

# Data Challenge

NLP – Introduction, Preprocessing, compréhension phrases, vectorisation

---

Université Paris Dauphine-PSL

# NLP

---

1. Introduction
2. NLP-preprocessing
3. Prétraitement des données
4. Compréhension des phrases
5. Vectorisation des données



# 1. Introduction

---

- NLP = traitement automatique du langage naturel
  - Croisement entre la linguistique, l'informatique, la statistique et des sciences cognitives. Champ de l'intelligence artificielle.
  - Langage naturel :
    - Différent de langage de programmation
    - Utilisé par humains pour communiquer
    - Pas de compilateur en langage de programmation
- Objectifs NLP
  1. Construire du langage à partir de modèles
  2. Donner des « labels » à des jeux de données (opinion, classification, sentiment...)
  3. Rendre explicable des documents, synthétiser...

# 1. Introduction

- De nombreuses applications

<b>Search</b>	Web	Documents	Autocomplete
<b>Editing</b>	Spelling	Grammar	Style
<b>Dialog</b>	Chatbot	Assistant	Scheduling
<b>Writing</b>	Index	Concordance	Table of contents
<b>Email</b>	Spam filter	Classification	Prioritization
<b>Text mining</b>	Summarization	Knowledge extraction	Medical diagnoses
<b>Law</b>	Legal inference	Precedent search	Subpoena classification
<b>News</b>	Event detection	Fact checking	Headline composition
<b>Attribution</b>	Plagiarism detection	Literary forensics	Style coaching
<b>Sentiment analysis</b>	Community morale monitoring	Product review triage	Customer care
<b>Behavior prediction</b>	Finance	Election forecasting	Marketing
<b>Creative writing</b>	Movie scripts	Poetry	Song lyrics

Un moteur de recherche fournit des résultats à partir de l'indexage des pages Web en tenant compte du texte en langage naturel

La saisie semi-automatique (Autocomplete) utilise les techniques de NLP pour compléter la pensée

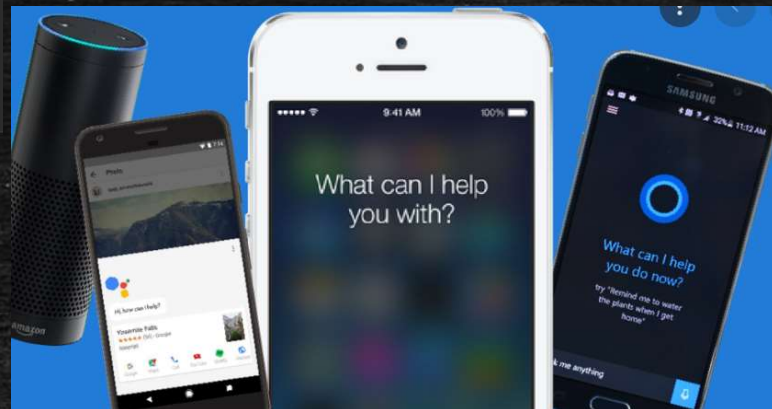
Même technique que sur les téléphones portables...



# 1. Introduction

- Solutions utilisant techniques NLP

"**Alexa**, where is the nearest HIV testing clinic?"



Conversation

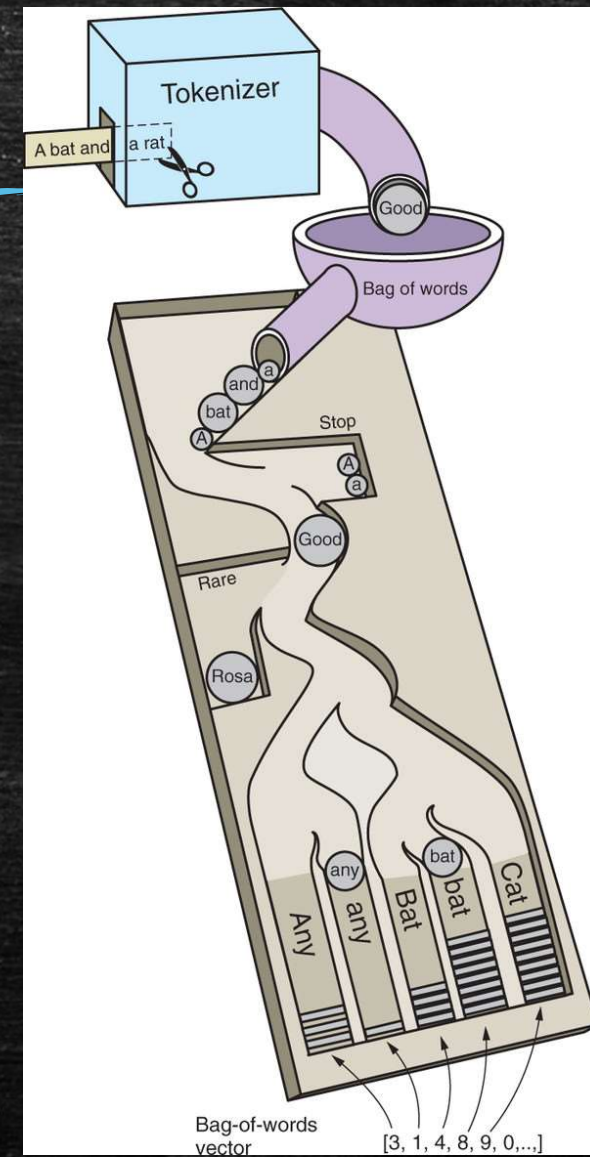
Bonjour ! je suis **Thomas**, votre nouvel assistant virtuel.  
J'apprends encore tous les jours grâce à vous, mais **je peux vous apporter mon aide** sur de nombreux sujets :

- Comment contacter un conseiller ?
- Comment verser sur mon plan d'épargne ?
- Comment récupérer mon épargne ?
- Où en est mon opération ?
- Comment modifier mes informations personnelles ?

Posez-moi une question...

# 1. Introduction

- NLP pre-processing





## 2. NLP pre-processing : Tokenizer

---

- NLP pre-processing :
- Identique aux étapes de construction d'un Chatbot :
  - Parse : extraire caractéristiques du texte
  - Analyze : évaluer les textes par des scores, analyse sémantique et grammaticale
  - Generate : composer des réponses possibles
  - Execute : générer des conversations basés sur l'historique



## 2. NLP pre-processing : Tokenizer

---

- Tokenizer : création de vocabulaire

- Première étape dans une analyse NLP

A partir de documents textes, construction de vocabulaire

Exemple de texte :

« Over the course of forty years, Thomas Jefferson designed and redesigned Monticello. The interior of the house was crafted by John Hemmings, an enslaved master craftsman. Construction on Monticello began in 1768, when Jefferson was twenty-five years old.

The first portion of the house, a two-story building, was completed in 1772. However, revisions and additions beginning in 1791 led to an expanded three-story home that remained unfinished until 1809. Thomas Jefferson would continue to make improvements to Monticello until his death in 1826. »





## 2. NLP pre-processing : Tokenizer

- Création de vocabulaire sur un exemple
  - Approche élémentaire : identification des mots en considérant les espaces comme séparateur

```
[1] import numpy as np

[2] inning in 1791 led to an expanded three-stor

[10] token_sequence = str.split(sentence)
      # affichage des 10 premiers mots
      token_sequence[0:10]

['Over',
 'the',
 'course',
 'of',
 'forty',
 'years,',
 'Thomas',
 'Jefferson',
 'designed',
 'and']
```

Fonction split dans  
bibliothèque numpy

Les éléments sont  
appelés des « Tokens »  
(Over, of...)

Ne distingue pas les  
chiffres des mots....

