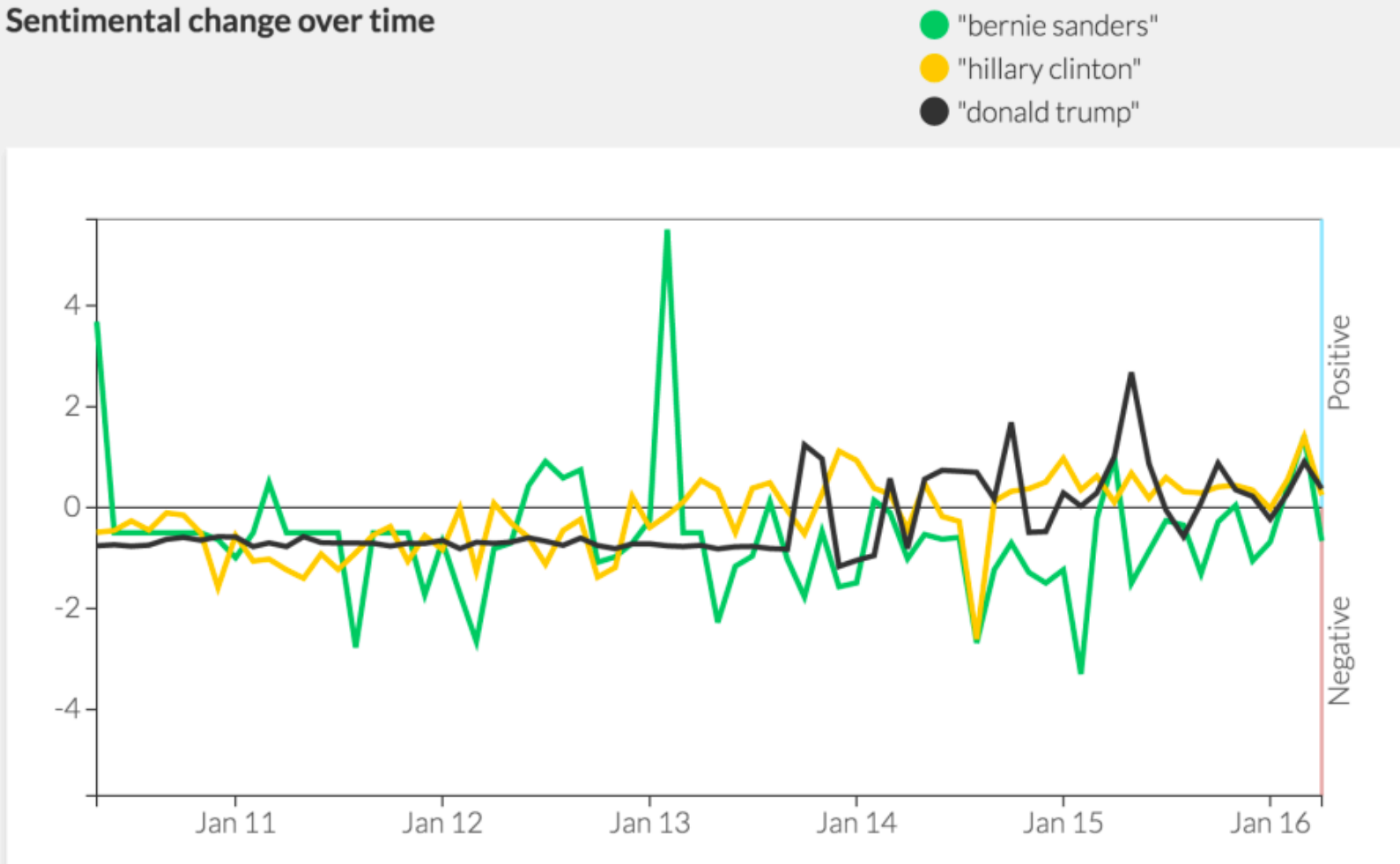
NEGATIVE

Sentiment analysis



Sentiment analysis

Sentimental change over time



Sentiment analysis

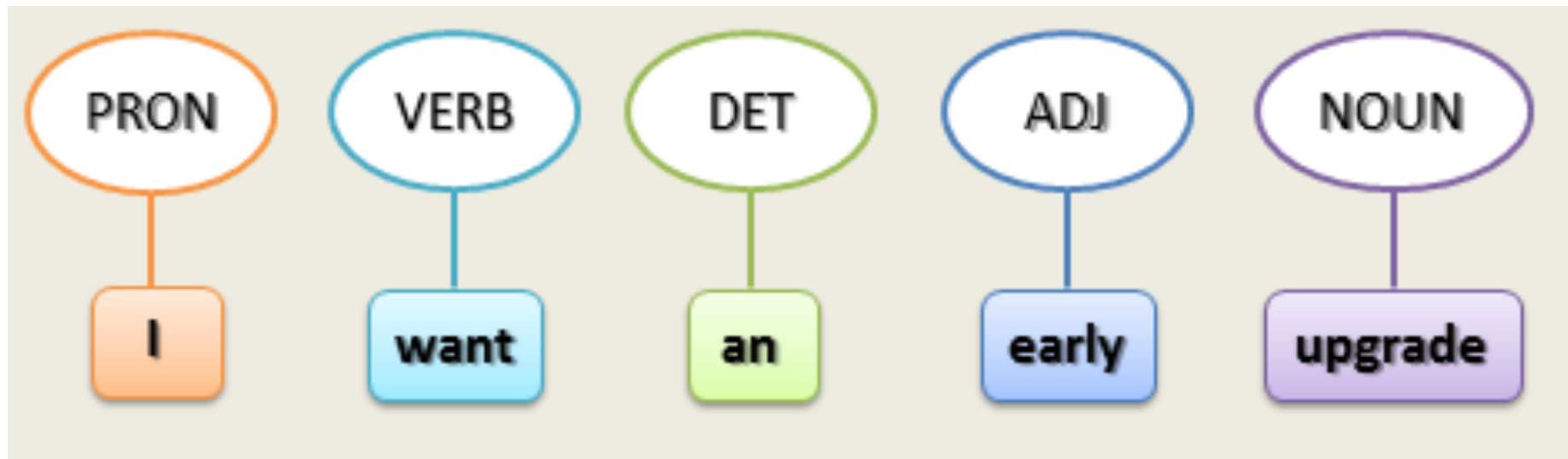
► https://www.csc2.ncsu.edu/faculty/healey/tweet_viz/



