



# Natural Language Processing

AIMS

Romain Benassi

Course 1

Spring 2023

# Romain Benassi



**Data Scientist (PhD)** with specialty in Deep Learning, **NLP**, Energy and Bayesian methods

- **Since 2020**: Consultant at **Publicis Sapient France**, mainly in **NLP** field



- **2013-2019** : Data Scientist in the field of energy (in several start-ups)

- **2013** : PhD on Bayesian statistics (**Centrale-Supélec**)



- **2009** : Engineer's degree (**IMT Atlantique**) with specialization in signal processing



# Romain Benassi

## Experience in **Natural Language Processing (NLP)**

### Since 2019, three projects on this field

- **Vidal**: work on deep learning algorithms for automatic indexation of medical documents
- **Enedis**: Automatic classification, and topic extraction, from strategic text documents
- **Tamalou** project: development of a supervised clustering algorithm and automatic reply association for medical texts from internet forums



**Co-speaker in Devovx France 2022 : l'IA pour le bon usage du médicament** (Vidal & Publicis Sapient)

<https://www.youtube.com/watch?v=1qeiou8GGj8>



**One blog article** (in French)

<https://blog.engineering.publicissapient.fr/2021/03/17/nlp-concepts-cles-et-etat-de-lart/>



### Online formation certifications (as student)

- Natural Language Processing Specialization (4-course specialization)
- Introduction to TensorFlow for Artificial Intelligence, Machine Learning, and Deep Learning
- Natural Language Processing in TensorFlow



# Course Schedule

- **Course 1:** NLP introduction
- **Course 2:** Word embedding
- **Course 3:** LSTM (Long Short-Term Memory) principle
- **Course 4:** “Attention” Mechanism and Transformer Architectures
- **Course 5:** Large Language Models and Generative AI

# Evaluation

1. A graded exam will be used as evaluation and will be done at the beginning of the last course (**Course 5**)

**This exercise will contain**

- multiple-choice-questions (MCQ)
- theoretical questions
- Coding questions

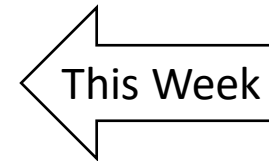
2. A graded project (details given in the **following Courses**)

# Course Schedule

- **Course 1:** NLP introduction
- **Course 2:** Word embedding
- **Course 3:** LSTM (Long Short-Term Memory) principle
- **Course 4:** “Attention” mechanism and Transformer architectures
- **Course 5:** Large Language Models and Generative AI

# Course Schedule

- **Course 1: NLP introduction**
  - **NLP use cases**
  - **Preprocessing** (tokenization, stemming, lemmatization...)
  - **NLP libraries** (spaCy and NLTK)
  - **Text Mining** (Bag of words, TF-IDF)
  - **Topic Modeling** (NMF, LDA)
- **Course 2:** Word embeddings
- **Course 3:** Long Short-Term Memory (LSTM) architecture
- **Course 4:** “Attention” mechanism and Transformer architectures
- **Course 5:** Large Language Models and Generative AI