Fonction split associée à  
un objet de type « List »

```
▶ sentence.split()

['Over',
 'the',
 'course',
 'of',
 'forty',
 'years,',
 'Thomas',
 'Jefferson',
```

## 2. NLP pre-processing : Tokenizer

### - Création de vocabulaire

```
#découpage du texte en tokens
tokens = sentence.split() # type liste
# définition du vocabulaire
# transformation liste en ensemble (pas de répétitions des mots et tri alphanumérique)
vocabulary = sorted(set(tokens))
vocabulary[0:10]
```

```
['1768,',
 '1772.',
 '1791',
 '1809.',
 '1826.',
 'Construction',
 'Hemmings,',
 'However,',
 'Jefferson',
 'John']
```

```
len(vocabulary)
```

64

Mélange de nombres,  
de noms propres, et  
communs....

Vocabulaire de taille 64



## 2. NLP pre-processing : Tokenizer

### - Création de vocabulaire

- Codage du texte dans une matrice de type one-hot encoder (dummy vector) → matrice creuse (sparse matrix)

```
# création d'une matrice one-hot encoder
M = np.zeros((len(tokens),len(vocabulary)))
for i, word in enumerate(tokens):
    M[i, vocabulary.index(word)]=1

M= pd.DataFrame(M, index = tokens, columns=vocabulary)
M.iloc[[0:8],52:64]
```

	the	three-story	to	twenty-five	two-story	unfinished	until	was	when	would	years	years,
Over	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
the	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
course	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
of	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
forty	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
years,	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
Thomas	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Jefferson	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Remarque :

- « years, » et « years » sont différents
- Chiffres distingués
- ....

→ Matrice creuse (sparse matrix) entraîne des problèmes de stockage et de taille mémoire.

## 2. NLP pre-processing : Matrice creuse

- Matrice creuse
  - Matrice contenant beaucoup de zéros
  - Calculs matriciels simplifiés mais stockage important dans RAM

### Sparse Matrix

- 20,000 rows (the tweets)
- 43,000 columns (the words)
- $20,000 * 43,000 = 860,000,000$
- Only 177,000 non-0 entries. About .02%

### Sparse matrix example:

```
1 0 0 0 0 0 0 0
0 0 2 0 0 0 0 0
0 1 0 0 0 3 0 0
1 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 1
0 0 1 0 0 0 4 0
```

Exemple : analyse de tweets

- Différents formats pour stocker les matrices
  - COO (COOrdinates)
  - CSR (Compressed Sparse Row ou Yale Format)



## 2. NLP pre-processing : Matrice creuse

---

- Matrice creuse
  - COO format
    - Vecteur des indices des lignes (row) pour chaque valeur non nulle
    - Vecteur des indices des colonnes pour chaque valeur
    - Vecteur des valeurs de la matrice

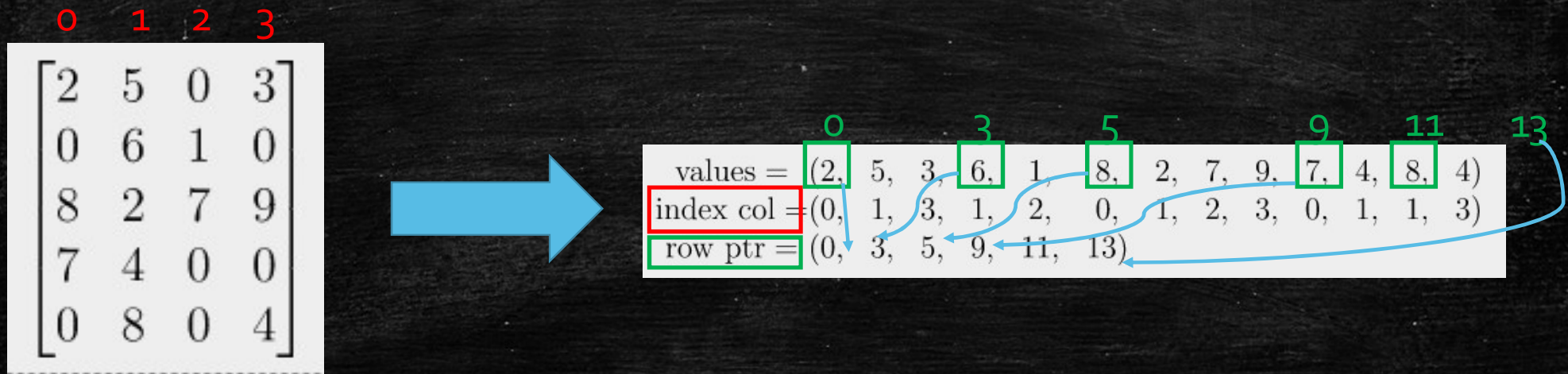
$$\begin{bmatrix} 2 & 5 & 0 & 3 \\ 0 & 6 & 1 & 0 \\ 8 & 2 & 7 & 9 \\ 7 & 4 & 0 & 0 \\ 0 & 8 & 0 & 4 \end{bmatrix}$$



row = (0, 0, 0, 1, 1, 2, 2, 2, 2, 3, 3, 4, 4)  
index col = (0, 1, 3, 1, 2, 0, 1, 2, 3, 0, 1, 1, 3)  
values = (2, 5, 3, 6, 1, 8, 2, 7, 9, 7, 4, 8, 4)

## 2. NLP pre-processing : Matrice creuse

- Matrice creuse
  - CSR (ou Yale format)
    - Vecteur des valeurs de la matrice
    - Vecteur des indices des colonnes pour chaque valeur
    - Vecteur des pointeurs des lignes (row\_ptr)





## 2. NLP pre-processing : Matrice creuse

---

- Matrice creuse
  - Exercice : passer format CSR à matrice creuse

```
rowptr:      ( 0 4 7 10 12 14 16 )
               |  |  |  |  |  |  |
colind: ( 0 1 2 3 0 1 2 0 1 2 0 3 4 5 4 5 )
val:    ( 7.5 2.9 2.8 2.7 6.8 5.7 3.8 2.4 6.2 3.2 9.7 2.3 5.8 5.0 6.6 8.1 )
```



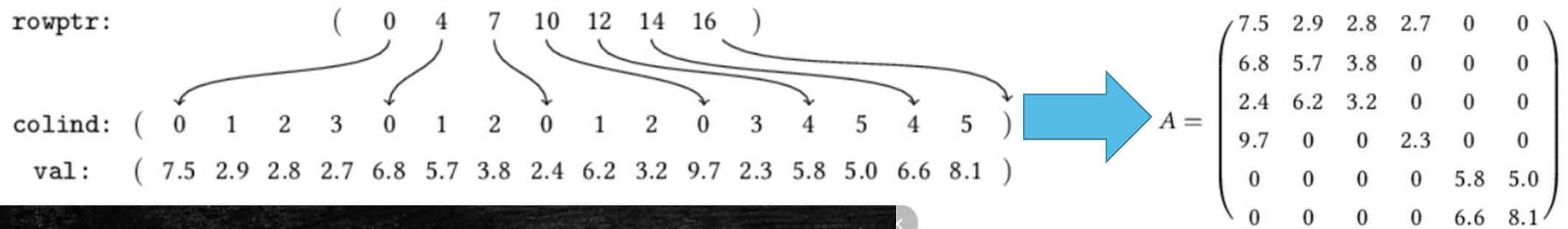
Matrice ?

## 2. NLP pre-processing : Matrice creuse

- Matrice creuse
  - Exercice : passer format CSR à matrice creuse

Figure 1 - uploaded by [Athena Elafrou](#)  
Content may be subject to copyright.