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

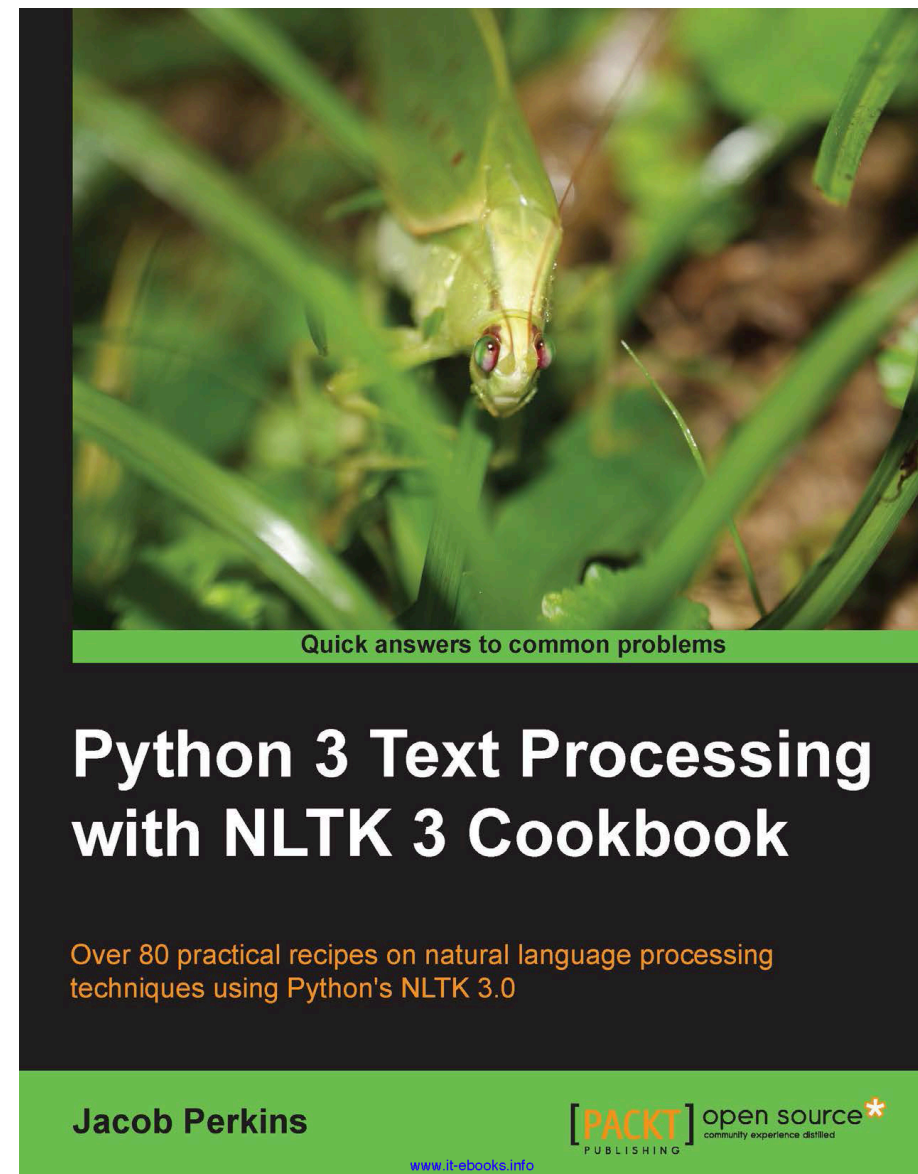
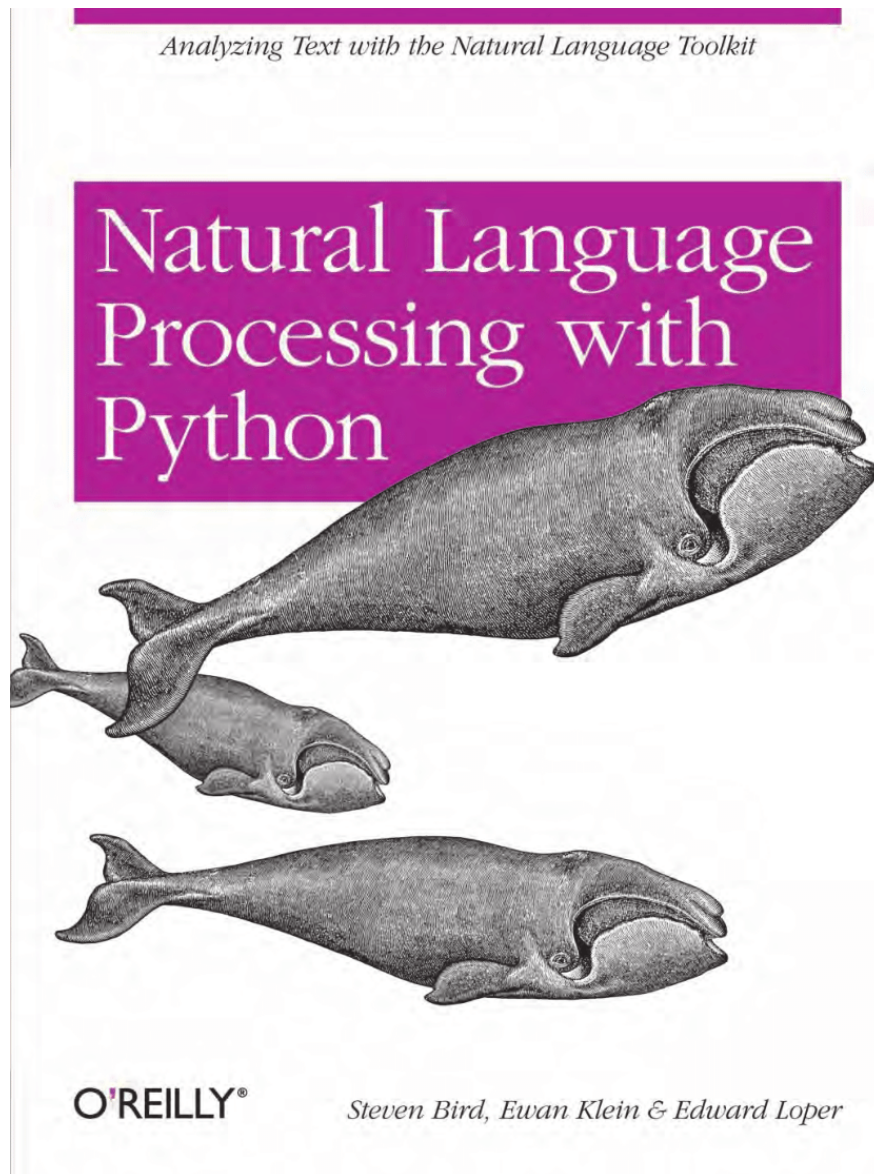
- ▶ Naming entity recognition
 - ▶ Associate a label to a word
- ▶ Named-entity recognition (NER) seeks to locate and classify named entities in text into pre-defined categories such as the names of persons, organizations, locations, expressions of times, quantities, monetary values, percentages, etc.
- ▶ 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...