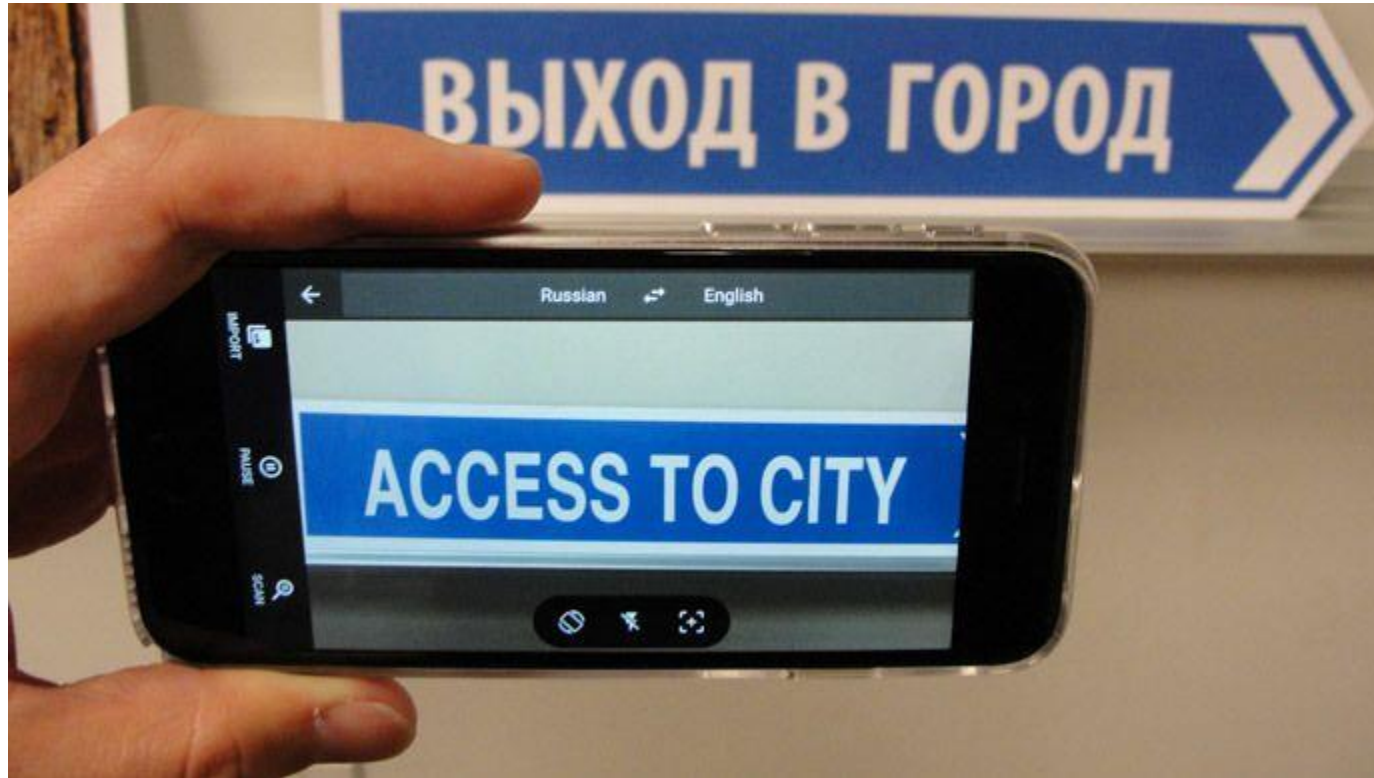
# Course 1: NLP Introduction



# NLP use cases

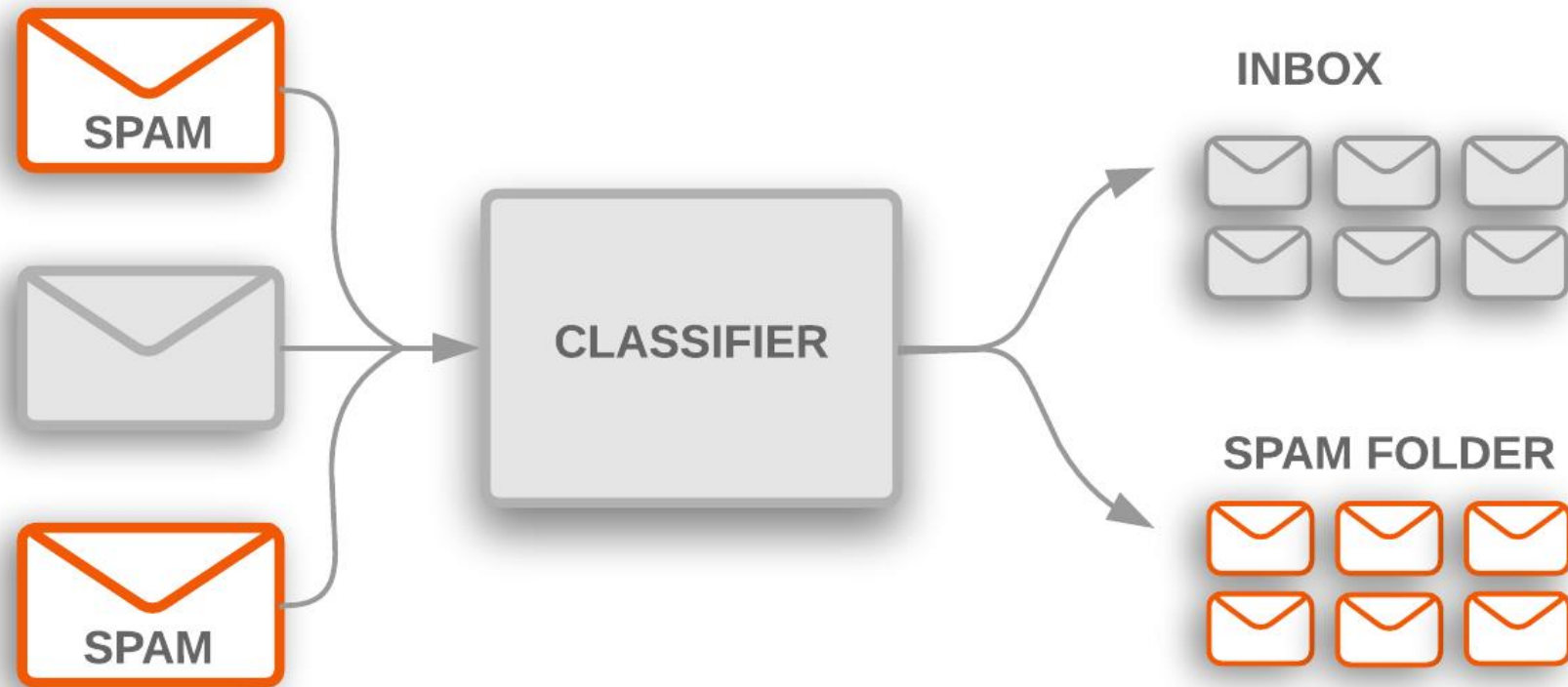
# NLP use cases

## Translation



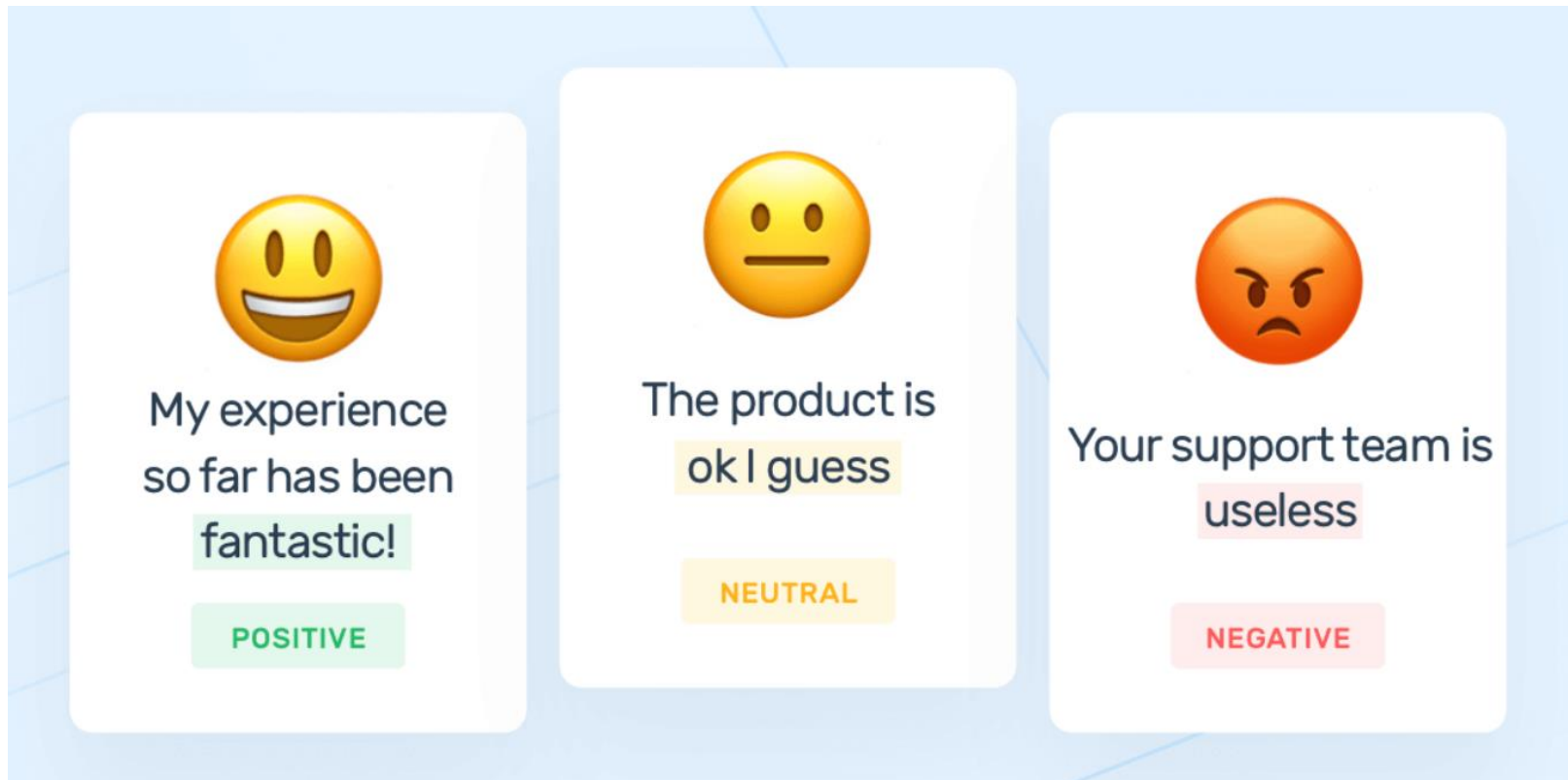
# NLP use cases

## Spam detection



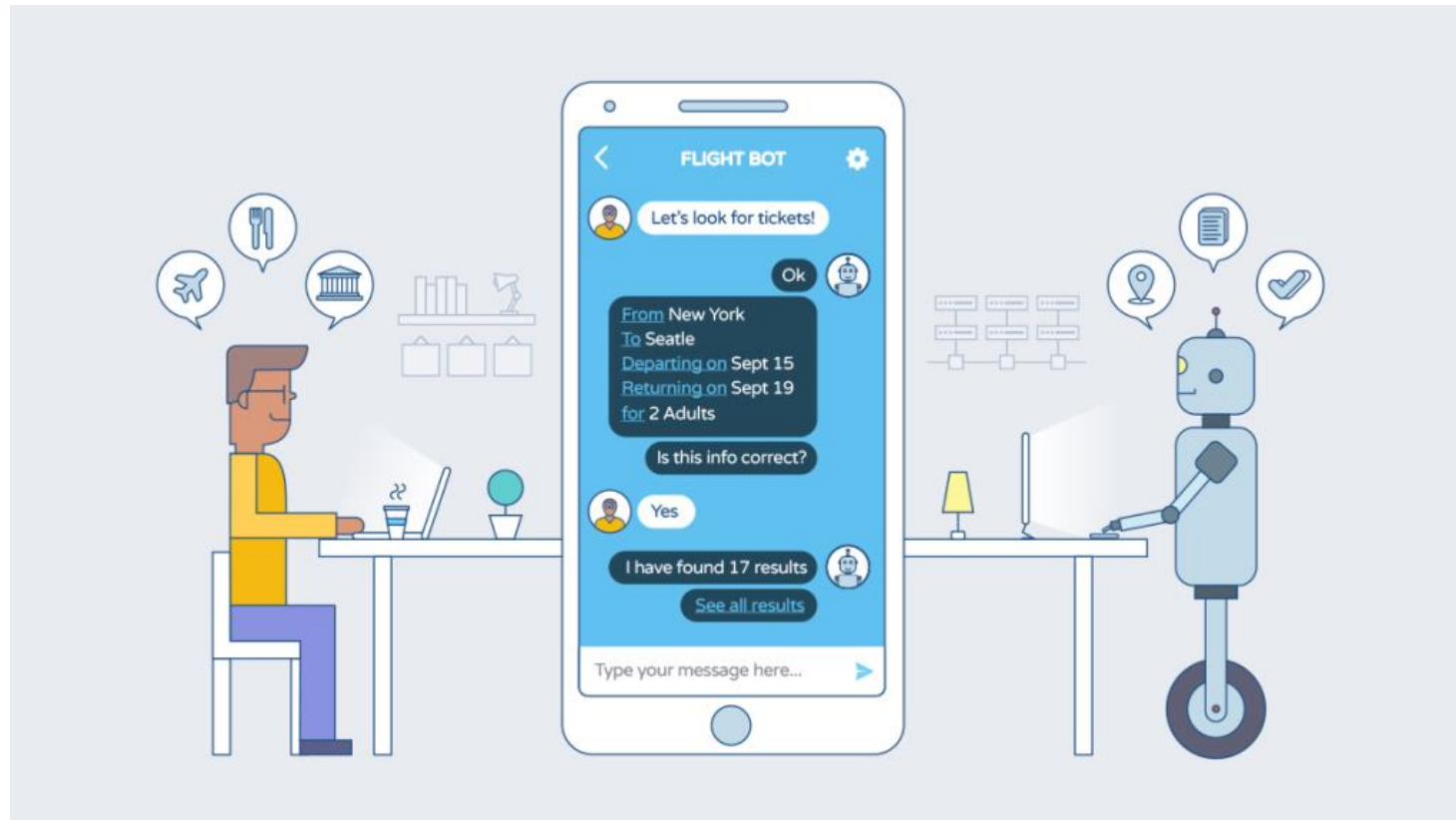
# NLP use cases

## Sentiment analysis



# NLP use cases

## Chatbot



# NLP use cases

**Image generation :** DALL-E 2, Midjourney, Imagen...

*Spider-Man from ancient Rome*



<https://labs.openai.com/s/MEWyqzcl8X85dlaMyNo5OEgy>

# NLP use cases

## Text generation : Chat-GPT

R

Can you write a poem in Shakespeare style?



Of course! I'd be delighted to craft a poem in the style of Shakespeare for you. Here it is:

Upon yon field where sunlight's beams doth dance,  
A tale of love and sorrow shall advance.  
Fair Juliet, a rose of beauty rare,  
Doth captivate young Romeo with her glare.

Their eyes did meet 'neath moon's celestial glow,  
A flame ignited, hearts a-thrill to grow.  
But fate, a cruel mistress, intervened,  
For love, it seemed, by circumstance was weaned.

Their kin, in strife, did brawl with might and hate,  
Which thrust upon them woes, too great a weight.  
Yet undeterred, young lovers found a way,  
To meet clandestinely, both night and day.

But darkness spreads its veil, foreboding nigh,  
And tears of anguish stain their cheeks awry.  
A potion brewed, to feign a death's embrace,  
And grant them fleeting moments, love to chase.

Alas, a tragic twist, unknown to both,  
As Romeo, by false tidings, took an oath.  
He found his love, lifeless, upon her bed,  
And thought to join her, fill their hearts with lead.

# NLP main issue



Computers are known to be good at handling numerical data

➡ **What can we do with text data?!**



# Preprocessing

# Common NLP tasks

- Tokenization
- Stemming
- Lemmatization
- Removal of stop words

# Tokenization

Tokenization is the breaking up of raw text into smaller relevant parts (tokens). These parts can be words or groups of successive letters.

Example of word tokenization:

**Raw text** : *My son's friend, however, plays a high-risk game.*

**Tokenized** : [My] [son] ['s] [friend] [,] [however] [,] [plays] [a] [high] [-] [risk] [game] [.]

# Stemming

Stemming is the process of reducing a word to its root form (word stem).

Most of the time, it consists to drop the last letters of the word until the stem is reached.