Download





## 2. NLP pre-processing : Matrice creuse

---

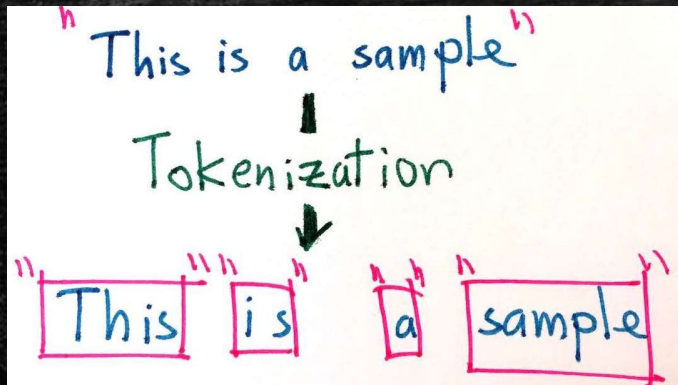
- Matrice creuse
- Intérêt :
  - utile dans les applications d'apprentissage automatique - par ex. fractionnement entre échantillon apprentissage et test
  - tokenisation des caractéristiques du texte (NLP)
  - multiplication d'une matrice par un vecteur de coefficients
  - calcul d'un gradient observation par observation, entre autres
- De nombreuses techniques et bibliothèques d'optimisation stochastique (par exemple LibSVM, VowpalWabbit) nécessitent que les entrées soient au format CSR ou similaire (voir aussi readsparse). Grande différence de performances.



### 3. Prétraitements des données

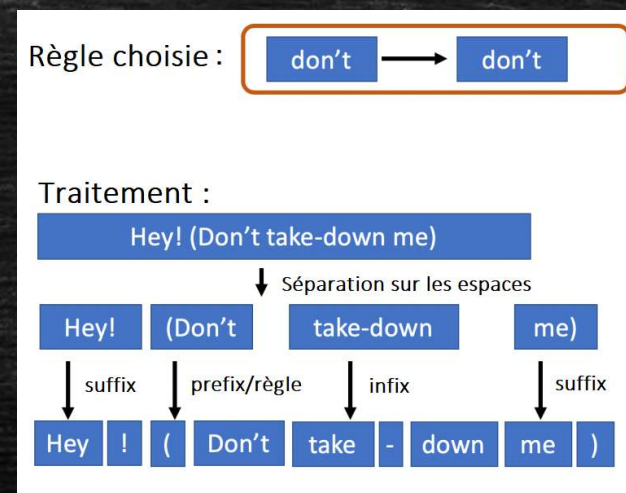
#### 1. Tokenization

- Découpage des phrases en éléments (token)
- Ex :



<https://medium.com/data-science-in-your-pocket/tokenization-algorithms-in-natural-language-processing-nlp-1fceb8454af>

- Il existe des méthodes de segmentation de mots et de suppression de mots (stopwords)
- Permet de distinguer les phrases



NB : attention aux « Tricky cases »

- ponctuation (, . « », ?...)
- Dates
- nombres

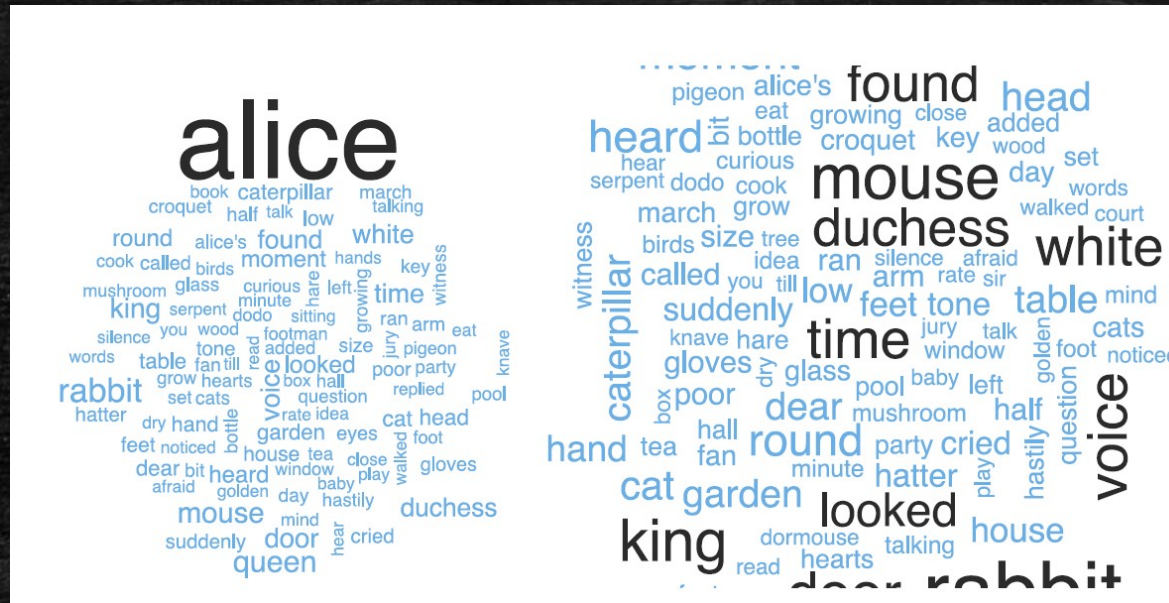
<https://lbourdois.github.io/blog/nlp/Les-tokenizers/>



### 3. Prétraitements des données

## 1. Tokenization

- Importance de la gestion des mots (ex: stopwords)
- Représentation nuage de mots



## Différence significative avant et après gestion des mots

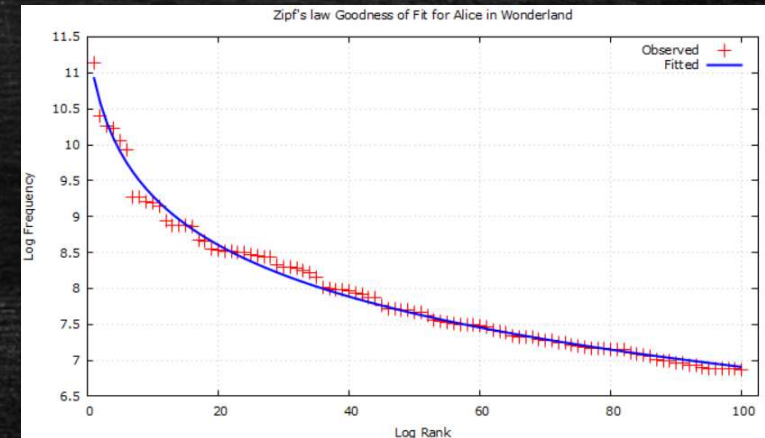


# 3. Prétraitements des données

## 1. Tokenization

- Plus un mot est fréquent, moins il est important → Loi de Zipf

Loi de Zipf construite sur le livre d'Alice au pays des merveilles (L. Carroll)



- Suppression des mots les plus fréquents (Thank you, Hello,...)
  - Bibliothèques NLTK (langue anglaise), spaCy en Python, packages TM , OpenNLP, spaCyR en R
  - FLEMM (langue française)

<https://github.com/fiamm/Flemmv31>



# 3. Prétraitements des données

## 1. Tokenization

- Récupération de motifs compatibles avec un pattern d'entrée (Regular Expression = RegEx)
- Utile pour visualiser, supprimer....