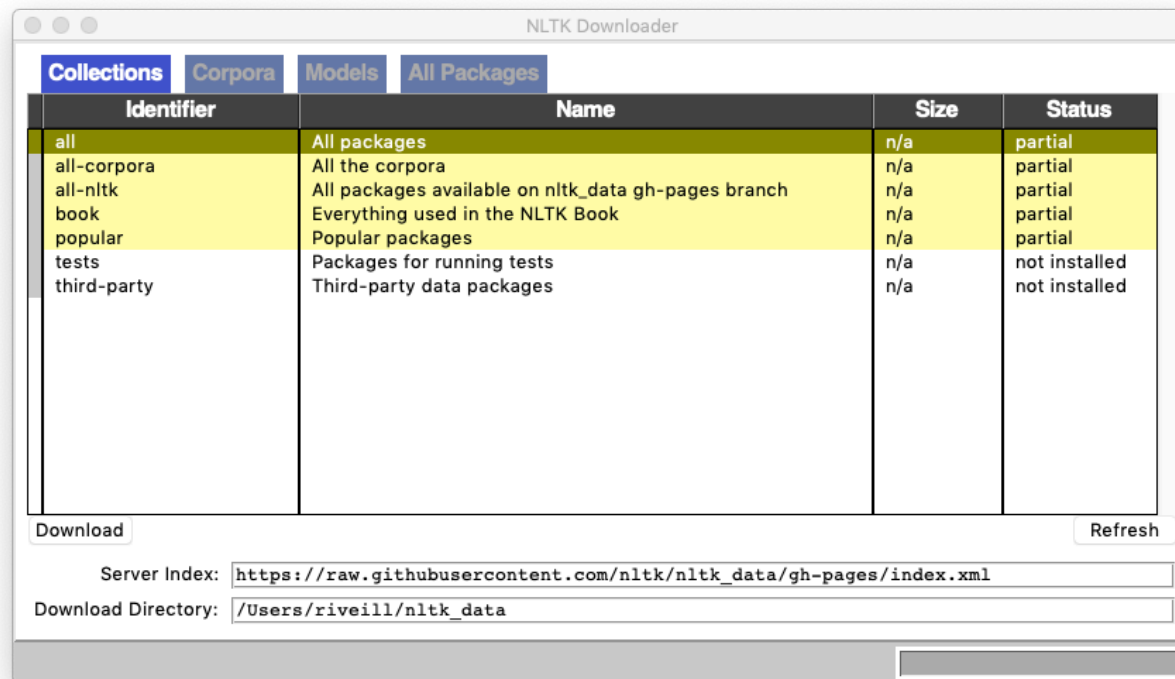


And one main library

- ▶ 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 nltk` in the Jupyter Notebook
 - ▶ `or conda install -c conda-forge nltk`