Example of stemming:

fishing, fished or fisher => fish

# Stemming

- Porter's algorithm (1980) is one of the most well-known stemming procedure used for English language
- Five reduction phases used in order to get the stem of the word
- Different variants of the original Porter's stemming procedure exists like Snowball or the English Stemmer (Porter2 stemmer)

# Stemming

- Porter's algorithm (1980) is one of the most well-known stemming procedure used for English language
- Five reduction phases used in order to get the stem of the word
- Different variants of the original Porter's stemming procedure exists like Snowball or the English Stemmer (Porter2 stemmer)

## First phase

| S1        | S2 | word       | stem   |
|-----------|----|------------|--------|
| SSES → SS |    | caresses → | caress |
| IES → I   |    | ponies →   | poni   |
|           |    | ties →     | ti     |
| SS → SS   |    | caress →   | caress |
| S →       |    | cats →     | cat    |

## More sophisticated rules

| S1                  | S2 | word         | stem     |
|---------------------|----|--------------|----------|
| (m>0) ATIONAL → ATE |    | relational → | relate   |
|                     |    | national →   | national |
| (m>0) EED → EE      |    | agreed →     | agree    |
|                     |    | feed →       | feed     |

# Stemming

## Porter Stemmer (NLTK) example

```
run----->run  
runner----->runner  
ran----->ran  
runs----->run  
easily----->easili  
fairly----->fairli  
fairness----->fair
```

# Stemming

Snowball Stemmer (NLTK) was developed as well by Porter and offers a slight improvement both in logic and speed.

```
run    -----> run
runner -----> runner
ran    -----> ran
runs   -----> run
easily -----> easili
fairly -----> fair
fairness -----> fair
```



# Lemmatization

Lemmatization is the algorithmic process of determining the canonical form (lemma) of a word.

Unlike stemming, it is not only a word reduction but depends on the meaning of the word in a sentence and needs to consider a language's full vocabulary.

Example of lemmatization:

*was* -> *be*

*meeting* -> *meet* or *meeting* (depending on the context)

*walking* -> *walk* (identical to stemming)

# Lemmatization

- Lemmatization gives a more informative result than stemming
- It looks at surrounding text to correctly identify the part of speech and meaning of the word
- For these reasons, a lemmatization procedure exists directly in spaCy whereas this is not the case for stemming, considered as a less efficient preprocessing step
- Stemming procedures (including both Porter and Snowball) can be found in NLTK if needed

# Removal of stop-words

- A stop-word is a word so frequent that it makes it not relevant to take it into account during an analysis
- Most of the time, there are filtered out before any NLP processing
- Obviously, each language has its own specific set of stop-words
- Examples
  - In English: *the, is, at, which, on...*
  - In French: *le, la, du, ce...*
- spaCy has a list of 305 English stop-words

# NLP Libraries

# spaCy and NLTK

- spaCy (2015) and NLTK (2001) are both popular NLP libraries
- spaCy implements generally only one algorithm for each task, but the most efficient one currently available
- NLTK provides generally several algorithms for each tasks, but with less efficient implementations

spaCy gives more efficient implementations but offers less possibilities than NLTK

The choice between both should be made depending on the applications

# spaCy and NLTK

| SYSTEM       | ABSOLUTE (MS PER DOC) |       |            | RELATIVE (TO SPACY) |      |            |
|--------------|-----------------------|-------|------------|---------------------|------|------------|
|              | TOKENIZE              | TAG   | PARSE      | TOKENIZE            | TAG  | PARSE      |
| <b>spaCy</b> | 0.2ms                 | 1ms   | 19ms       | 1x                  | 1x   | 1x         |
| CoreNLP      | 0.18ms                | 10ms  | 49ms       | 0.9x                | 10x  | 2.6x       |
| ZPar         | 1ms                   | 8ms   | 850ms      | 5x                  | 8x   | 44.7x      |
| NLTK         | 4ms                   | 443ms | <i>n/a</i> | 20x                 | 443x | <i>n/a</i> |

# spaCy and NLTK

|                         | SPACY  | SYNTAXNET | NLTK   | CORENLP |
|-------------------------|--------|-----------|--------|---------|
| Programming language    | Python | C++       | Python | Java    |
| Neural network models   | ✓      | ✓         | ✗      | ✓       |
| Integrated word vectors | ✓      | ✗         | ✗      | ✗       |
| Multi-language support  | ✓      | ✓         | ✓      | ✓       |
| Tokenization            | ✓      | ✓         | ✓      | ✓       |
| Part-of-speech tagging  | ✓      | ✓         | ✓      | ✓       |
| Sentence segmentation   | ✓      | ✓         | ✓      | ✓       |
| Dependency parsing      | ✓      | ✓         | ✗      | ✓       |
| Entity recognition      | ✓      | ✗         | ✓      | ✓       |
| Coreference resolution  | ✗      | ✗         | ✗      | ✓       |

# NLP Basics: Tutorial

*course1\_basics\_NLP.ipynb*

**Goal**: Get used to classic NLP preprocessing operations (lemmatization, stemming, entity detection...) using both spaCy and NLTK library



# Text Mining

# Text Mining

- Examining of document collection (*corpus*) to discover new information
- This can be used as a first step for lots of different applications (topic modeling, spam detection, sentiment analysis, security...)
- We will focus on two of the most famous text mining approaches
  - Bag-of-words (BoW) model
  - TF-IDF

# Bag-of-words (BoW) model

In this model, a document  $d$  is represented as a set of tuples  
 $\{w: \text{nb of occurrences of } w \text{ in } d \mid \text{for all word } w \text{ in } d\}$

## Example

For the (short) document:

*“Xavier likes to play football. Eric likes football too.”*

The bag-of-word representation is:

*{Xavier: 1, likes: 2, to: 1, play: 1, football: 2, Eric: 1, too: 1}*

# Bag-of-words (BoW) model

In this model, a document  $d$  is represented as a set of tuples  
 $\{w: \text{nb of occurrences of } w \text{ in } d \mid \text{for all word } w \text{ in } d\}$

## Example

For the (short) document:

*“Xavier **likes** to play **football**. Eric **likes** **football** too.”*

The bag-of-word representation is:

$\{Xavier: 1, likes: 2, to: 1, play: 1, football: 2, Eric: 1, too: 1\}$

It is equivalent to a word-frequency histogram representation

# Bag-of-words (BoW) model

If we have several texts:

*Document1: “Xavier likes to play football. Eric likes football too.”*

*Document2: “Eric prefers tennis to football.”*

We have the representation below:

|           | Xavier | likes | to | play | football | Eric | too | prefers | tennis |
|-----------|--------|-------|----|------|----------|------|-----|---------|--------|
| Document1 | 1      | 2     | 1  | 1    | 2        | 1    | 1   | 0       | 0      |
| Document2 | 0      | 0     | 1  | 0    | 1        | 1    | 0   | 1       | 1      |

This representation can be used to characterize texts (e-g in spam filtering)

# Bag-of-words (BoW) model

If we have several texts:

*Document1: "Xavier likes to play football. Eric likes football too."*

*Document2: "Eric prefers tennis to football."*

The documents are  
transformed into numerical  
representations

We have the representation below:

|           | Xavier | likes | to | play | football | Eric | too | prefers | tennis |
|-----------|--------|-------|----|------|----------|------|-----|---------|--------|
| Document1 | 1      | 2     | 1  | 1    | 2        | 1    | 1   | 0       | 0      |
| Document2 | 0      | 0     | 1  | 0    | 1        | 1    | 0   | 1       | 1      |