### Formulaire RegEx

[abc]	A single character of: a, b, or c	.	Any single character	(...)	Capture everything enclosed
[^abc]	Any single character except: a, b, or c	\s	Any whitespace character	(a b)	a or b
[a-z]	Any single character in the range a-z	\S	Any non-whitespace character	a?	Zero or one of a
[a-zA-Z]	Any single character in the range a-z or A-Z	\d	Any digit	a*	Zero or more of a
^	Start of line	\D	Any non-digit	a+	One or more of a
\$	End of line	\w	Any word character (letter, number, underscore)	a{3}	Exactly 3 of a
\A	Start of string	\W	Any non-word character	a{3,}	3 or more of a
\Z	End of string	\b	Any word boundary	a{3,6}	Between 3 and 6 of a

<https://docs.appcues.com/article/514-regex-help>

[https://fr.wikipedia.org/wiki/Expression\\_r%C3%A9guli%C3%A8re](https://fr.wikipedia.org/wiki/Expression_r%C3%A9guli%C3%A8re)

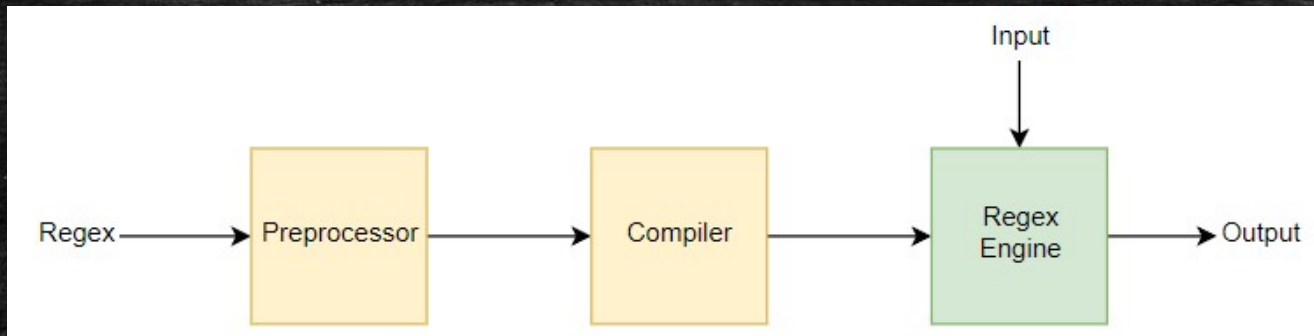
### 3. Prétraitements des données

#### 1. Tokenization

- Différents moteurs d'interprétation des scripts en RegEx selon les langages et logiciels :

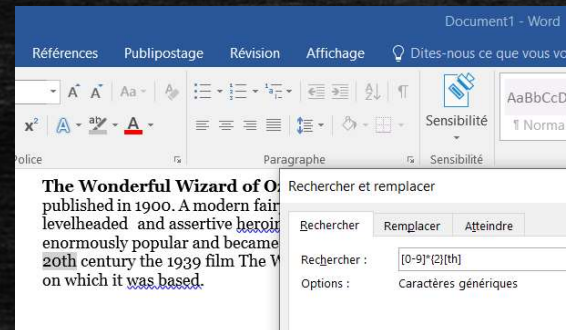
String

Instruction



Exécution du code compilé (instruction par instruction) par le moteur pour chaque input

- Python : Ruby
- R : POSIX, PERL
- Office Microsoft : VBScript
- ...



<https://devopedia.org/regex-engines>



### 3. Prétraitements des données

#### 2. Stemming (racinisation)

- Distinction de la racine de mots et regroupement en un seul mot

- Ex :

- Division, diviseur → divis
- Étude, étudiant → étud
- ...

Stemming =regroupement de variantes de mots en un seul et même mot (racine, forme non canonique )

- Analyse morphologique des mots

- Pour la langue anglaise, algorithme de Porter (1980) basée sur des règles morphologiques

(F)	Rule		Example
	SSSES	→ SS	caresses → caress
	IES	→ I	ponies → poni
	SS	→ SS	caress → caress
	S	→	cats → cat



### 3. Prétraitements des données

---

#### 2. Stemming (racinisation)

- Comparaison de 3 algorithmes différents de « stemming » en langue anglaise

Texte original

*Sample text:* Such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation

3 versions de  
stemming (Porter  
stemmer très  
populaire)

*Lovins stemmer:* such an analys can reve featur that ar not eas vis from th vari in th individu gen and can lead to a pictur of expres that is mor biolog transpar and acces to interpres

*Porter stemmer:* such an analysi can reveal featur that ar not easili visibl from the variat in the individu gene and can lead to a pictur of express that is more biolog transpar and access to interpret

*Paice stemmer:* such an analys can rev feat that are not easy vis from the vary in the individ gen and can lead to a pict of express that is mor biolog transp and access to interpret

**Figure 2.8:** A comparison of three stemming algorithms on a sample text.



## 3. Prétraitements des données

---

### 3. Lemmatization (lemmatisation)

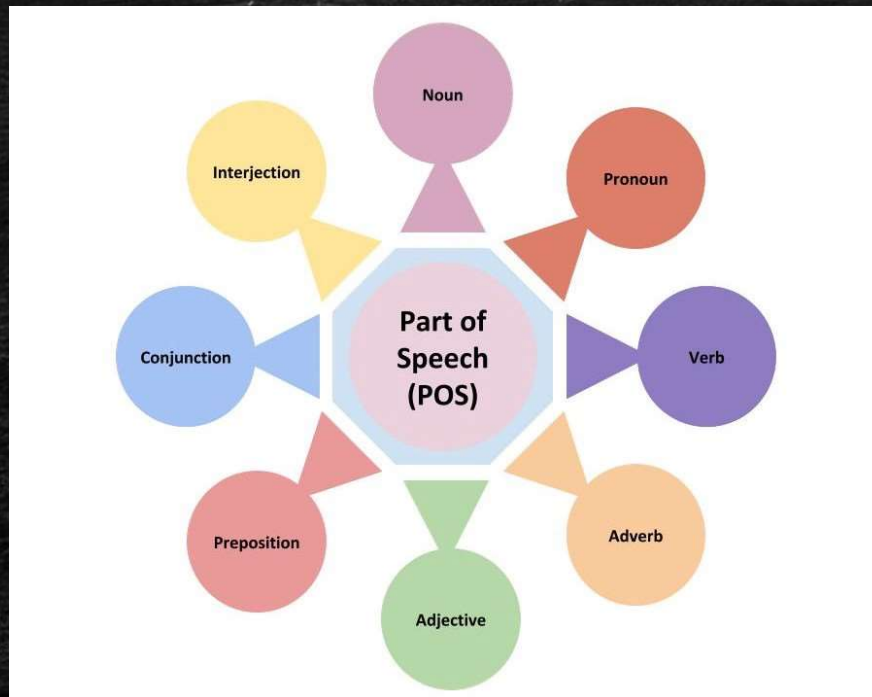
- ramener un mot à sa forme la plus simple quels que soient son accord, sa déclinaison, son genre...
- Ex :
  - Suis → être
  - sommes → être
  - Avons été → être
  - Serons → être
  - .....
- Choix entre stemming et lemmatization : dépend de la langue. Par exemple : lemmatisation marche bien pour l'Allemand
- Choix privilégié par librairies de calcul (Ex : spaCy)

lemmatization = forme canonique correcte la plus simple d'un mot

## 4. Compréhension structure phrase

### 1. Part Of Speech (POS) Tagging

- Définition des mots et du contexte dans une phrase



Regroupement en 9 catégories :

- Noun (N) — Pierre, Prais, Bonheur...
- Pronoun (PRO) — Je, Nous, On, Lui, leur
- Verb (V) — Lire, aimer, mange, a , est
- Adverb (ADV) — Lentement, toujours....
- Adjective (ADJ) — Rapide, Heureux, Gentil....
- Preposition (P) — Sur, dans, avec, entre...
- Conjunction (CON) — ET, où, donc, car...
- Interjection (INT) — Oh, Hé, Waouh...