The NLTK library

- ▶ After installing NLTK, manage the packages
 - ▶ Import nltk
 - ▶ `nltk.download()`



Main step for an NLP pipeline

- ▶ **Normalize a text**
 - ▶ Tokenization
 - ▶ Spelling Correction
 - ▶ Lemmatization (or Stemming)
 - ▶ Find equivalence classes (using thesauri, e.g. WordNet) (semantic stuff)
- ▶ **Build statistical language models**
- ▶ **In order to work with text, we need some ‘corpus’**
 - ▶ You can use any piece of text
 - ▶ For example: all wikipedia articles, all documents published on PubMed
 - ▶ Nltk provides some corpus

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>

Text normalization

- ▶ Some sentences
 - ▶ Jack's cat in the hat is different from other cats!
 - ▶ They lay back on the San Francisco (U.S.A) grass and looked at the stars
- ▶ Every NLP task needs to do text normalization:
 - ▶ Tokenization:
 - ▶ process of splitting a string into a list of pieces (tokens)
 - ▶ Correct misspelled word:
 - ▶ Normalization:
 - ▶ Text normalization is the process of transforming text into a single canonical form
- ▶ Correct and reduce the vocabulary
 - ▶ Remove stop word
 - ▶ Stemming:
 - ▶ Crude heuristic process that cuts off the end of words in the hope of achieving a reduction in the forms of a word.
 - ▶ Lemmatisation:
 - ▶ Lemmatization generally involves doing things correctly using vocabulary and morphological analysis of words in order to reduce inflections or variant forms to base form

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'
- ▶ Not always an easy task ➤ Meaning
 - ▶ “ 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
 - ▶ Punctuation 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)`
- ▶ Available tokenizers are on `~/nltk_data/tokenizers/punkt/PY3`

Python separating contractions

- ▶ Default word tokenization
 - ▶ `word_tokenize("can't is a contraction.")`
 - ▶ `['ca', 'n't', 'is', 'a', 'contraction', '.']`
- ▶ If this convention is unacceptable, use another tokenizer
 - ▶ `from nltk.tokenize import WordPunctTokenizer`
 - ▶ `WordPunctTokenizer().tokenize("can't is a contraction.")`
 - ▶ `['can', "'", 't', 'is', 'a', 'contraction', '.']`
- ▶ Or use `RegexpTokenizer`
 - ▶ `from nltk.tokenize import RegexpTokenizer`
 - ▶ `RegexpTokenizer("[\w]+").tokenize("can't is a contraction.")`
 - ▶ `["can't", 'is', 'a', 'contraction']`
 - ▶ Treat contraction as whole words – suppress punctuation sign

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

Removing repeating character

- ▶ In every language, people are often not stricky grammatical

- ▶ I looove it

- ▶

```
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')
```


Spelling correction using distance

► Generic algorithm

```
► class SpellingReplacer(object):  
    def __init__(self, max_dist=2, distance=edit_distance):  
        self.max_dist=max_dist  
        self.distance = distance  
    def replace(self, word):  
        if word exist:  
            return word  
        suggestions=find_suggestion(word)  
        find the suggested_word with  
            the min(self.distance(word, suggestions))  
        return suggested_word
```

► We have to resolve

1. determine if the word exists
2. find a suggestion list
3. choose a distance to take the nearest suggestion

1. Determine if the word exists

- ▶ Use a pre-existing list of word
 - ▶ Dictionary
 - ▶ 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 dictionary 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:
 - ▶ $d(a, a) = 0$
 - ▶ $d(a, b) \geq 0$
 - ▶ $d(a, c) \leq d(a, b) + d(b, c)$
- ▶ 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 s_1 into s_2
 - ▶ `edit_distance('langage', 'language')`
 - ▶ `l`
 - ▶ 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)

Reduce word form

Stemming and Lemmatization

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



Stem and Lem in Python

- ▶ Stemming: many algorithms
 - ▶ For example, you can use Porter algorithm
 - ▶ `porter = nltk.PorterStemmer()`
 - ▶ `stemming_form = porter.stem(token)`
- ▶ Lemmatization:
 - ▶ `WNlemma = nltk.WordNetLemmatizer()`
 - ▶ `Lemma_form = WNlemma.lemmatize(token)`

Stemming

adjustable	→	adjust
formality	→	formaliti
formaliti	→	formal
airliner	→	airlin 

Lemmatization

was	→	(to) be
better	→	good
meeting	→	meeting

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 from search queries in order to save space in their index
- ▶ NLTK comes with a stopwords 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?