This representation can be used to characterize texts (e-g in spam filtering)

# Bag-of-words (BoW) model

This simple model has some drawbacks

- Term frequencies are not necessarily the best representation of a text
- Each time of new word appears, the length of the vector increases as well
- Most of the time, text representations are sparse (many zeros)
- No information about grammar or about the original word ordering is kept

# TF-IDF

- TF-IDF means term frequency-inverse document frequency
- This is a ponderation method used in information retrieval.
- This statistical measure gives **an evaluation of how important is a word to a document**, depending on the corpus considered
- The weight increase proportionally to the occurrences of a word in a text
- This weight is **also** offset by the number of documents in the corpus containing the word



# TF-IDF

**TF-IDF** is calculated as :  $\text{tfidf}(t, d, D) = \text{tf}(t, d) \cdot \text{idf}(t, D)$

## TF-IDF: $\text{tf}(t,d)$

**TF-IDF** is calculated as :  $\text{tfidf}(t, d, D) = \text{tf}(t, d) \cdot \text{idf}(t, D)$

**Term frequency  $\text{tf}(t,d)$**

$$\text{tf}(t, d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}},$$

With  $f_{t,d}$  the number of times term  $t$  appears in document  $d$

# TF-IDF: $tf(t,d)$

Term frequency  $tf(t,d)$  variants :

Variants of term frequency (tf) weight

| weighting scheme         | tf weight  |
|--------------------------|--|
| binary                   | 0, 1   |
| raw count                | $f_{t,d}$  |
| term frequency           | $f_{t,d} / \sum_{t' \in d} f_{t',d}$                           |
| log normalization        | $\log(1 + f_{t,d})$  |
| double normalization 0.5 | $0.5 + 0.5 \cdot \frac{f_{t,d}}{\max_{\{t' \in d\}} f_{t',d}}$ |
| double normalization K   | $K + (1 - K) \frac{f_{t,d}}{\max_{\{t' \in d\}} f_{t',d}}$     |

## TF-IDF: idf(t,d)

**TF-IDF** is calculated as :  $\text{tfidf}(t, d, D) = \text{tf}(t, d) \cdot \text{idf}(t, D)$

**Inverse document frequency idf(t,D)**

$$\text{idf}(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

With  $N$  the number of documents in corpus  $D$

# TF-IDF: idf(t,d)

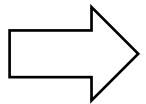
Inverse document frequency idf(t,D) variants :

Variants of inverse document frequency (idf) weight

| weighting scheme                         | idf weight ( $n_t =  \{d \in D : t \in d\} $ )               |
|--|--|
| unary                                    | 1  |
| inverse document frequency               | $\log \frac{N}{n_t} = -\log \frac{n_t}{N}$                   |
| inverse document frequency smooth        | $\log \left( \frac{N}{1 + n_t} \right) + 1$                  |
| inverse document frequency max           | $\log \left( \frac{\max_{t' \in d} n_{t'}}{1 + n_t} \right)$ |
| probabilistic inverse document frequency | $\log \frac{N - n_t}{n_t}$                                   |

# TF-IDF

**TF-IDF** is calculated as :  $\text{tfidf}(t, d, D) = \text{tf}(t, d) \cdot \text{idf}(t, D)$



$$\frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}} \cdot \log \frac{N}{|\{d \in D : t \in d\}|}$$

# TF-IDF

*Document1: “Xavier likes to play football. Eric likes football too.”*

*Document2: "Eric prefers tennis to football."*

[illegible]

# TF-IDF

*Document1: “Xavier likes to play football. Eric likes football too.”*

*Document2: “Eric prefers tennis to football.”*

|             | Xavier | likes  | to  | play   | football | Eric | too    | prefers | tennis |
|-------------|--------|--------|-----|--------|----------|------|--------|---------|--------|
| Document1   | 1      | 2      | 1   | 1      | 2        | 1    | 1      | 0       | 0      |
| Document2   | 0      | 0      | 1   | 0      | 1        | 1    | 0      | 1       | 1      |
| idf         | Log(2) | Log(2) | 0   | Log(2) | 0        | 0    | Log(2) | Log(2)  | Log(2) |
| tf_doc1     | 1/9    | 2/9    | 1/9 | 1/9    | 2/9      | 1/9  | 1/9    | 0       | 0      |
| tf_doc2     | 0      | 0      | 1/5 | 0      | 1/5      | 1/5  | 0      | 1/5     | 1/5    |
| tf-idf_doc1 | 0,033  | 0,067  | 0   | 0,033  | 0        | 0    | 0,033  | 0       | 0      |
| tf-idf_doc2 | 0      | 0      | 0   | 0      | 0        | 0    | 0      | 0,06    | 0,06   |

Again, the documents  
are transformed  
into  
numerical  
representations



# TF-IDF: Exercise

*course1\_tfidf\_ex.ipynb*

**Goal:** Illustration of TF-IDF on a very simple corpus

**Remarks:**

- We compute the TF-IDF twice on the same corpus (directly from the formula and with *sklearn* library)
- The results may be different depending on the settings used with the *sklearn* function