<https://parts-of-speech.info/>



## 4. Compréhension structure phrase

### 1. Part Of Speech (POS) Tagging

– Exemple :

Une phrase

```
[ ] phrase = ("Le sorcier habite dans un château. \n Guillaume est l'apprenti du sorcier.")  
    doc = nlp(phrase) #création d'un objet doc
```

```
[ ] [mot.text for mot in doc]  
    # Tokenization  
    def return_token(phrase):  
        doc = nlp(phrase)  
        return [mot.text for mot in doc]  
    # Part Of Speech  
    def return_pos(phrase):  
        doc = nlp(phrase)  
        return [mot.pos_ for mot in doc]  
    # Dependance syntaxique  
    def return_dep(phrase):  
        doc = nlp(phrase)  
        return [mot.dep_ for mot in doc]
```

POS et  
liaison entre  
les tokens

## 4. Compréhension structure phrase

### 1. Part Of Speech (POS) Tagging

– Exemple (suite) :

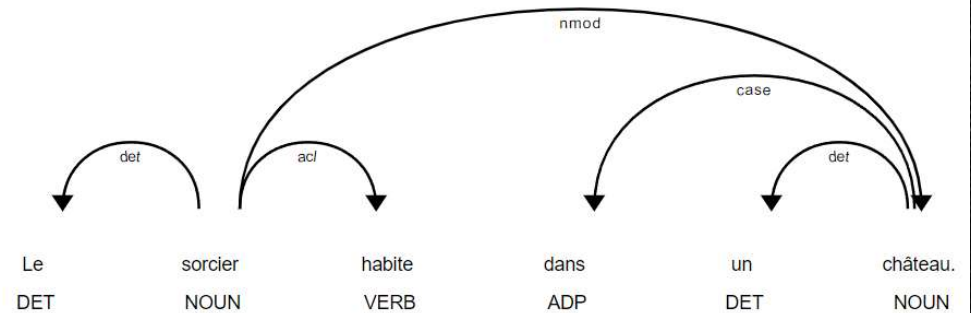
```
[ ] print(return_token(phrase))  
    print(return_pos(phrase))  
    print(return_dep(phrase))
```

```
['Le', 'sorcier', 'habite', 'dans', 'un', 'château', '.']  
['DET', 'NOUN', 'VERB', 'ADP', 'DET', 'NOUN', 'PUNCT']  
['det', 'ROOT', 'acl', 'case', 'det', 'nmod', 'punct']
```

Dépendance pas très claire à interpréter

Dépendance plus facile à visualiser sous forme graphique

```
from spacy import displacy  
displacy.render(doc, style='dep', jupyter=True, options={'distance': 130})
```

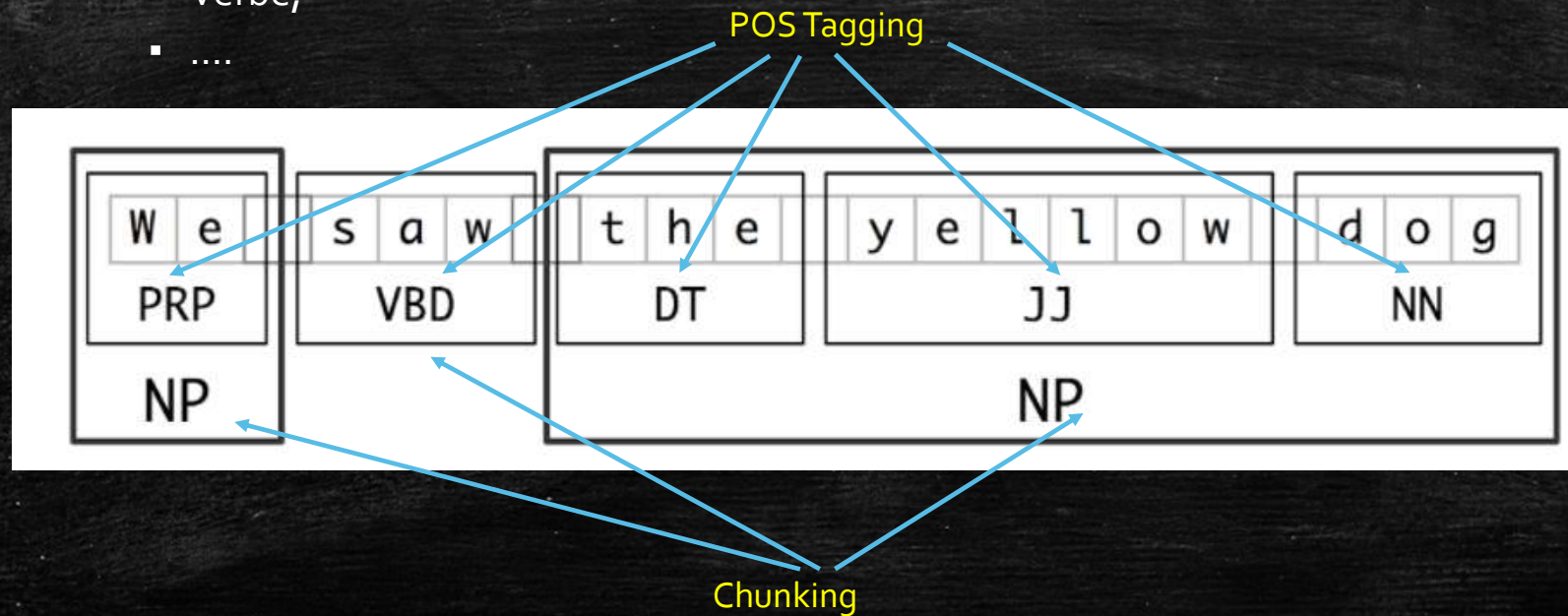




## 4. Compréhension structure phrase

### 2. Chunking

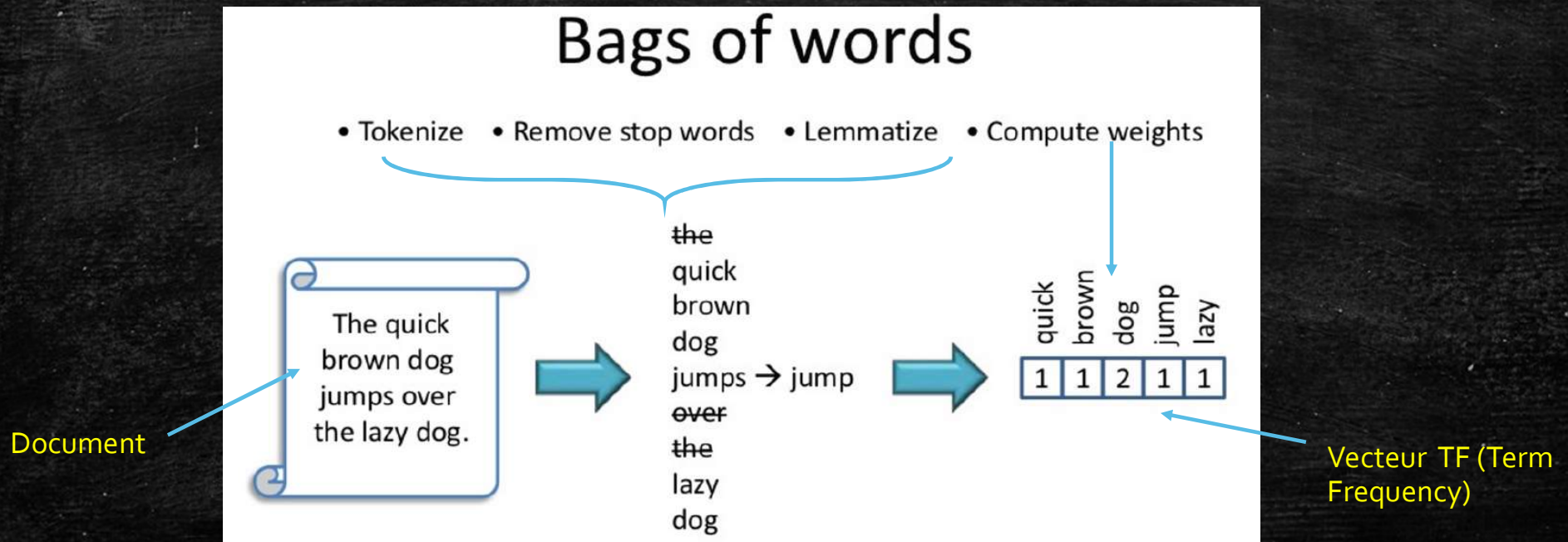
- Processus d'identification des constituants syntaxiques d'une phrase
  - Nom,
  - Verbe,
  - ....