Some experiment with stop word removal

- ▶ If you are working
 - ▶ with basic NLP techniques like BOW, Count Vectorizer or TF-IDF
 - ▶ removing keywords is a good idea because keywords act as noise for these methods.
 - ▶ with LSTMs or other models that capture semantic meaning and the meaning of a word depends on the context of the previous text
 - ▶ it becomes important not to delete empty words.
- ▶ The goal of the `nlpprocess` Python package is to do a smarter cleanup of text by removing unnecessary words but keeping negatives.
 - ▶ `from nlpprocess import NLP`
 - ▶ `import pandas as pd`

 - ▶ `nlp = NLP()`
 - ▶ `df = pd.read_csv('some_file.csv')`
 - ▶ `df['text'] = df['text'].apply(nlp.process)`
- ▶ Now, if you utilize this package to preprocess the above samples we'll get something like this
 1. *product really very good. — POSITIVE*
 2. *products seems good. — POSITIVE*
 3. *good product. really liked. — POSITIVE*
 4. *not like product. — NEGATIVE*
 5. *product not good. — NEGATIVE*

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

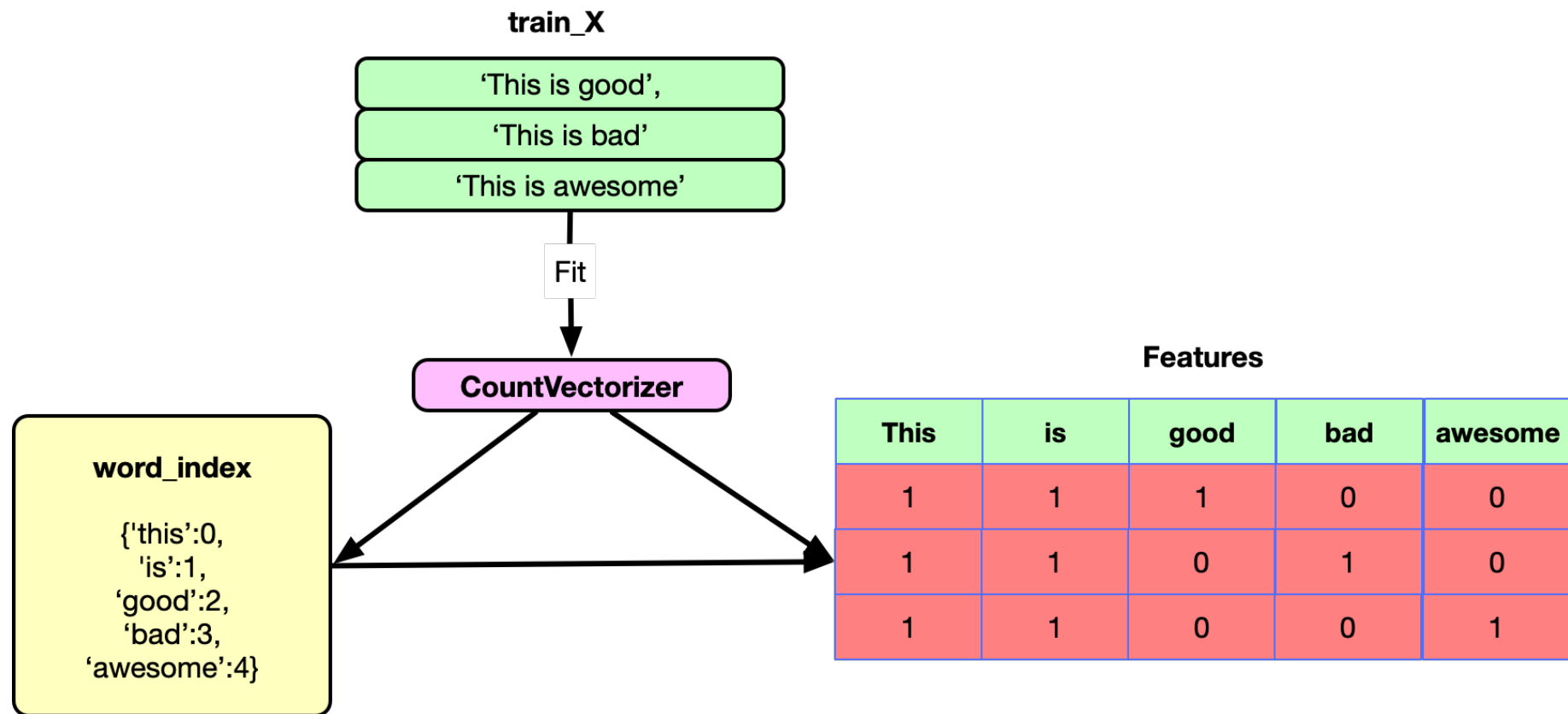
- ▶ Stemming: use Porter algorithm - Several algorithms available
 - ▶ `porter = nltk.PorterStemmer()`
 - ▶ `stemming_form = porter.stem(token)`
- ▶ Lemmatization - Several algorithms available
 - ▶ `WNlemma = nltk.WordNetLemmatizer()`
 - ▶ `Lemma_form = WNlemma.lemmatize(token)`
- ▶ Stop word removal
 - ▶ Use stop word list from Wikipedia and nltk library