# Topic Modeling

# Topic modeling

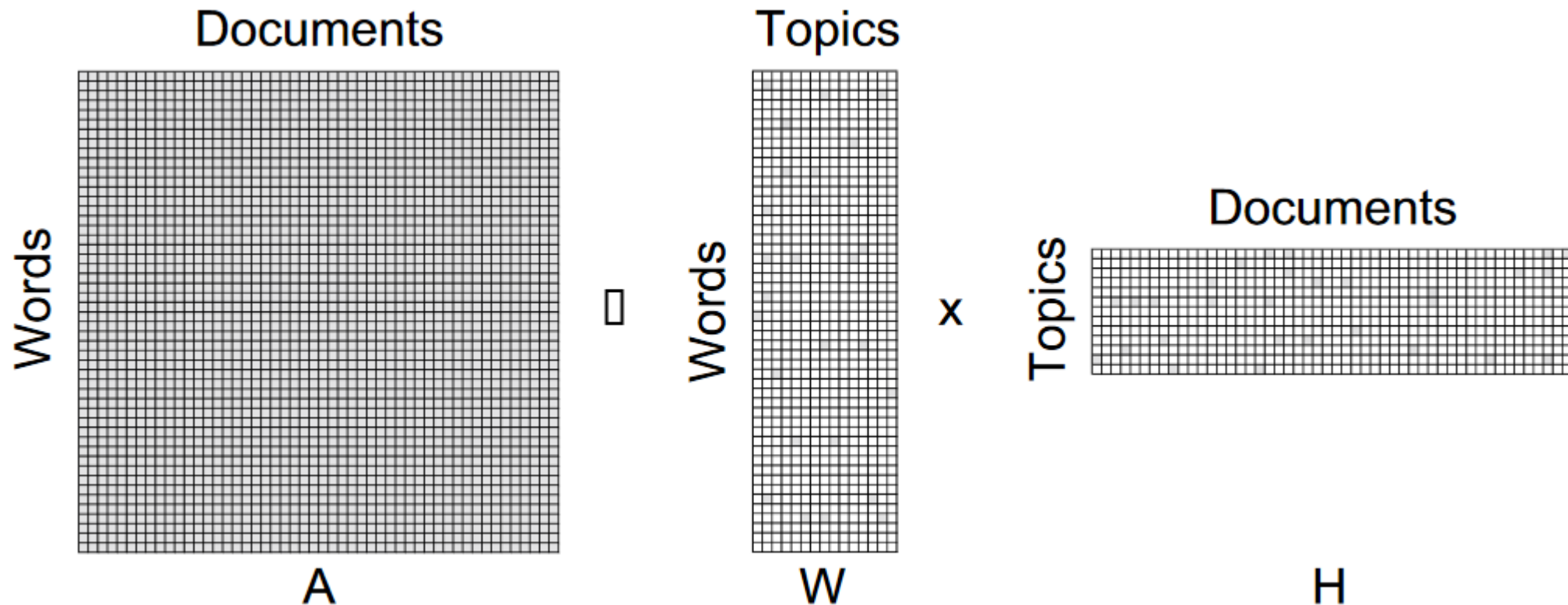
- Unsupervised algorithms
- Use to analyze text by clustering document into topics
- The idea is for the documents in the same cluster to correspond to a common topic
- Can be used to analyze large volumes of text

# Non-negative Matrix factorization (NMF)

- Non-negative Matrix Factorization is an unsupervised topic modeling algorithm
- The number of clusters must be defined by the user
- It allows both clustering and dimensionality reduction
- Generally, it is used from a **TF-IDF procedure** performed on all the documents of the corpus

# Non-negative Matrix factorization (NMF)

The procedure involves the decomposition of matrix A as the product of two other matrices W and H. All three matrices are non-negative



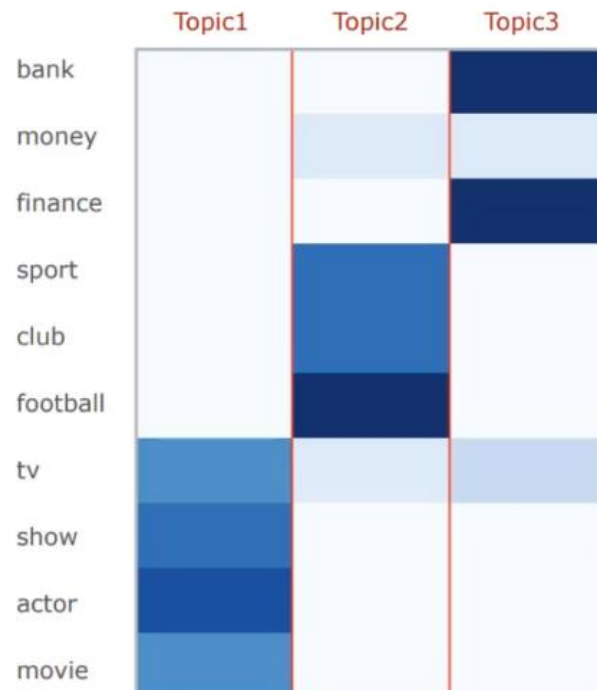
# Non-negative Matrix factorization (NMF)

The  $A'$  matrix comes from a TF-IDF vectorization

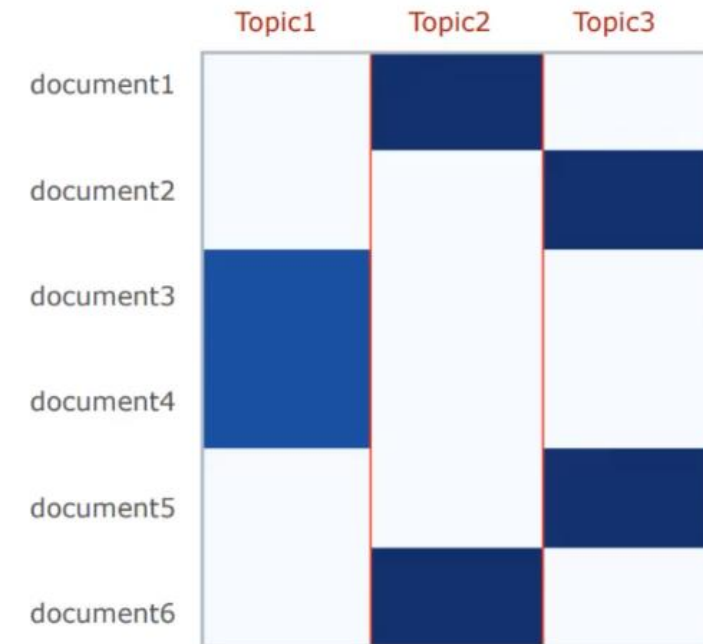


# Non-negative Matrix factorization (NMF)

*Basis vectors  $\mathbf{W}$ : topics (clusters)*



*Coefficients  $\mathbf{H}^T$ : memberships for documents*



# Non-negative Matrix factorization (NMF)

## Procedure

- **Inputs:** Matrix  $A$ , a number  $k$ , matrices  $W$  and  $H$  randomly generated
- **Objective function:** the reconstruction error between  $A$  and  $WH$

$$\frac{1}{2} ||\mathbf{A} - \mathbf{WH}||_F^2 = \sum_{i=1}^n \sum_{j=1}^m (A_{ij} - (WH)_{ij})^2$$

- The computation of  $W$  and  $H$  are performed sequentially with an **Expected-Maximization** (EM) procedure until convergence

1. Update  $\mathbf{H}$

$$H_{cj} \leftarrow H_{cj} \frac{(W\mathbf{A})_{cj}}{(W\mathbf{WH})_{cj}}$$

2. Update  $\mathbf{W}$

$$W_{ic} \leftarrow W_{ic} \frac{(\mathbf{AH})_{ic}}{(\mathbf{WHH})_{ic}}$$



# Latent Dirichlet Allocation (LDA)

- Dirichlet was a German mathematician (1805-1859)
- The Dirichlet Distribution was named after him
- LDA is a statistical model based on this specific distribution
- It was first presented in 2002 (Blei, Ng and Jordan) for document topic detection
- Since then, this model has been used in many applications (data mining, NLP...)