## 5. Vectorisation des données

### 1. Bag Of Words (BOW)

- Représentation d'un document par un vecteur où chaque mot est caractérisé par sa fréquence d'apparition





## 5. Vectorisation des données

### 1. Bag Of Words (BOW)

- Calcul sur plusieurs documents → DTM (Document-Term Matrix) après stemming ou lemmatisation

Terms

	◆ text ▼	◆ analysis ▼	◆ is ▼	◆ fun ▼	◆ I ▼	◆ like ▼
<b>Text analysis is fun</b>	1	1	1	1	0	0
<b>I like doing text analysis</b>	1	1	0	0	1	1
<b>I like puppies, they are fun</b>	0	0	0	1	1	1
<b>I like this blog post</b>	0	0	0	0	1	1

Documents

Frequency  
(calcul des poids)

## 5. Vectorisation des données

### 1. Bag Of Words

- Autre calcul des poids → matrice TF-IDF (Term Frequency – Inverse Document Frequency)

Fréquence des mots du document 2

Fréquence du mot dans l'ensemble des documents Ex : 2 pour Jump

Document 1

jump

Document 2

The quick  
brown dog  
jumps over  
the lazy dog.

	TF	DF	IDF	TFIDF
quick	1	1	0.69	0.69
brown	1	1	0.69	0.69
dog	2	1	0.69	1.39
jump	1	2	0	0
lazy	1	1	0.69	0.69

$$1 * \ln(2/1)$$

$$2 * \ln(2/1)$$

$$1 * \ln(2/2)$$

$$TFIDF = TF \times IDF$$
$$IDF = \log_e \frac{|D|}{DF}$$
$$|D| = 2$$

Nombre de documents



## 5. Vectorisation des données

---


### 1. Bag Of Words

- Avantages :
  - Simplicité et facile à utiliser
  - matrice DTM (TF, TF-IDF) facile à calculer
  - Matrice utilisable pour prédire le classement de documents ou donner une annotation
- Inconvénients :
  - Temps de calcul
  - BOW = Matrice creuse de grande dimension
  - Ne prend en compte l'ordre des mots
  - Relation sémantique entre les mots liée à la dimension des vecteurs de mots (curse of dimensionality)

## 5. Vectorisation des données

---

### 1. Bag Of Words

- Alternative : Word Embedding
  - Mots représentés par des vecteurs de longueur fixe
  - Capturer des régularités sémantiques et syntaxiques du langage à partir de grands ensembles de documents non supervisés, tels que Wikipedia
  - Les mots qui apparaissent dans le même contexte sont représentés par des vecteurs proches les uns des autres (cosine similarity)
  - Différents algorithmes :
    - Word2Vec 
    - Glove
    - fastText

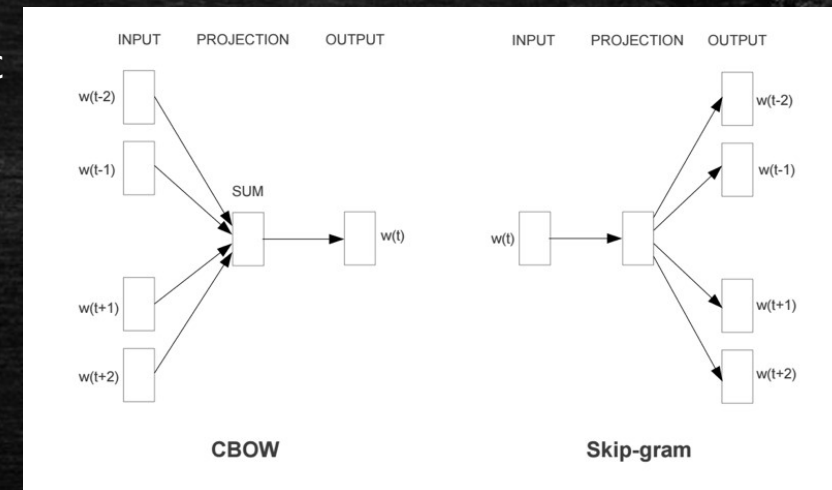


# 5. Vectorisation des données

## 2. Word2Vec

- Idée : prédire les mots entourant chaque mot dans une phrase
- Plus de comptage des occurrences des mots
- Mots représentés par des vecteurs, apprendre à partir de l'espace de dimensions des mots
- 2 architectures proposées par auteurs de Word2vec
  - CBOW
  - Skip-Gram

<https://proceedings.neurips.cc/paper/2013/file/gaa42b31882ec039965f3c4923ce901b-Paper.pdf>

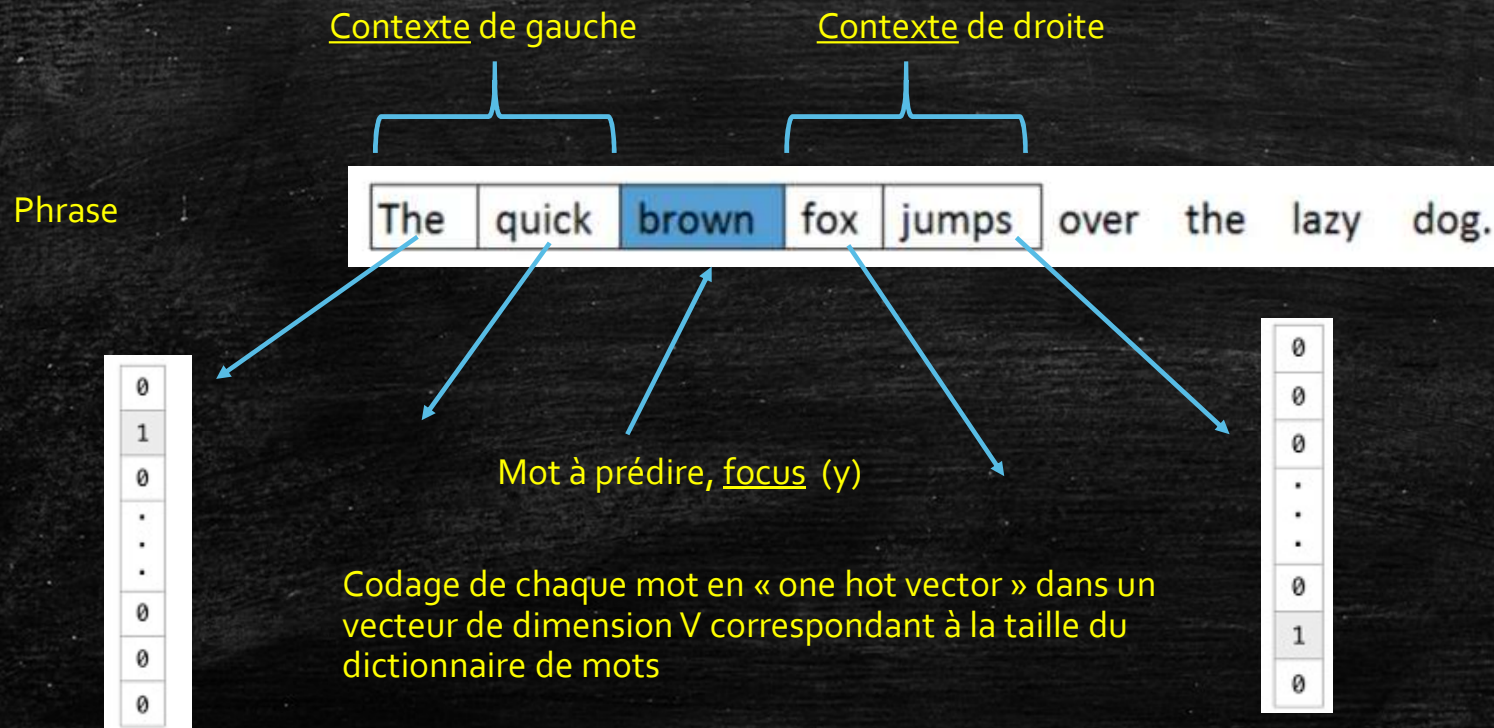


## 5. Vectorisation des données

### 2. Word2Vec

- Architecture CBOW (Continuous Bag Of Words)