Features extraction

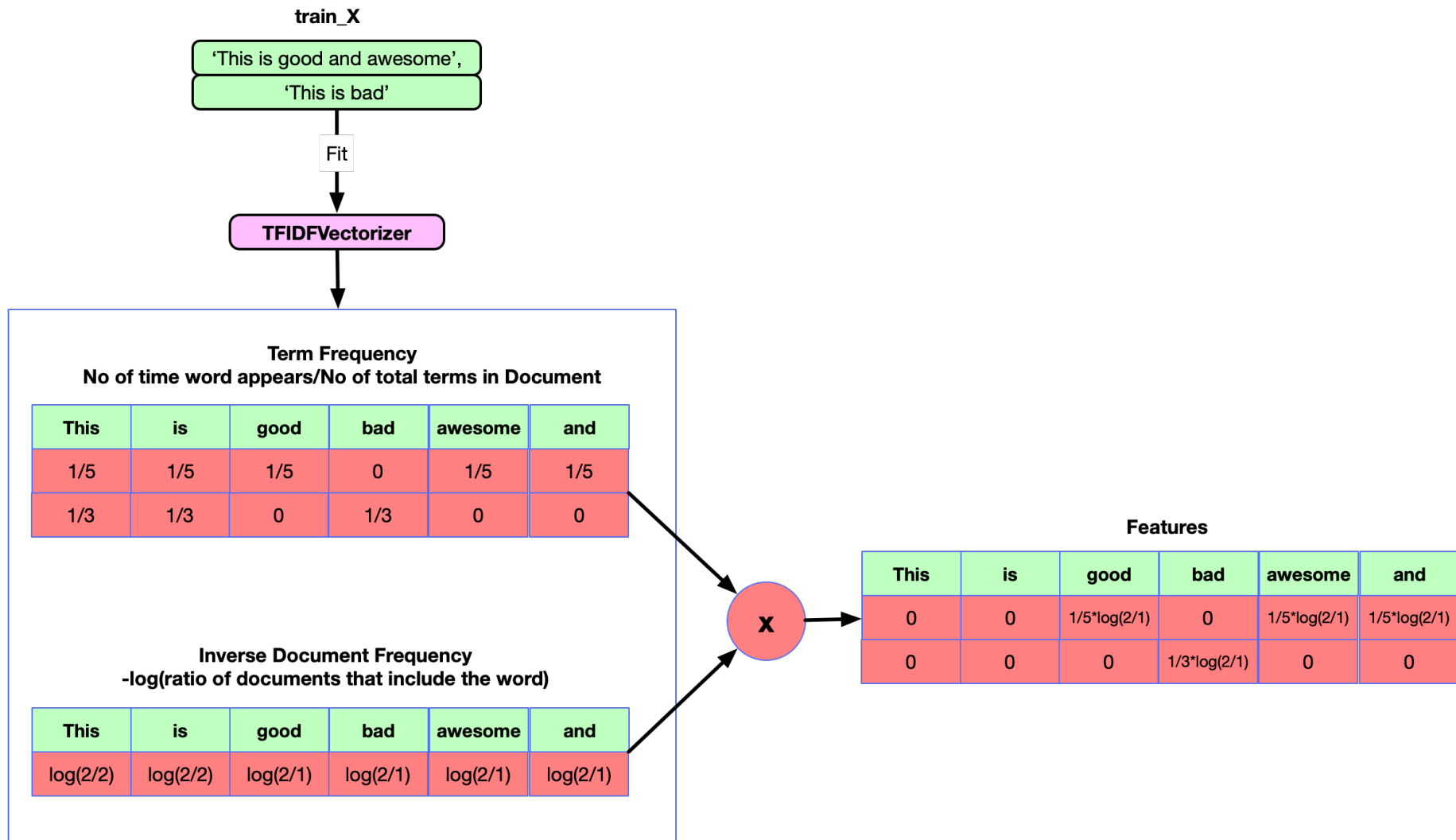


Bag of Word

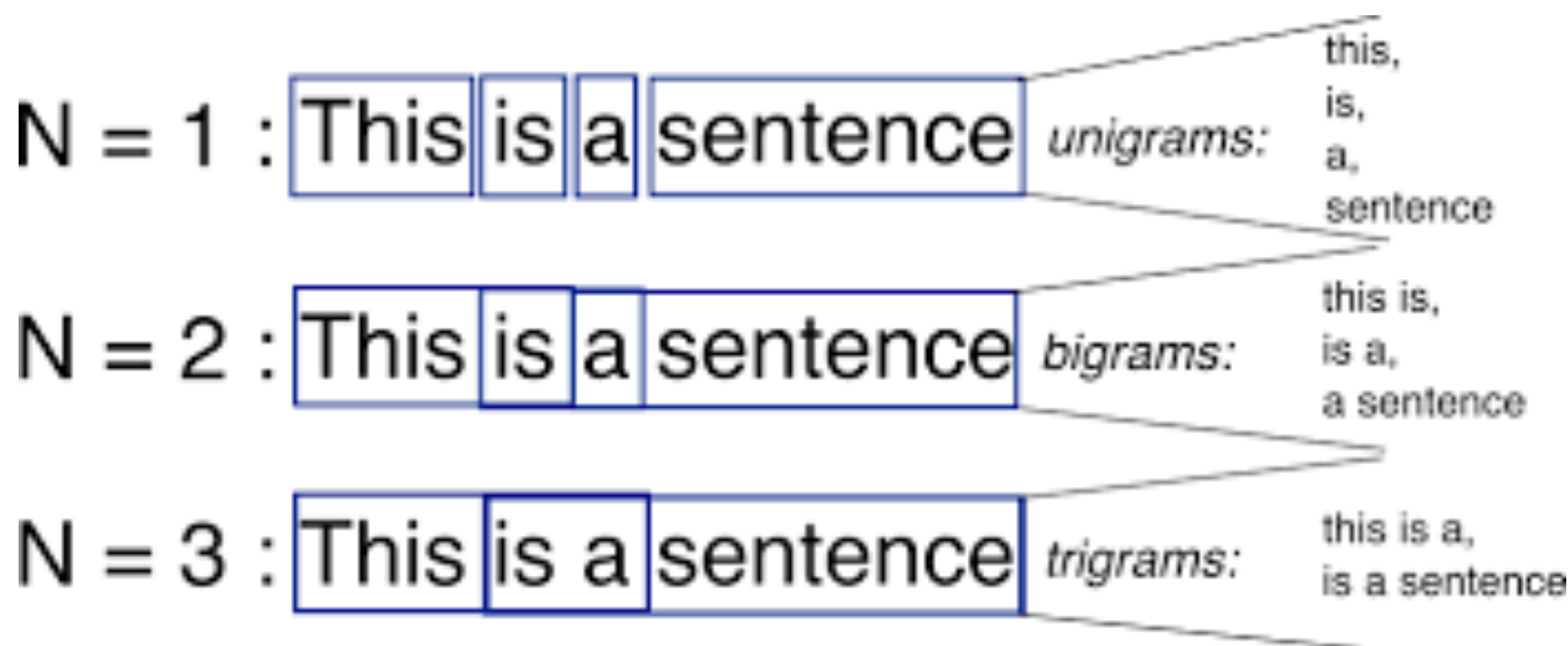


Instead of indicating the presence or absence of a word, it is also possible to count the number of times it appears.