# Latent Dirichlet Allocation (LDA)

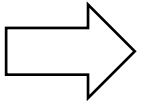
## LDA principles

- Documents with similar topics use similar groups of words
- Latent topics correspond to groups of word frequently occurring together in the corpus documents

# Latent Dirichlet Allocation (LDA)

## LDA principles

- Documents with similar topics use similar groups of words
- Latent topics correspond to groups of word frequently occurring together in the corpus documents

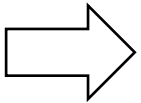


- Each document is a mixture of a small number of topics
- Each word's presence is attributable to a topic (considered here as a probability distribution over words)

# Latent Dirichlet Allocation (LDA)

## LDA principles

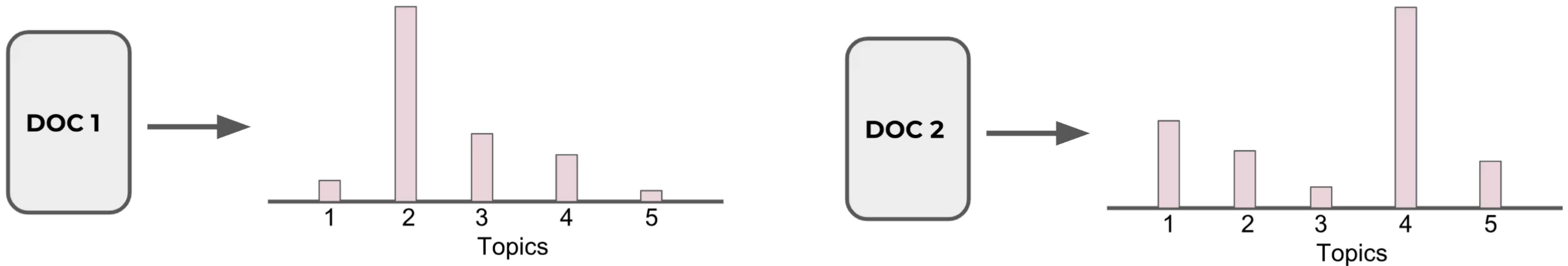
- Documents with similar topics use similar groups of words
- Latent topics correspond to groups of word frequently occurring together in the corpus documents



- **Each document is a mixture of a small number of topics**
- Each word's presence is attributable to a topic (considered here as a probability distribution over words)

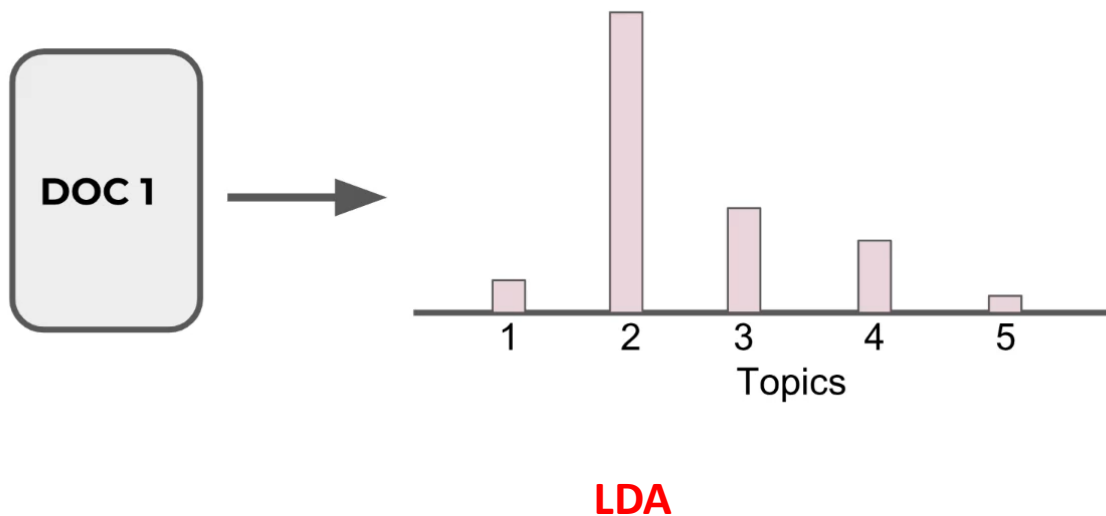
# Latent Dirichlet Allocation (LDA)

Each document is a mixture of a small number of topics

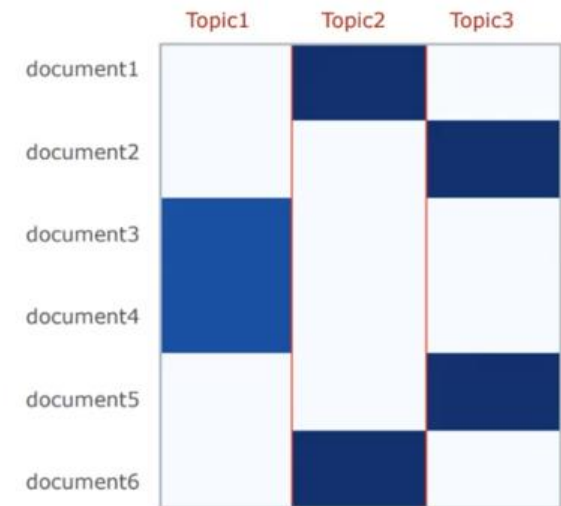


# Latent Dirichlet Allocation (LDA)

**Comparison with NMF:** weights of topics for each document



Coefficients  $\mathbf{H}^T$ : memberships for documents

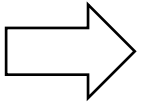


NMF

# Latent Dirichlet Allocation (LDA)

## LDA principles

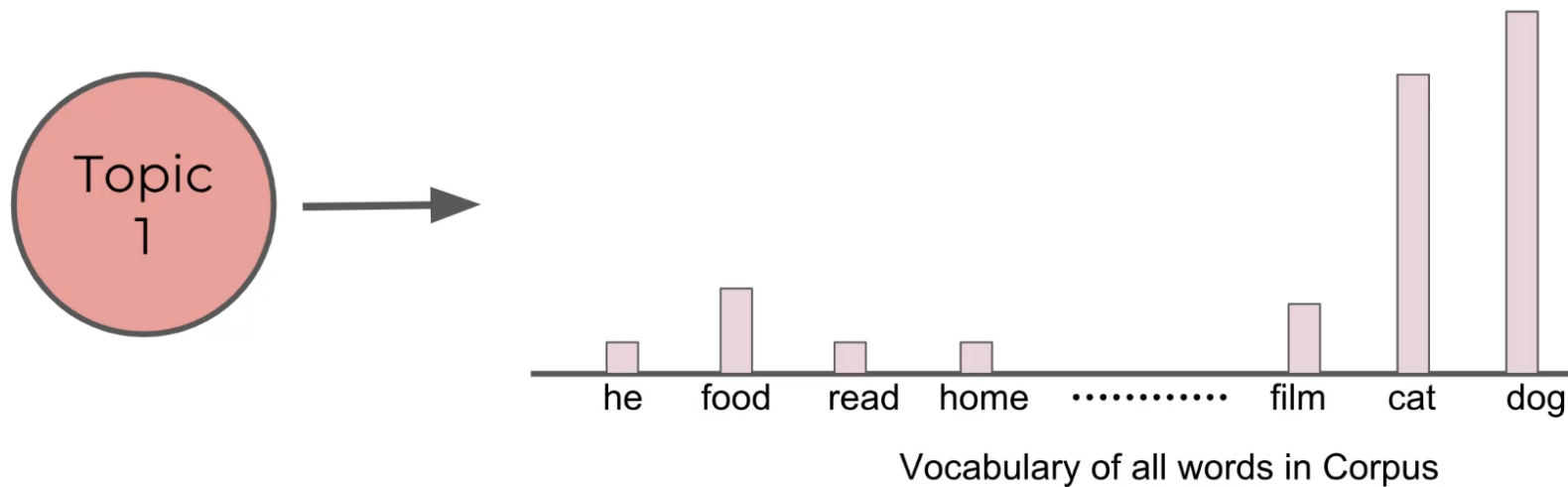
- Documents with similar topics use similar groups of words
- Latent topics correspond to groups of word frequently occurring together in the corpus documents