A quick brown fox jumps over the lazy dog





## 5. Vectorisation des données

### 2. Word2Vec

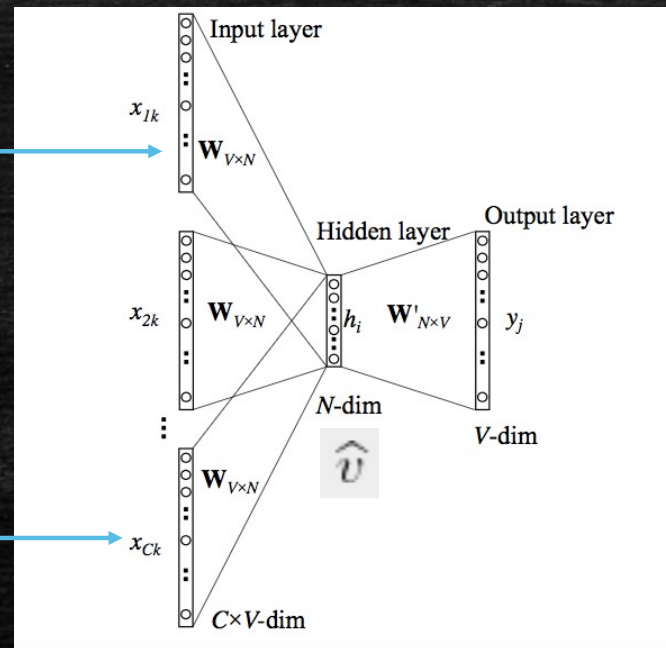
- Architecture CBOW (Continuous Bag Of Words)

Phrase

The quick brown fox jumps over the lazy dog.

0  
1  
0  
.  
.  
0  
0  
0

0  
0  
0  
.  
.  
0  
1  
0



Hidden layer

- Pour chaque vecteur de mot  $x_{jk}$  (dimension V), calcul de  $v_{jk} = W^t * x_{jk}$
- Vecteur de couche cachée:  $\hat{u}$

Fonction d'activation identité

Output Layer

- Fonction softmax appliquée

$$\hat{y}_j = \frac{\exp_j^{W'^t * \hat{u}}}{\sum_{k=1}^{|V|} \exp_k^{W'^t * \hat{u}}}$$

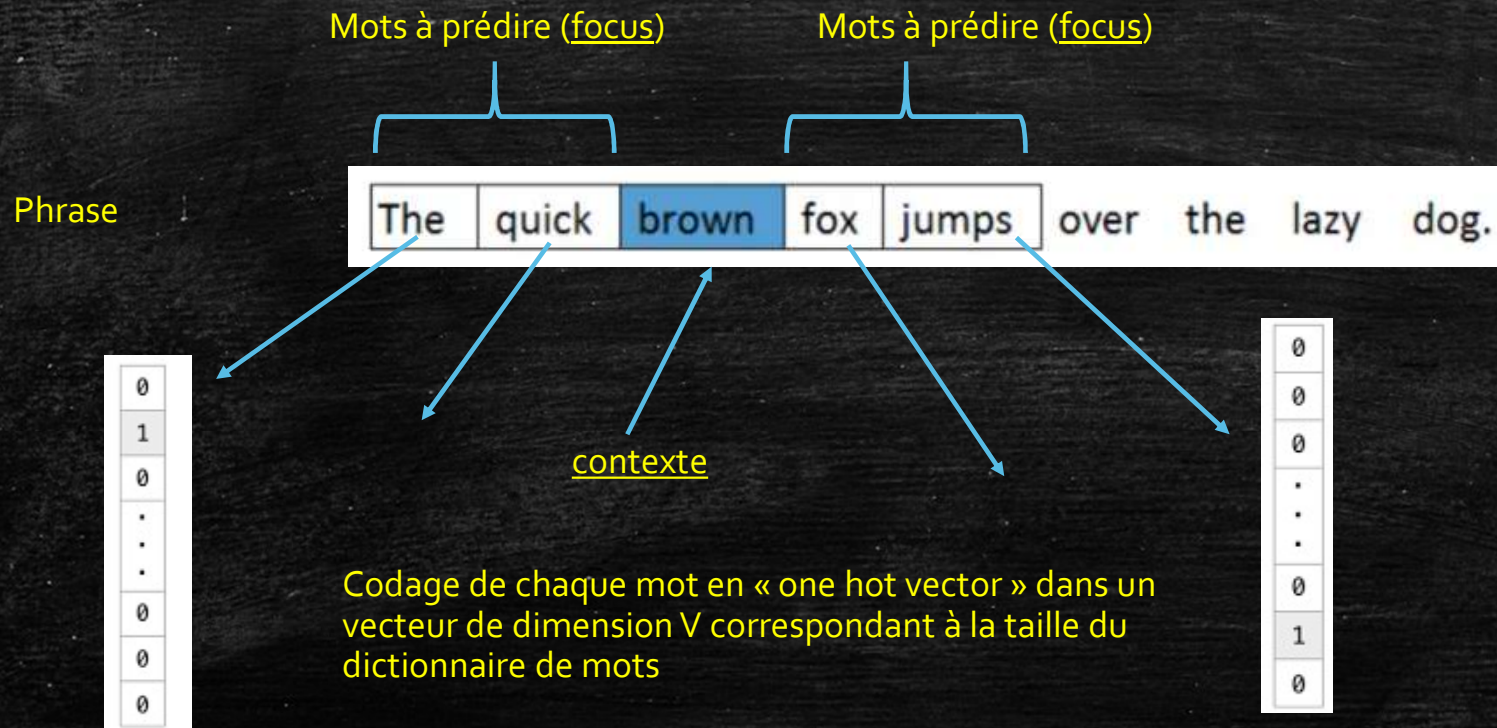
Détermination des matrices W et W' en minimisant la fonction perte (cross entropy) avec méthode du gradient