TF-IDF



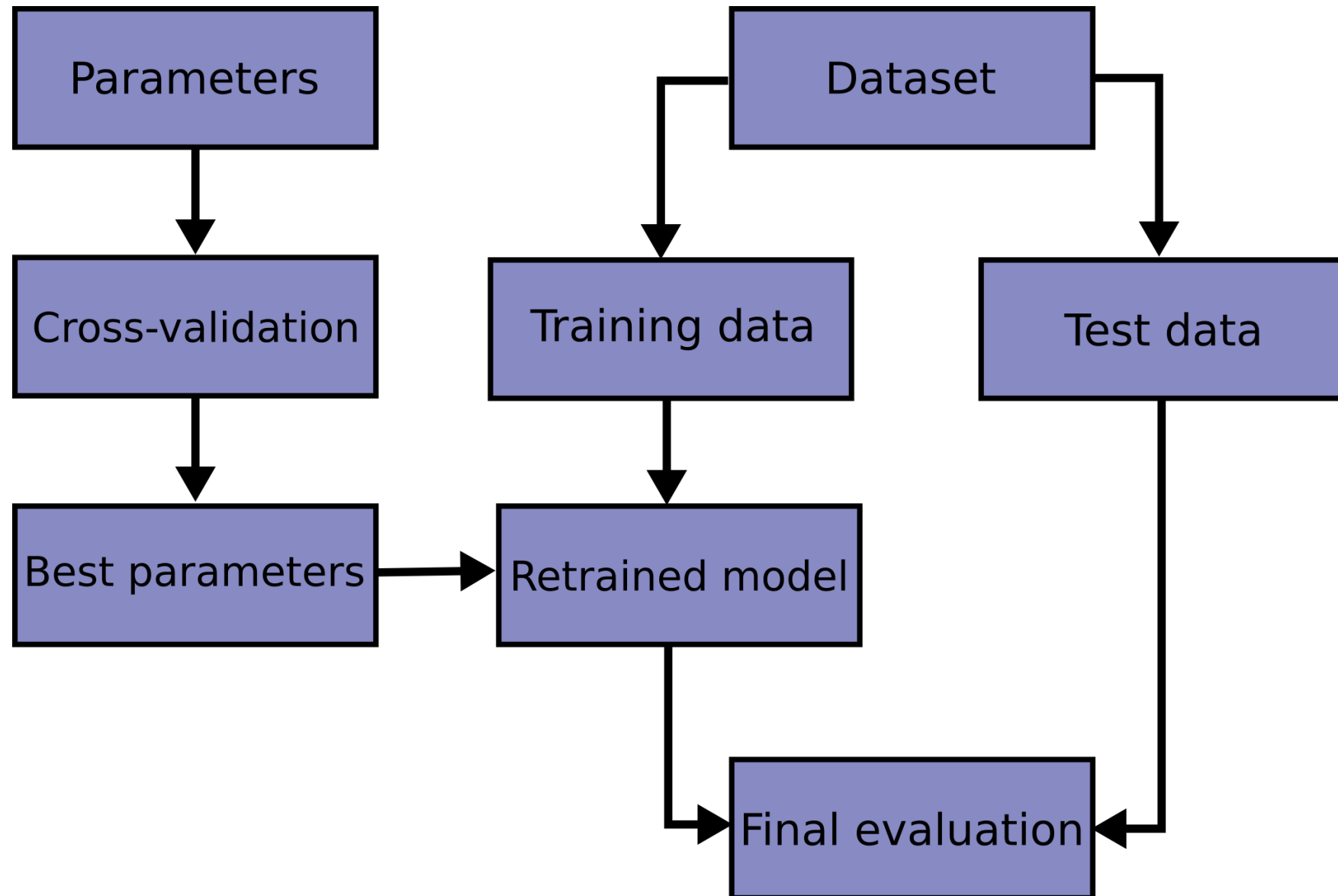
N-gram



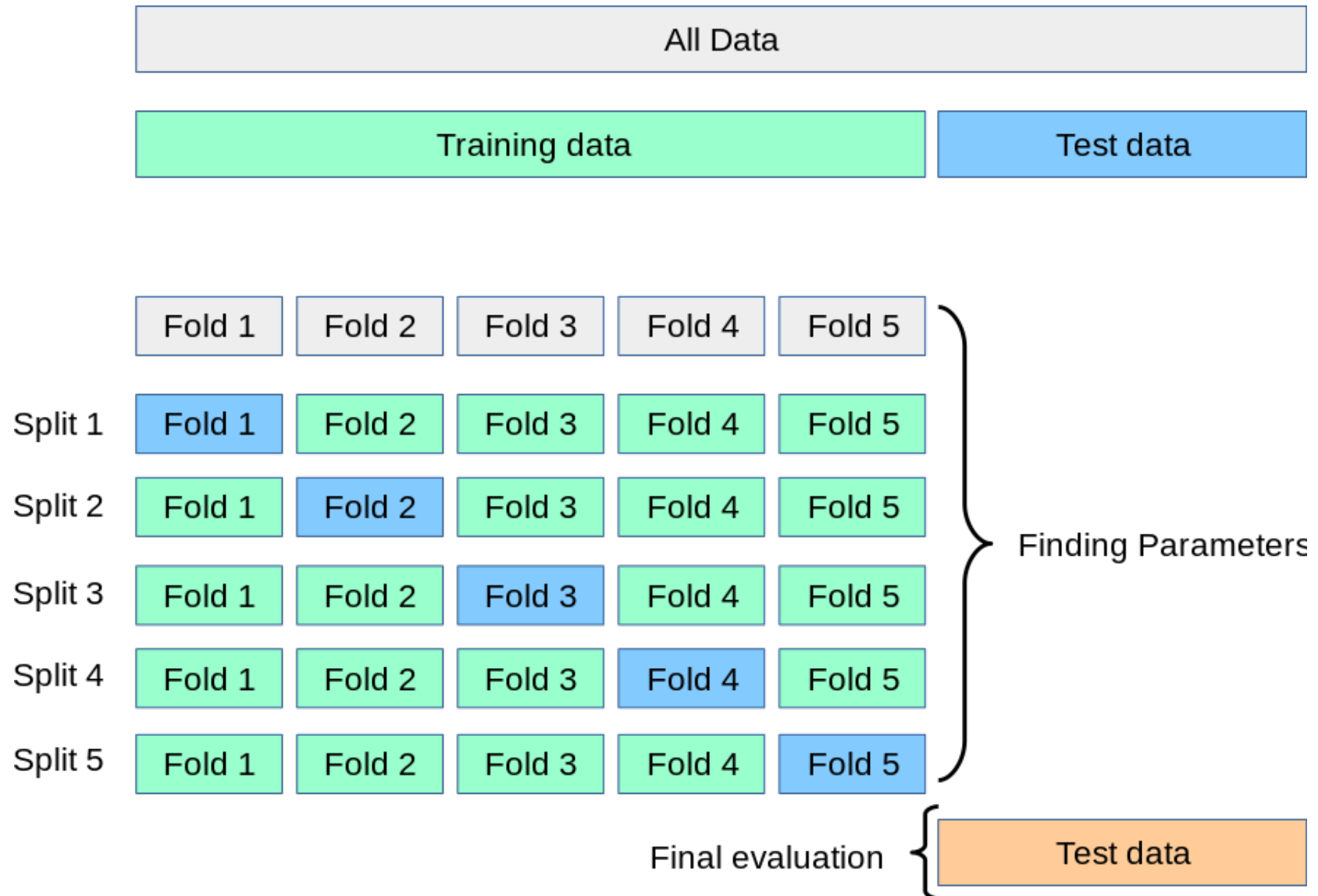
N-grams

- ▶ 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 NTKK:
 - ▶ 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')}`

Grid Search



Cross validation



Hyper-parameter fitting with sklearn

- ▶ `from sklearn.model_selection import GridSearchCV`
 - ▶ Or `import RandomizedSearchCV`
- ▶ `from sklearn.linear_model import LogisticRegression`
- ▶ `clf = LogisticRegression(max_iter=1_000,
 solver='lbfgs',
 multi_class='auto')`
- ▶ `grid_values = {'C':[0.001,0.01,1,10,100]}`
- ▶ `grid = GridSearchCV(clf, param_grid = grid_values, cv=3)`
- ▶ `grid.fit(X_train_enc, y_train)`
- ▶ `grid.best_estimator_.C`

Hyper-parameter fitting in a pipeline with sklearn

- ▶ `from sklearn.pipeline import Pipeline`
- ▶ `pipeline = Pipeline(
 [('feature_extraction', CountVectorizer()),
 ('classification', LogisticRegression(max_iter=1_000,
 solver='lbfgs',
 multi_class='auto'))])`
- ▶ `parameters = {}`
- ▶ `parameters['feature_extraction__max_df'] = [0.8, 0.9]`
- ▶ `parameters['feature_extraction__ngram_range'] = [(1,1), (2,2)]`
- ▶ `parameters['feature_extraction__binary'] = [True, False]`
- ▶ `parameters['classification__C'] = [0.001, 0.01, 1, 10, 100]`
- ▶ `grid = RandomizedSearchCV(pipeline, parameters, cv=3)`
- ▶ `grid.fit(X_train, y_train)`

More on lab – Lab 1

- ▶ study the impact of the representation of words in order to understand the meaning of a sentence or, more precisely, to measure the distance between two sentences.

More on lab – Lab 2

1. Preprocess a text

- ▶ Tokenization
- ▶ Normalisation
- ▶ Stemming / Lemming
- ▶ Stop word removal

2. Extract feature

- ▶ Count Vectorizer
- ▶ TfIdf
- ▶ Ngrams

3. Use the extracted feature for a specific task

- ▶ Sentiment analysis