- Each document is a mixture of a small number of topics
- **Each word's presence is attributable to a topic (considered here as a probability distribution over words)**

# Latent Dirichlet Allocation (LDA)

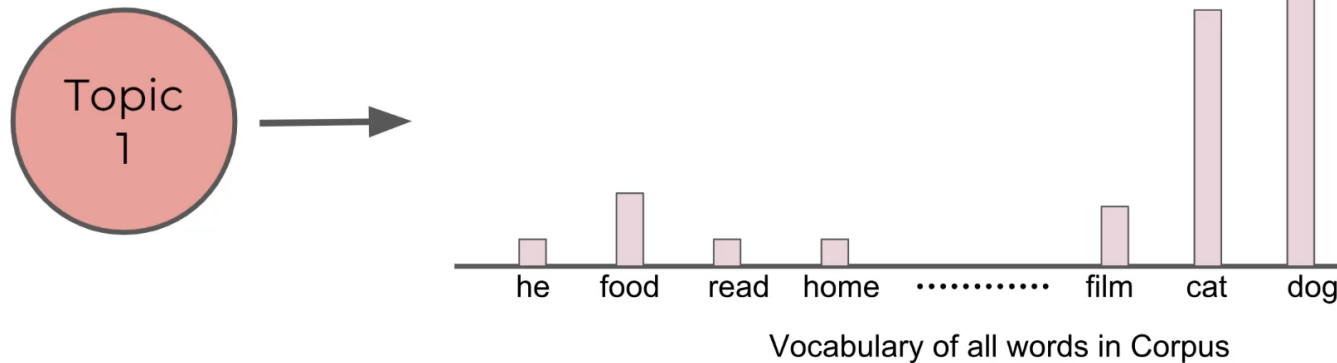
Each word's presence is attributable to a topic (considered here as a probability distribution over words)





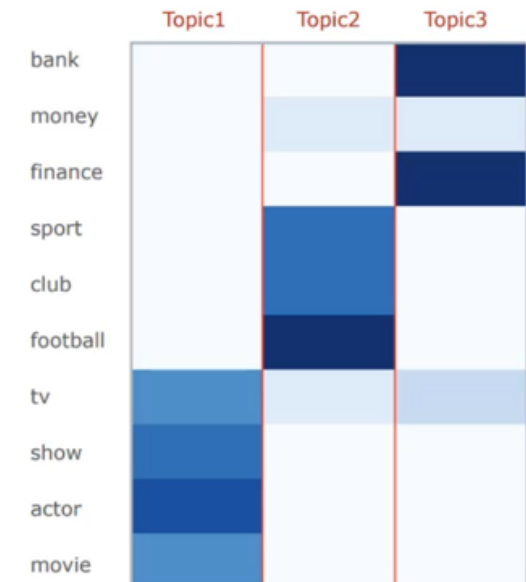
# Latent Dirichlet Allocation (LDA)

## Comparison with NMF: weights of words on each topic



**LDA**

*Basis vectors  $\mathbf{W}$ : topics (clusters)*



**NMF**

# Latent Dirichlet Allocation (LDA)

A document can be seen as

- A chosen  $N$  number of words
- A specific mixture of topic (selected according to a Dirichlet distribution over the set of topics)-e.g.: 50% cinema, 30% sport, 20% animals

Each word in the document are generated by the following procedure

- Chose a topic (according to the topics multinomial distribution) and generate from it a word

This is the generative model use to create a set of documents

# Latent Dirichlet Allocation (LDA)

Let's go back to the topic modelling objective

- We have a set of documents
- We chose a number  $k$  of topics to discover
- We use LDA for topic modelling, and to determine the topic representation of each document and the words associated to each topic

# Latent Dirichlet Allocation (LDA)

## Procedure

1. Randomly assign, according to a Dirichlet distribution, each word of each document to one of the  $k$  topics
2. You have a first (but meaningless) representation of the word and document distribution of the topics
3. We iterate over every word and every document in the following way. For each topic  $t$ , document  $d$  and word  $w$  we calculate:  
 $P(t/d)$  = proportion of words in  $d$  currently assigned to  $t$   
 $P(w/t)$  = proportion of assignments to  $t$  over all documents containing  $w$
4. Reassign each word  $w$  a new topic  $t$ , according the probability  $p(w/t) \cdot p(t/d)$
5. Repeat the previous steps enough times to converge to acceptable topic assignments

# Latent Dirichlet Allocation (LDA)

- In the end of the procedure, we can assign document to topics
- However, we do not have relevant labels to associate topics with
- It is up to the user, in a next step, to find a way to identify these topics found

# LDA & NMF: Exercise

*course1\_LDA\_NMF\_ex.ipynb*

**Goal:** Learn how to use **LDA** and **NMF** to address **Topic Modeling** on a **real dataset**

## **Remarks:**

- We use *scikit-learn* implementation for **LDA** and **NMF**
- The dataset is composed of **real** questions taken from the internet forum ***Quora***

# Take-away from Course 1

- One of the main issues of NLP is finding a **relevant numerical representation for texts**
- Before converting texts into numerical vectors, some **preprocessing steps** (tokenization, lemmatization, stemming, stop word removal...) exist and can prove **very useful**
- Bag-of-words (BoW) and TF-IDF can transform a corpus of texts into an **array of numerical vectors**
- These numerical vectors **can be used as input data** for classical ML procedures (e.g., topic modeling with LDA and NMF algorithms which rely respectively on BoW and TF-IDF)

# References

## Online formations

- <https://www.udemy.com/course/nlp-natural-language-processing-with-python>
- <https://www.coursera.org/specializations/natural-language-processing>

## Book

*Koehn*, Statistical Machine Translation, Cambridge University Press (2009)