## 5. Vectorisation des données

### 2. Word2Vec

- Architecture Skip-Gram

A quick brown fox jumps over the lazy dog

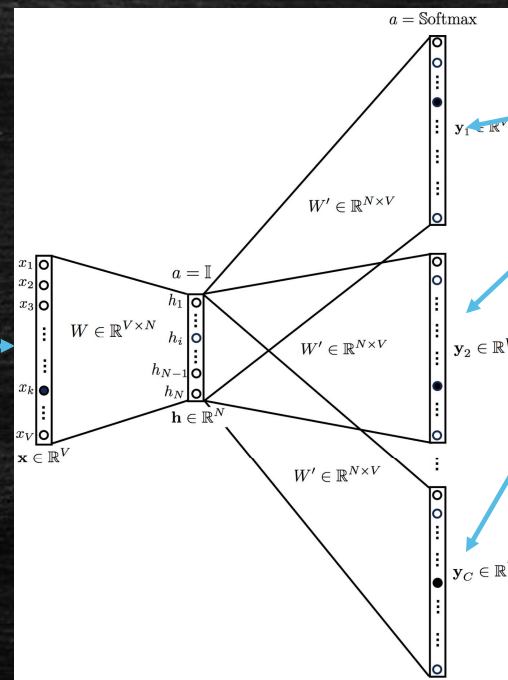
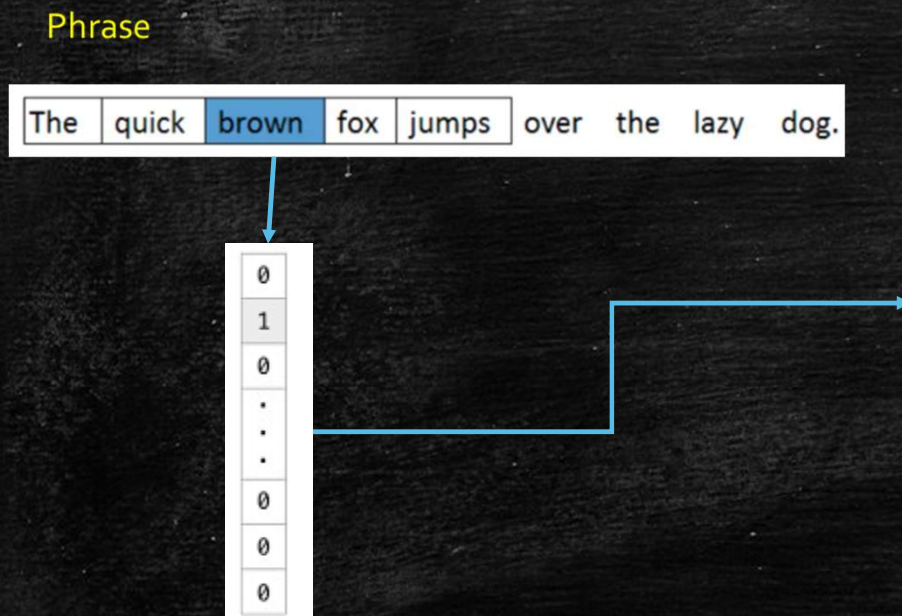




## 5. Vectorisation des données

### 2. Word2Vec

#### – Architecture Skip-Gram



Mots à prédire :

The  
Quick  
...  
jumps

Même propriété du réseau de neurones que pour l'architecture CBOW :

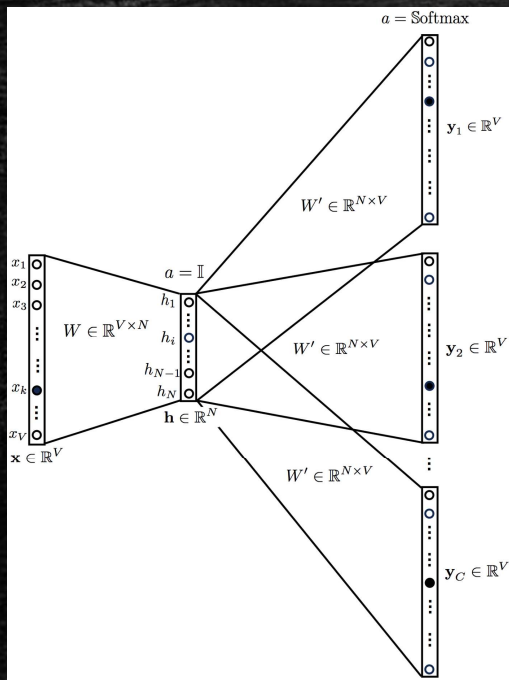
- Fonction d'activation identité pour couche cachée
- Fonction d'activation softmax pour couche de sortie

Détermination des matrices  $W$  et  $W'$  en minimisant la fonction perte (cross entropy) avec méthode du gradient

# 5. Vectorisation des données

## 2. Word2Vec

### – Architecture Skip-Gram



Mots d'apprentissage

$w_1, w_2, w_3, \dots, w_T$



Maximisation de la moyenne de la log-probabilité

$$\frac{1}{T} \sum_{t=1}^T \sum_{-c \leq j \leq c, j \neq 0} \log p(w_{t+j} | w_t)$$



Détermination des matrices  $W$  et  $W'$



## 5. Vectorisation des données

---

### 3. Comparaison architectures proposées par Word2Vec

#### – CBOW

- Prédiction d'un « focus » à partir de plusieurs contextes (2,4,...)
- Détermination de la probabilité d'occurrence la plus grande d'un mot
- Plusieurs contextes pour un seul mot → plus grande base d'apprentissage de fait
- Apprentissage plus rapide et plus rapide en temps de calcul

#### – Skip-Gram

- Prédiction des contextes (2,4,...) à partir d'un seul focus
- Comparaison de différents contextes d'apparition séparément
- Besoin de plus de données d'apprentissage pour apprendre de l'information sur les contextes
- Plus lent en temps de calcul, nécessite plus d'apprentissage

## 5. Vectorisation des données

3. Comparaison architectures proposées par Word2Vec
  - Exemple de comparaison en temps de calcul

Neural Architecture	Time to Train
SkipGramSI (subword)	19 min 37s
SkipGram	16 min 31s
CBOW	3 min 41s

kavita-ganesan.com

<https://kavita-ganesan.com/comparison-between-cbow-skipgram-subword/#.YiCL1trMJPY>

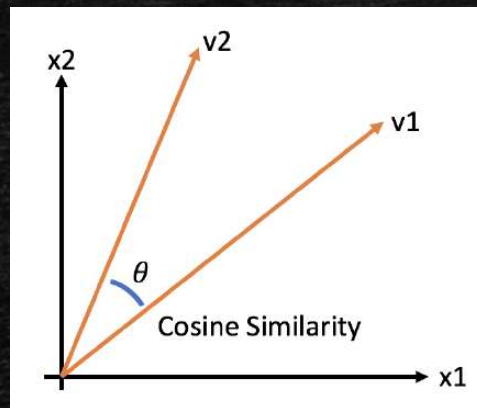
- Résultats bons dans l'ensemble, approche différente



## 5. Vectorisation des données

### 4. Calcul de similarités

- Cosine similarity : mesure de similarité entre 2 vecteurs de mots
- Angle formé par 2 vecteurs (produit vectoriel de 2 vecteurs)
- Valeurs comprises entre -1 et 1 :
  - -1 : vecteurs opposés, 0 : orthogonaux, 1 : colinéaires (similaires)



$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$

# 5. Vectorisation des données

## 4. Calcul de similarités

- Différentes applications

### Words similarity

#### Word similarities

```
doc = nlp("happy joyous sad")
for token1 in doc:
    for token2 in doc:
        print(token1.text, token2.text, token1.similarity(token2))
```

```
happy happy 1.0
happy joyous 0.63244456
happy sad 0.37338886
joyous happy 0.63244456
joyous joyous 1.0
joyous sad 0.5340932
...
```

### Document similarity

#### Document similarities

```
# Generate doc objects
sent1 = nlp("I am happy")
sent2 = nlp("I am sad")
sent3 = nlp("I am joyous")
```

```
# Compute similarity between sent1 and sent2
sent1.similarity(sent2)
```

```
0.9273363837282105
```

```
# Compute similarity between sent1 and sent3
sent1.similarity(sent3)
```

```
0.9403554938594568
```

### Système de recommandation

#### The get\_recommendations function

```
get_recommendations('The Lion King', cosine_sim, indices)
```

```
7782          African Cats
5877  The Lion King 2: Simba's Pride
4524          Born Free
2719          The Bear
4770  Once Upon a Time in China III
7070          Crows Zero
739          The Wizard of Oz
8926          The Jungle Book
1749  Shadow of a Doubt
7993          October Baby
Name: title, dtype: object
```